ФГБОУ ВПО «Воронежский государственный
технический университет»

А.В. Строгонов

СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ ИНТЕГРАЛЬНЫХ СХЕМ

Утверждено Редакционно-издательским советом
университета в качестве учебного пособия

Воронеж 2012

УДК 621.3.049.77

Строгонов А.В. Системное проектирование программируемых логических интегральных схем: учеб. пособие / А.В. Строгонов. Воронеж: ФГБОУ ВПО «Воронежский государственный технический университет», 2012. 322 с.

В учебном пособии рассматриваются различные архитектуры и структуры межсоединений ПЛИС, трехмерные БИС. Даются практические примеры проектирования ПЛИС как на системном уровне с привлечением системы визуально-имитационного моделирования аналоговых и дискретных систем Matlab/Simulink, так и на функциональном уровнях с использованием высокоуровневого языка описания аппаратурных средств VHDL в САПР ПЛИС Quartus II компании Altera. Уделено внимание вопросам проектирования функциональных узлов микропроцессорных устройств для реализации в базисе ПЛИС.

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего профессионального образования по направлению подготовки бакалавров 210100 «Электроника и наноэлектроника», профилю «Микроэлектроника и твердотельная электроника», дисциплинам «Проектирование БИС», «Проектирование ПЛИС», «Проектирование цифровых устройств в базисе ПЛИС». Учебное пособие подготовлено в электронном виде в текстовом редакторе MS Word for Windows и содержится в файле Сист проектир ПЛИС.doc.

Табл. 14. Ил. 155. Библиогр.: 21 назв.

Научный редактор д-р физ.-мат. наук, проф. С.И. Рембеза

Рецензенты: кафедра физики полупроводников и микроэлектроники Воронежского государственного университета (зав. кафедрой д-р физ.-мат. наук, проф. Е.Н. Бормонтов);
д-р техн. наук, проф. М.И. Горлов

ВВЕДЕНИЕ

ПЛИС – цифровые БИС высокой степени интеграции, имеющие программируемую пользователем внутреннюю структуру и предназначенные для реализации сложных цифровых устройств. Использование ПЛИС и САПР позволяет в сжатые сроки создавать конкурентоспособные устройства и системы, удовлетворяющие жестким требованиям по производительности, энергопотреблению, надежности, массо-габаритным параметрам, стоимости.

ПЛИС широко используются в качестве интерфейсных схем, в микропроцессорных системах для организации обмена и стыковки различных ИС между собой и устройствами ввода-вывода. В базисе ПЛИС могут быть спроектированы логические блоки и системы, преобразователи кодов, периферийные контроллеры, микропрограммные устройства управления, конечные автоматы, а также другие специализированные устройства типа умножителей, небольших процессоров (микропроцессорные ядра), процессоров быстрого преобразования Фурье и др.

В первой главе рассматриваются академические и индустриальные ПЛИС с одноуровневой и многоуровневой структурами межсоединений, технологии соединений трассировочных ресурсов, программные инструменты проектирования ПЛИС. Во второй главе рассмотрены трехмерные БИС и стековые 3D БИС. Освещены вопросы, связанные с проектированием БИС по субмикронным проектным нормам. Показаны примеры проектирования ПЛИС с использованием новейших достижений нанотехнологии: углеродные нанотрубки в качестве

реконфигурационной памяти и мемристорные структуры (нанокоммутаторы).

В третьей главе основное внимание уделено разработке функциональной модели ПЛИС типа ППВМ с одноуровневой структурой межсоединений в САПР Quartus II с использованием технологии соединений multi-driver и имитационной модели с использованием технологии соединений single-driver в системе визуально-имитационного моделирования Matlab/Simulink.

В четвертой главе рассмотрены различные подходы в проектировании микропроцессорных ядер для реализации в базисе ПЛИС с использованием системы визуально-имитационного моделирования Matlab/Simulink с приложениями StateFlow и Simulink HDL Coder. Микропроцессорные ядра, представленные в виде сложно-функциональных блоков в базисе ПЛИС, позволяют реализовать современную концепцию “система на кристалле”. Использование более высокой степени абстракции в проектировании БИС и сложно-функциональных блоков в виде готовых модулей позволяют создавать конкурентоспособные изделия в кратчайшие сроки.

1. ЦИФРОВЫЕ БИС

1.1. Специализированные БИС

Термин специализированные интегральные схемы (ИС) конкретного применения (Application Specific Integrated Circuits (ASIC)) охватывает, по классификации фирмы Altera, две приведённые ниже технологии проектирования.

Заказной БИС (Full Custom Large Integrated Circuits) называют однокристальное устройство, которое проектируют полностью “с пустого места” - без предварительной подготовки базовых технологических слоев, специально сконструированных элементов и функциональных макроблоков. Основное отличие от других технологий проектирования: необходимо проектировать единые маски для всех технологических слоев.

Полузаказные матричные БИС проектируют на основе БМК - базовых матричных кристаллов (Standart Cells – стандартные ячейки и Gate Arrays – вентильные матрицы). Логика развития технологии матричных БИС делает их экономически оправданными только в сочетании с системами автоматизированного проектирования (САПР) и при условии, что полный цикл проектирования можно провести на столе разработчика без итераций физического моделирования. Для этого САПР должна содержать систему моделирования высокой адекватности, с помощью которой можно было бы решить все проблемы работоспособности готового изделия до того, как результат проектирования будет передан на технологическую линию.

В мире интерес к ASIC достаточно велик, и год от года он продолжает неуклонно расти. Отмечено активное смещение “центра тяжести” ASIC в сторону изделий системного уровня интеграции, когда в кристалл интегрируется стандартное фиксированное ядро массового применения

(микроконтроллер, драйвер ЖКИ, периферийные контроллеры, массивы памяти и т. д.).

Выпускаемые ASIC фирмы Epson содержат на кристалле более 2 млн вентилей. При их производстве используется несколько различных базовых полупроводниковых технологий. Первой из них является 0.25-мкм КМОП-процесс изготовления низковольтных схем с 6 слоями металлизации. Фирма использует в своих ASIC динамические запоминающие устройства (ЗУ), статические ЗУ (6-транзисторные ячейки), а также флэш-память. Модифицированная флэш-память может включаться в структуры ИС с напряжением питания 2,7 В, изготавливаемых по 0,35-мкм технологии, а также в микросхемы с питанием 2,3 В, изготавливаемые с помощью 0,25-мкм процесса. По выбору потребителя в ИС могут быть использованы две технологии флэш ЗУ, удовлетворяющие противоречивым требованиям: высокому быстродействию и минимальному потреблению мощности. Главная проблема интеграции флэш-памяти заключается в том, что при комбинации с аналоговыми технологическими процессами общий кристалл становится слишком дорогим.

ASIC, выпускаемые фирмой FUJITSU, имеют степень интеграции до 20 млн вентилей на кристалл. При их изготовлении используется 0,18-мкм технологический процесс с 5 слоями металлизации. При производстве массовой продукции используется технология с разрешением 0,11 мкм и 8 слоями металлизации. Предполагается также внедрение разновидности 0,11-мкм технологии с 7 слоями медных межсоединений.

Фирма Altera в отдельную группу выделяет БИС программируемой логики (программируемые логические ИС, ПЛИС). ПЛИС отличаются от матричных БИС тем, что конечный результат достигается с помощью программатора на столе разработчика: технологическая линия из цикла

проектирования исключена. Предприятия электронной промышленности поставляют “заготовки” (или “полуфабрикаты”) ПЛИС с мощными САПР высокой адекватности и средствами электрического или логического программирования (и перепрограммирования). Роль технологий проектирования “на столе разработчика” резко увеличивается, если поставщик предоставляет возможность прямой трансляции проекта на БМК, чтобы изготовить БИС без дополнительных усилий разработчиков.

ASSP (Application Specific Standard products) - это БИС для специализированных приложений массового применения. Каждая ИС обязана своим появлением какой-либо идее, задаче, приложению. Классическим примером могут служить чипсеты для материнских плат компьютеров и видеокарт. Главная особенность: ASSP- это стандартные изделия. Будучи один раз разработаны для конкретной цели, они затем производятся массовыми тиражами и могут быть использованы несколькими потребителями в разных конечных устройствах.

1.2. Программируемые логические ИС

Вследствие быстрого роста сложности электронных систем все чаще требуется применение высокоинтегрированных специализированных интегральных схем. Разработка и изготовление таких схем по заказу представляет собой длительный и дорогостоящий процесс, который экономически оправдан только при достаточно большом объеме выпуска. При малой потребности (до 10000 шт. в год) более выгодно использование стандартных “полуфабрикатов” ИС, специализируемых в сфере потребления после их изготовления. Поскольку для всех ИС используются одни и те же фотошаблоны, то для изготовителя

эти ИС являются стандартными изделиями. К числу таких изделий микроэлектроники относятся программируемые пользователем логические ИС (ПЛИС) или программируемые логические приборы — ПЛП (PLD — Programmable Logic Devices). Программирование ПЛИС осуществляется самим пользователем, конструктором аппаратуры. В результате программирования в схему вносятся обратимые или необратимые (с точки зрения возможности последующего перепрограммирования) изменения исходной структуры ПЛИС.

ПЛИС - цифровые БИС высокой степени интеграции, имеющие программируемую пользователем внутреннюю структуру и предназначенные для реализации сложных цифровых устройств. Использование ПЛИС и соответствующих средств автоматизации проектирования позволяет в сжатые сроки создавать конкурентноспособные устройства и системы, удовлетворяющие жестким требованиям по производительности, энергопотреблению, надежности, массо-габаритным параметрам, стоимости.

Основное преимущество ПЛИС перед другими специализированными схемами — малое время изготовления требуемых заказных вариантов схем. Изделие в готовом виде всегда имеется на складе, и нет необходимости обращаться к изготовителям ИС для нанесения металлической маски и установки кристалла в корпус. Достаточно включить соответствующие средства программирования и через несколько секунд или минут заказная схема будет готова. Из-за простоты и доступности процесса специализации программаторы ПЛИС иногда называют “фабрикой на столе”.

ПЛИС широко используются в качестве интерфейсных схем, в микропроцессорных системах для организации обмена и стыковки различных ИС между собой и устройствами ввода-вывода. На базе ПЛИС могут быть изготовлены логические блоки и системы, преобразователи кодов, периферийные

контроллеры, микропрограммные устройства управления, конечные автоматы, а также другие специализированные устройства типа умножителей, небольших процессоров и процессоров быстрого преобразования Фурье.

Исторически технология и физические принципы программирования ПЛИС повторяют путь, пройденный ИС запоминающих устройств (ЗУ) с изменяемой конфигурацией. Создание ПЛИС было стимулировано необходимостью сокращения разрыва в степени интеграции между ИС ЗУ и произвольной логикой. Поэтому и возникла идея построения ПЛИС по принципу, аналогичному ЗУ. Эти приборы должны были выполнять функции произвольной логики со многими переменными и использоваться там, где применение для этих целей ЗУ неэффективно.

Первые ПЛИС были изготовлены по биполярной технологии и как программируемые постоянные ЗУ (ППЗУ) программировались пережиганием плавких перемычек. Затем появилась КМОП-технология ПЛИС с плавкими перемычками, были созданы репрограммируемые ПЛИС с ультрафиолетовым и электрическим стиранием записанных логических функций (СПЛИС и ЭСПЛИС), использующие технологию репрограммируемых ППЗУ. Наконец, созданы ПЛИС, изготовленные по технологии КМОП статических оперативных запоминающих устройств (СОЗУ). Современные КМОП ПЛИС изготавливаются по КМОП-технологии 25-180 нм с использованием 3D-интеграции. Фирма Xilinx разработала новейшую серию ПЛИС Virtex-7 с функциональной емкостью 2 млн. эквивалентных вентилей с использованием интеграции на уровне коммутационной Si-пластины и 28 нм КМОП-технологического процесса.

Программируемые логические матрицы (ПЛМ) - наиболее традиционный тип ПЛИС, имеющий программируемые матрицы “И” и “ИЛИ”. В зарубежной литературе соответствующими этому классу аббревиатурами

являются FPLA (Field Programmable Logic Array) и FPLS (Field Programmable Logic Sequensers). Примерами таких ПЛИС могут служить отечественные схемы К556РТ1, РТ2, РТ21. Недостаток такой архитектуры - слабое использование ресурсов программируемой матрицы “ИЛИ”, поэтому дальнейшее развитие получили микросхемы, построенные по архитектуре программируемой матричной логики, (ПМЛ или PAL - Programmable Array Logic) - это ПЛИС, имеющие программируемую матрицу ”И” и фиксированную матрицу “ИЛИ”. К этому классу относятся большинство современных ПЛИС небольшой степени интеграции. В качестве примеров можно привести отечественные ИС КМ1556ХП4, ХП6, ХП8, ХЛ8, ранние разработки (середина–конец 1980-х годов) ПЛИС фирм INTEL, ALTERA, AMD, LATTICE и др. Разновидностью этого класса являются ПЛИС, имеющие только одну (программируемую) матрицу “И”, например, схема 85С508 фирмы INTEL.

Упрощенно традиционные фундаментальные структуры ПЛМ представлены на рис.1.1. Оба типа ПЛИС: ПЛМ (рис.1.1, а, в) и ПМЛ (рис.1.1, б, г), имеют матрицы И и ИЛИ. Основное различие между ними заключается в том, что в ПМЛ матрица ИЛИ фиксирована, а в ПЛМ программируются обе матрицы, что обеспечивает ей большую гибкость по сравнению с ПМЛ.

При программировании ПЛИС задаются термы произведений (матрица И), и суммы произведений (матрица ИЛИ), и тем самым, — логические функции ПЛИС.

Фрагмент программируемых соединений показан на рис.1.2. Кружками обозначены плавкие перемычки или другие программируемые соединения.

Истинное и инверсное значения сигнала каждого входа через плавкие перемычки подаются на каждый вентиль матрицы И. Группы вентилей И соединяют с вентилями ИЛИ, создавая программируемую реализацию суммы произведений, к которой можно свести все логические функции.

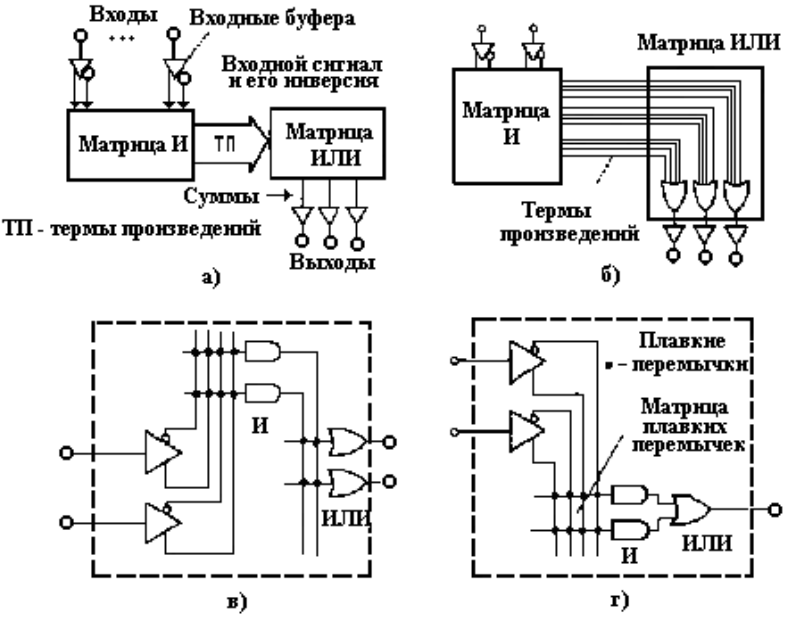


Рис.1.1. Модель и упрощенная структура ПЛИС

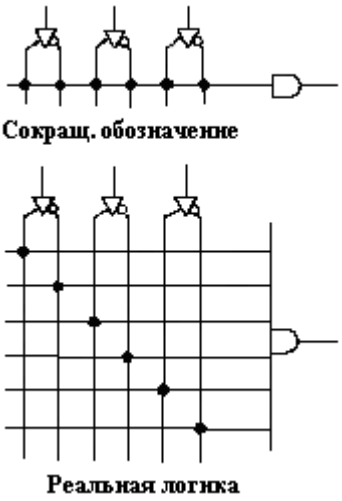


Рис.1.2. Фрагмент программируемых соединений с многовходовым вентилем И

Архитектура ПЛИС строится таким образом, чтобы на каждом имеющемся вентиле И образовывать как можно больше произведений в пределах возможности прибора. Поэтому логическую схему И называют также термом произведения. Логическое произведение затем выводится на выход через программируемую или фиксированную матрицу ИЛИ. Каждый терм произведения потенциально может использоваться для реализации логических функций и управления внутри ПЛИС.

В качестве базовой для ПЛИС была выбрана алгебра Буля, так как она наиболее изучена конструкторами.

При всей гибкости ПЛМ считаются достаточно сложными для большинства потребителей с точки зрения их проектирования. Кроме того, наличие плавких перемычек в обеих матрицах влечет за собой их относительно большие размеры по сравнению с ПМЛ и меньшее быстродействие.

Введением своей программируемой матричной логики фирма MMI упростила ПЛМ, закрепив термы произведений за специальными выходами. Наличие одной программируемой матрицы И и фиксированной ИЛИ привело к уменьшению размеров ПЛИС и времени распространения сигнала через кристалл, упрощению программирования схем. Архитектура ПМЛ завоевала наибольшую популярность у конструкторов аппаратуры. Интегральные микросхемы ПМЛ могут программироваться большинством стандартных программаторов ППЗУ с добавлением карт индивидуализации. Во время программирования одна половина выводов ПЛИС используется для программирования, а другая - для адресации. Затем выходы переключаются для программирования других элементов. При проверке используется так же процедура, причем линии программирования удерживаются в состоянии с низким уровнем.

Следующий традиционный тип ПЛИС - программируемая макрологика. Они содержат единственную

программируемую матрицу “И-НЕ” или “ИЛИ-НЕ”, но за счёт многочисленных инверсных обратных связей способны формировать сложные логические функции. К этому классу относятся, например, ПЛИС PLHS501 и PLHS502 фирмы SIGNETICS, имеющие матрицу “И-НЕ”, а также схема XL78C800 фирмы EXEL, основанная на матрице “ИЛИ-НЕ”.

Вышеперечисленные архитектуры ПЛИС содержат небольшое число ячеек, к настоящему времени морально устарели и применяются для реализации относительно простых устройств, для которых не существует готовых ИС средней степени интеграции. Естественно, для реализации алгоритмов ЦОС они не пригодны.

ИС ПМЛ (PLD) имеют архитектуру, весьма удобную для реализации цифровых автоматов. Развитие этой архитектуры — программируемые коммутируемые матричные блоки (ПКМБ) — это ПЛИС, содержащие несколько матричных логических блоков (МЛБ), объединённых коммутационной матрицей. Каждый МЛБ представляет собой структуру типа ПМЛ, то есть программируемую матрицу “И”, фиксированную матрицу “ИЛИ” и макроячейки (МЯ). ПЛИС типа ПКМБ, как правило, имеют высокую степень интеграции (до 10000 эквивалентных вентилей, до 256 макроячеек). К этому классу относятся ПЛИС семейства MAX5000 и MAX7000 фирмы ALTERA, схемы XC7000 и XC9500 фирмы XILINX, а также большое число микросхем других производителей (Atmel, Vantis, Lucent и др.). В зарубежной литературе они получили название Complex Programmable Logic Devices (CPLD).

На рис.1.3 показана архитектура блока ПЛИС типа ПКМБ (CPLD) с присущими им характерными особенностями. ПЛИС содержат триггеры для хранения сумм произведений, которые можно использовать для проектирования синхронных схем и последовательной логики, например машины состояний (цифрового автомата с памятью). МЯ имеют

программируемую полярность (ПП) выхода, или возможность выбора активно-низкого или активно-высокого выходного сигнала. Обратная связь (ОС) от выхода МЯ к матрице И позволяет использовать выход как двунаправленную линию ввода-вывода. Содержимое триггера может быть передано обратно в матрицу для создания машины состояний. Некоторые ПЛИС содержат термы произведений, которые отпирают выходные буфера (разрешение выхода). Разрешение выхода может осуществляться индивидуально и асинхронно. Термы произведений можно подавать на вход синхронизации внутренних триггеров (программируемая синхронизация), позволяя логике синхронизировать индивидуальные триггеры. Термы произведений могут управлять установкой и сбросом линий внутренних триггеров.

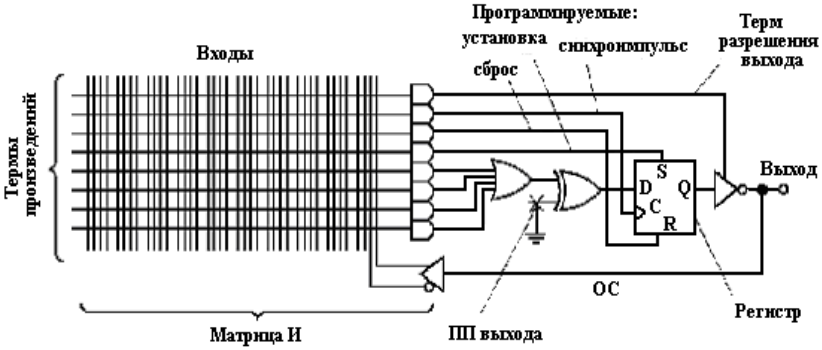


Рис.1.3. Архитектура блока (макроячейки) ПЛИС типа программируемые коммутируемые матричные блоки

Другой тип архитектуры ПЛИС — программируемые пользователем вентильные матрицы (ППВМ), состоящие из логических блоков (ЛБ) и коммутирующих путей — программируемых матриц соединений. Логические блоки таких ПЛИС состоят из одного или нескольких относительно простых логических элементов, в основе которых лежит

таблица перекодировки (ТП, Look-up table - LUT), программируемый мультиплексор, D-триггер, а также цепи управления. Таких простых элементов может быть достаточно много, например, у современных ПЛИС ёмкостью до 1 млн вентилей число логических элементов достигает нескольких десятков тысяч. За счёт такого большого числа логических элементов они содержат значительное число триггеров, а также некоторые семейства ПЛИС имеют встроенные реконфигурируемые модули памяти (embedded array block - EAB), что делает ПЛИС данной архитектуры весьма удобным средством реализации алгоритмов цифровой обработки сигналов, основными операциями в которых являются перемножение, умножение на константу, суммирование и задержка сигнала.

В зарубежной литературе такие ПЛИС получили название Field Programmable Gate Array (FPGA). К FPGA (ППВМ) классу относятся ПЛИС XC2000, XC3000, XC4000, Spartan, Virtex фирмы XILINX; ACT1, ACT2 фирмы ACTEL, а также семейства FLEX8000 фирмы ALTERA, некоторые ПЛИС Atmel и Vantis.

Существующие в настоящее время ПЛИС, выпускаемые различными производителями, имеют разнообразную архитектуру (рис.1.4). Систематизация ПЛИС производится обычно по следующим классификационным признакам: архитектура функционального преобразователя (логического блока); организация трассировочных ресурсов (структуры межсоединений); тип используемого программируемого элемента; наличие внутренней оперативной памяти; степень интеграции (логическая емкость).

Функциональные преобразователи ПЛИС включают в себя настраиваемые средства реализации логических функций и триггер. Наиболее часто логические функции реализуются или в виде суммы логических произведений (sum of product) или на 16-разрядных ПЗУ (таблицы перекодировки, LUT).

Использование этих двух подходов в ПЛИС основных фирм - производителей показано на рис.1.5. ПЛИС с функциональными преобразователями на базе сумм произведений позволяют проще реализовать сложные логические функции, а с преобразователями на базе таблиц перекодировки создавать насыщенные триггерами устройства.

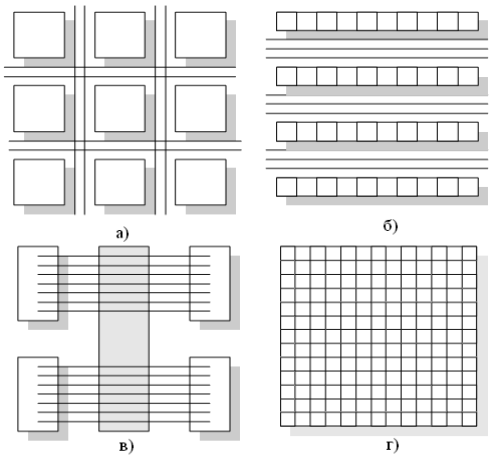


Рис.1.4. Классификация ПЛИС: а) симметричные; б) строковые; в) иерархические; г) типа “море вентилей”

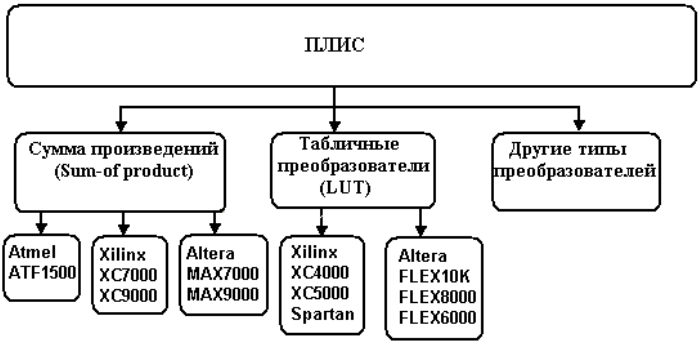


Рис.1.5. Архитектура функционального преобразователя

Различаются ПЛИС организацией внутренней структуры глобальных трассировочных ресурсов и структуры матрицы локальных межсоединений функциональных

преобразователей (рис.1.6). Большинство фирм выпускает сложные ПЛИС, располагая, функциональные преобразователи в виде квадратной матрицы на площади кристалла, при этом связи между преобразователями выполняются в виде проводников, разделенных на отдельные участки (сегменты) электронными ключами (рис.1.4, *а*). Такая одноуровневая структура получила название ППВМ. Иерархическая (многоуровневая) организация ПЛИС позволяет улучшить их технические характеристики (рис.1.4, *в* и рис.1.7). Функциональные преобразователи в этом случае группируются в блоки (например, в логический блок (кластер) ПЛИС семейств FLEX10K фирмы Altera входит 8 логических элементов), имеющие свою собственную локальную шину межсоединений. Блоки обмениваются сигналами через шины межсоединений верхнего уровня. Проводники межсоединений непрерывны (т.е. не разделены на сегменты электронными ключами), что обеспечивает малые задержки распространения сигналов и позволяет существенно сократить количество электронных ключей. Кроме того, благодаря этому свойству логические блоки можно заменять без изменения временной модели устройства, что существенно ускоряет процедуру реализации проекта и упрощает временное моделирование.

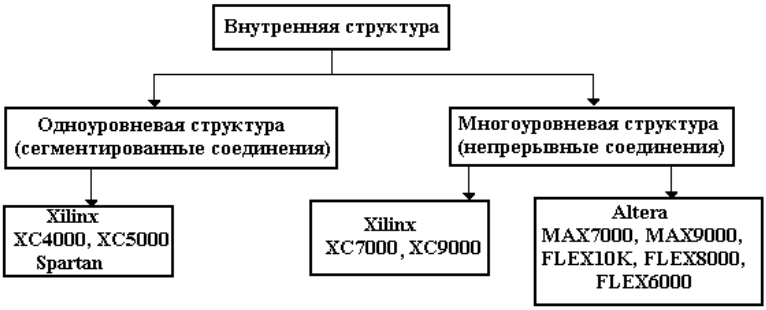


Рис.1.6. Организация внутренней структуры межсоединений в трассировочных ресурсах ПЛИС различных производителей

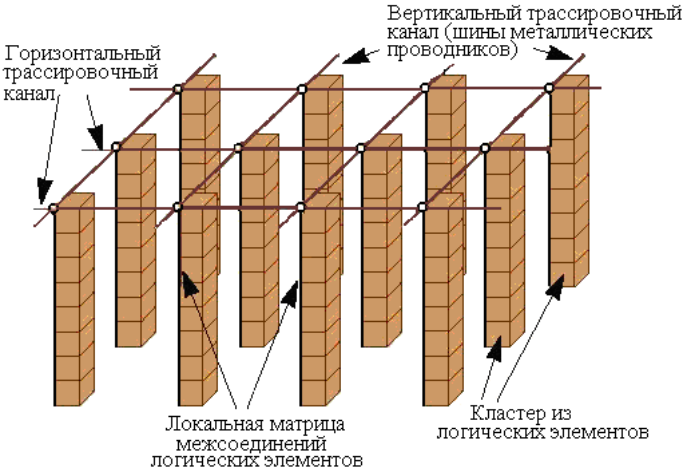


Рис.1.7. Непрерывные (несегментированные) соединения и иерархическая структура ПЛИС фирмы Altera

Тип используемого программируемого элемента - электронного ключа - определяет возможности ПЛИС по программированию, перепрограммированию и сохранению конфигурации при отключении питания. Наиболее перспективны программируемые элементы, выполненные по EEPROM- и FLASH-технологии (полевые транзисторы с плавающим затвором), обеспечивающие энергонезависимое сохранение конфигурации и многократное перепрограммирование (в том числе ИС, распаянной непосредственно на плате), а также элементы, выполненные по SRAM-технологии. Последние представляют собой электронный ключ и триггер оперативной памяти, в который при включении питания должна быть записана информация о конфигурации. SRAM-технология позволяет уменьшить энергопотребление и реконфигурировать ПЛИС за десятки миллисекунд, обеспечивая исходную загрузку конфигурирующей памяти и при необходимости - реконфигурирование “на лету” для адаптации структуры

реализуемого устройства. Особое место занимают семейства ПЛИС фирмы Actel, программируемые элементы которых - antifuse - представляют собой *pn*-переходы, пробиваемые при программировании. Эти устройства имеют повышенную способность к сохранению конфигурации при спецвоздействиях, но не получили широкого распространения в силу высокой стоимости и однократности программирования.

ПЛИС с внутренней памятью выпускаются фирмами Altera (семейство FLEX10K), Atmel (семейство AT40K), Xilinx (семейство XC4000). Организация внутренней памяти в ПЛИС различных производителей различна: в семействе FLEX10K фирмы Altera это крупные выделенные модули памяти объемом 2 кбит, в ПЛИС других производителей - распределенные по кристаллу небольшие блоки (теневые ОЗУ таблиц перекодировки объемом 32 бит в ПЛИС фирмы Xilinx и расположенные в узлах матрицы межсоединений блоки памяти объемом 32x4 бит в ПЛИС фирмы Atmel).

Степень интеграции (логическая емкость) — наиболее важная характеристика ПЛИС, по которой осуществляется ее выбор. Производители ПЛИС стоят на передовых рубежах электронной технологии, и число транзисторов в ПЛИС большой емкости составляет десятки миллионов. Но ввиду избыточности структур, включающих большое число коммутирующих транзисторов, логическую емкость измеряют в эквивалентных логических вентилях типа 2И-НЕ (2ИЛИ-НЕ), которые понадобились бы для реализации устройств той же сложности, что и на соответствующих СБИС.

1.3. Академические ПЛИС

Большинство коммерческих архитектур ПЛИС типа ППВМ (программируемые пользователем вентильные матрицы, FPGA) по технологии СОЗУ имеет одноуровневую структуру, когда логические блоки окружены с четырех сторон межсоединениями горизонтальных и вертикальных трассировочных каналов, равномерно распределенных по всей площади кристалла (рис.1.8) или многоуровневую структуру (рис.1.9). В ПЛИС семейств Stratix используется трехсторонняя трассировочная структура, а в ПЛИС Virtex-5 - двухсторонняя (рис.1.10). Уровень 1 использует прямые соединения, а уровень 2 и 3 программируемые соединения, которые отмечены кружками в пересечениях соединений. С повышением уровня соединений, при удалении от логических блоков, возрастает ширина трассировочных каналов.

Большинство фирм выпускает сложные одноуровневые ПЛИС, располагая LUT-таблицы входящие в состав ЛБ в виде квадратной матрицы на площади кристалла, при этом связи между LUT-таблицами выполняются в виде соединений (треков или дорожек), разделенных на отдельные участки (сегменты) электронными ключами (рис.1.8).

Иерархическая (многоуровневая) организация ПЛИС позволяет улучшить их технические характеристики (рис.1.9). Функциональные преобразователи в этом случае группируются в блоки (например, в логический блок ПЛИС семейства FLEX фирмы Altera входит 8 логических элементов), имеющие свою собственную локальную шину межсоединений. Логические блоки обмениваются сигналами через шины межсоединений верхнего уровня. Межсоединения (проводники) в каналах непрерывны (т.е. не разделены на сегменты электронными ключами), что обеспечивает малые задержки распространения сигналов между логическими

блоками и позволяет существенно сократить количество электронных ключей.

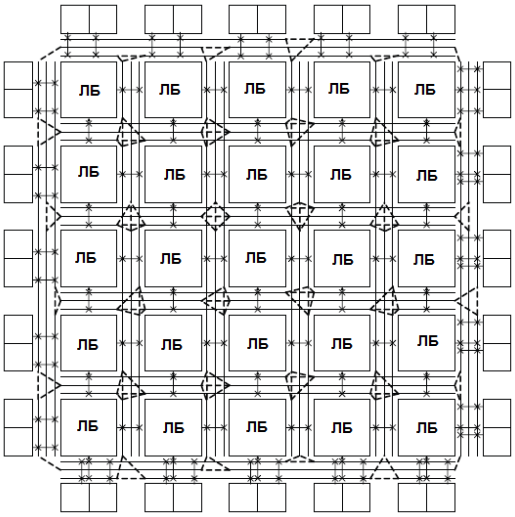


Рис.1.8. Одноуровневая структура ПЛИС типа ППВМ

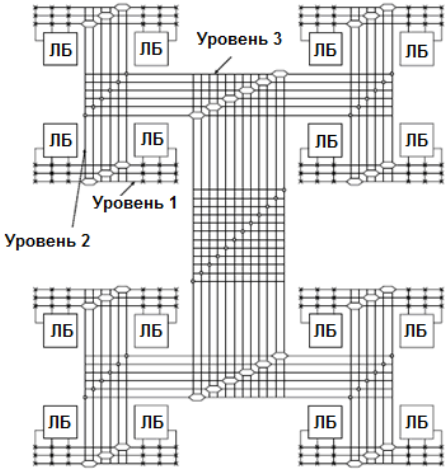


Рис.1.9. Многоуровневая структура ПЛИС типа ППВМ (FLEX10K, APEX, APEX II фирмы Altera)

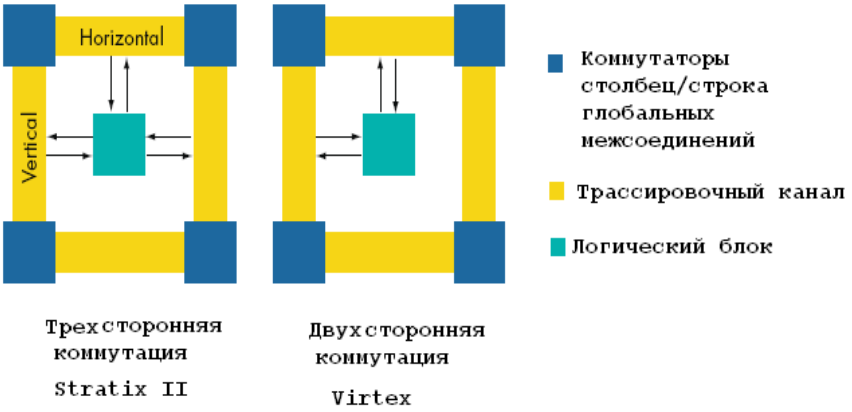
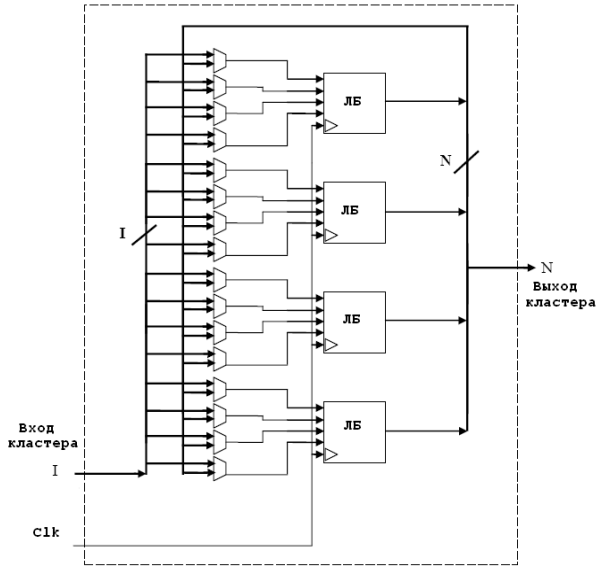


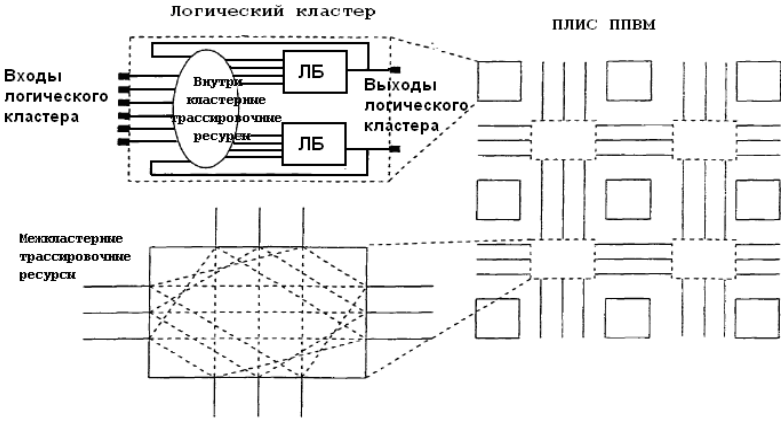
Рис.1.10. Трассировочная структура межсоединений в ПЛИС типа ППВМ Stratix фирмы Altera и Virtex фирмы Xilinx

Для одноуровневой и многоуровневой структуры ПЛИС логические блоки зачастую выгодно объединять в кластеры (рис.1.11). Так, под терминологией конфигурируемый логический блок фирмы Altera подразумевается кластер из 8 логических блоков (для ПЛИС серии FLEX10K).

Для ранних архитектур ПЛИС Xilinx серий XC3000 и XC4000 (рис.1.12) характерно: наличие канальных межсоединений разделенных электронными ключами в коммутационном блоке; прямые межсоединения, соединяющие выходы логического блока (ЛБ) со входами/выходами четырех соседних ЛБ; длинные горизонтальные и вертикальные линии проходящие вдоль всего кристалла; сеть тактовых синхросигналов, охватывающая весь кристалл, подключаемая к синхровходам триггеров ЛБ.



а)



б)

Рис.1.11. Объединение логических блоков в кластеры (а) и использование кластеров в одноуровневых ПЛИС типа ППВМ (б)

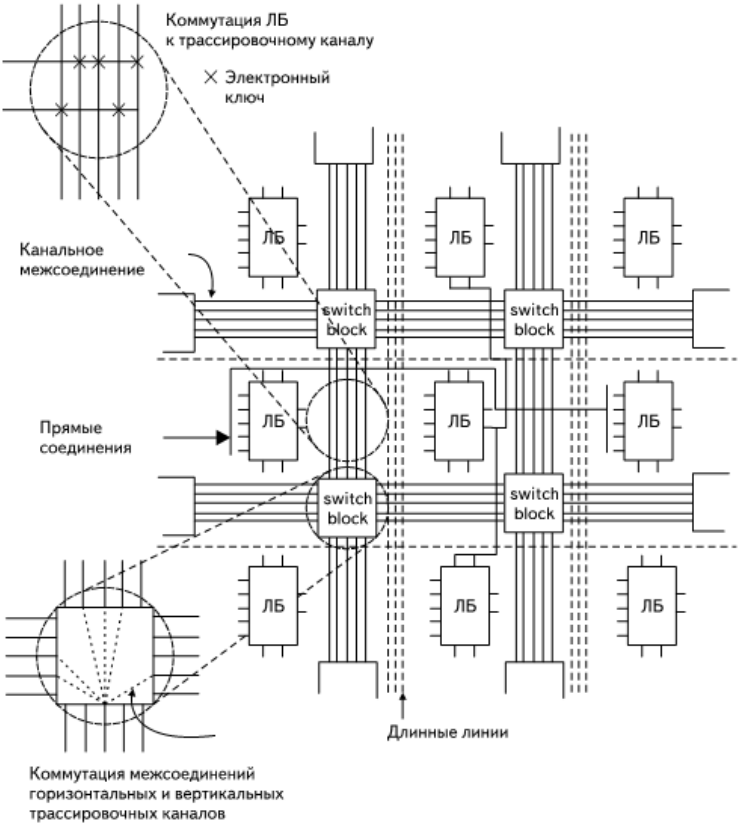


Рис.1.12. Архитектура ПЛИС типа ППВМ Xilinx 3000

ПЛИС с одноуровневой структурой межсоединений представляют массив логических блоков (логических элементов) подключаемых с помощью соединительных блоков С1 и С2 к вертикальным и горизонтальным трассировочным каналам межсоединений (рис.1.13). Блок С1 подключает один из входов (второй) LUT-таблицы и сигнал set/reset к вертикальному каналу, а блок С2 подключает один из входов (третий) и выход ЛЭ к горизонтальному каналу. Соединительные блоки С1 и С2 представляют собой программируемые коммутаторы на мультиплексорных

структурах, позволяющие подключать любое межсоединение из горизонтального или вертикального трассировочных каналов на один из входов логического блока. Блок С2 подключает любое межсоединение из канала к входу in3, а блок С1 подключает любое межсоединение из канала к входу in2. Для коммутации выходов ЛБ в блоке С2 используются коммутаторы (демультиплексоры) типа track-to-pin (один n-МОП ключ на каждое соединение) на проходных n-МОП ключах. В соединительных блоках С1 и С2 в качестве буферных элементов используются два последовательно соединенных инвертора.

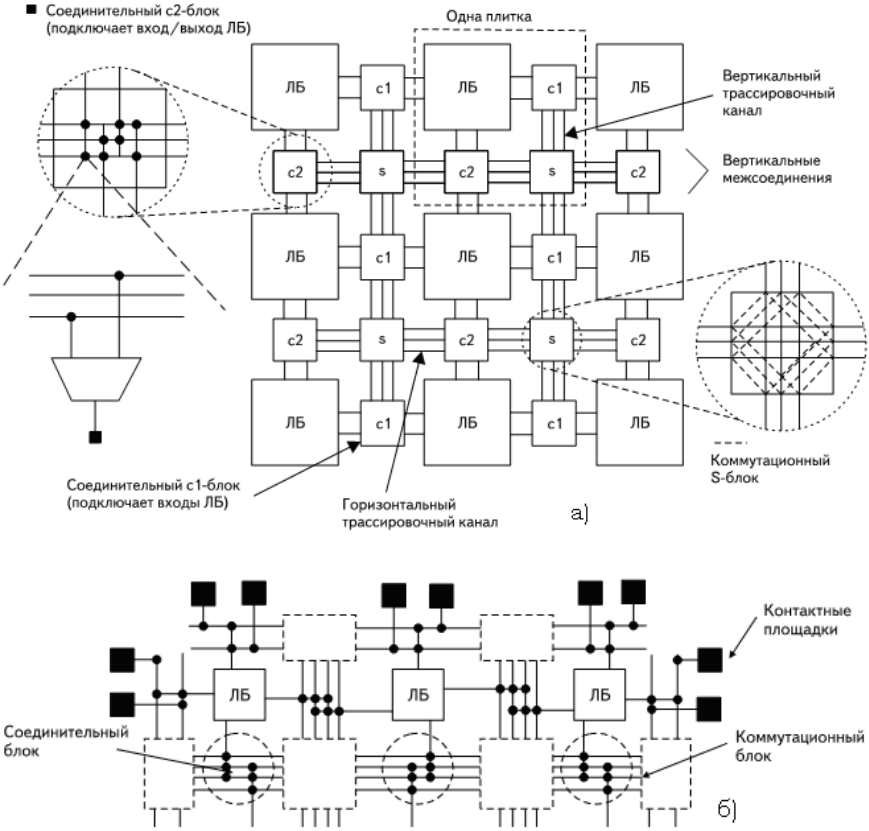


Рис.1.13. Архитектура академических ПЛИС

Коммутация межсоединений в каналалах осуществляется с помощью программируемого коммутатора-маршрутизатора (S-блок) представляющего из себя два шеститранзисторных n-МОП ключа, два непрерывных прямых горизонтальных каналов и два вертикальных канала. Совместно с n-МОП ключем применяется буфер восстановления уровня сигнала (рис.1.14), т.к. при использовании n-МОП ключей высокий уровень в цепочке снижается после каждого элемента на величину порогового напряжения. Буфер с восстановлением уровня с p-МОП транзистором в обратной связи широко используется в коммутаторах межсоединений коммерческих ПЛИС, например, серии Flex, Stratix фирмы Altera.

Замена программируемого межсоединения на жесткое, показано на рис.1.15. Пунктиром показаны направления коммутации межсоединений определяемые конфигурационными ключами. Что позволяет уменьшить число конфигурационных ячеек памяти, буферов восстановления уровня сигнала и существенно увеличить быстродействие ПЛИС.

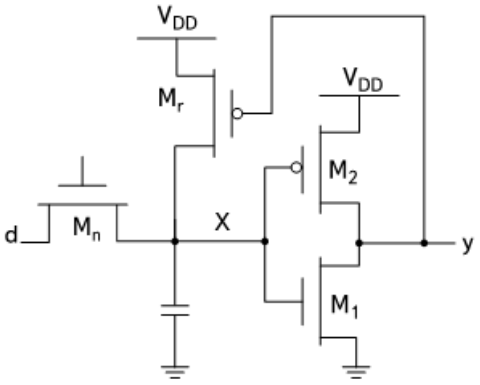


Рис.1.14. Схема буфера с восстановлением уровня с p-МОП транзистором (фиксатором) в положительной обратной связи

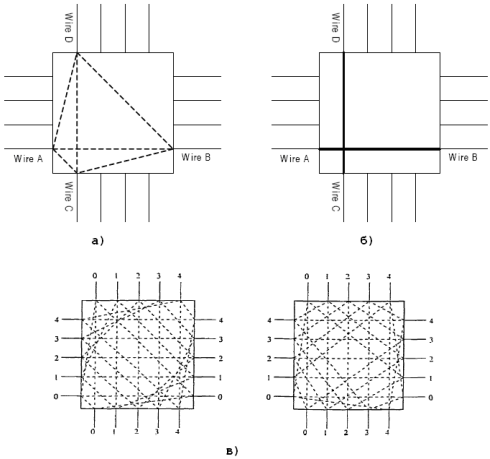


Рис.1.15. Замена программируемого соединения (а) на жесткое (б) высокоскоростное соединение и различные варианты реализации коммутационного блока (в)

1.4. Индустриальные ПЛИС фирмы Altera

ПЛИС можно разделить на два больших подмножества. Это ПЛИС с энергонезависимой прошивкой (т.е. независимо от наличия или отсутствия питания внутренняя конфигурация ПЛИС - “зашивка” - сохраняется) и ПЛИС с загрузкой конфигурации при включении питания (при выключении питания конфигурация “сбрасывается”).

Первые выполнены по технологии EEPROM (семейства MAX 7000/E/S/A, MAX 9000A) и допускают не менее 100 циклов стирания-записи, вторые - по технологии SRAM (семейства FLEX 6000/A, FLEX 8000A, FLEX 10K/KA/KB/KE) с неограниченным количеством циклов перезаписи и позволяющих перезагружать конфигурацию непосредственно в процессе работы. К первому подмножеству относятся также микросхемы семейства Classic с однократным программированием или с ультрафиолетовым стиранием.

1.4.1. Семейство ПЛИС MAX3000 и MAX7000

ПЛИС семейства MAX3000 выполнены по КМОП ЭСПЗУ технологии (ПЗУ с электрическим стиранием) при соблюдении технологических норм 0.35 мкм, что позволило существенно удешевить их по сравнению с семейством MAX7000S. Все ПЛИС MAX3000 поддерживают технологию программирования в системе (ISP, In-system programmability) и периферийного сканирования (boundary scan) в соответствии со стандартом IEEE Std. 1149.1 JTAG. Элементы ввода-вывода (ЭВВ) позволяют работать в системах с уровнями сигналов 5 В, 3.3 В, 2.5 В. Матрица соединений имеет непрерывную структуру, что позволяет реализовать время задержки распространения сигнала до 4.5 нс. ПЛИС MAX3000 имеют возможность аппаратной эмуляции выходов с открытым коллектором (open - drains pin) и удовлетворяют требованиям стандарта PCI. Имеется возможность индивидуального программирования цепей сброса, установки и тактирования триггеров, входящих в макроячейку. Предусмотрен режим пониженного энергопотребления. Программируемый логический расширитель позволяет реализовать на одной макроячейке функции до 32 переменных. Имеется возможность задания бита секретности (security bit) для защиты от несанкционированного тиражирования разработки. Реализация функции программирования в системе поддерживается с использованием стандартных средств загрузки, таких как ByteBlasterMV, BitBlaster, MasterBlaster, а также поддерживается формат JAM. ПЛИС MAX3000 выпускаются в корпусах от 44 до 208 выводов. На рис.1.16 представлена функциональная схема ПЛИС семейства MAX3000.

Основными элементами структуры ПЛИС семейства MAX3000 являются: логические блоки (ЛБ) (LAB, Logic array

blocks); макроячейки (МЯ) (macrocells); логические расширители (expanders) (параллельный (parallel) и разделяемый (shareble)); программируемая матрица соединений (ПМС) (Programmable interconnect array, PIA); элементы ввода-вывода (ЭВВ) (I/O control block).

ПЛИС семейства MAX3000 имеют четыре вывода, закрепленных за глобальными цепями (dedicated inputs). Это глобальные цепи синхронизации сброса и установки в третье состояние каждой макроячейки. Кроме того, эти выводы можно использовать как входы или выходы пользователя для “быстрых” сигналов, обрабатываемых в ПЛИС.

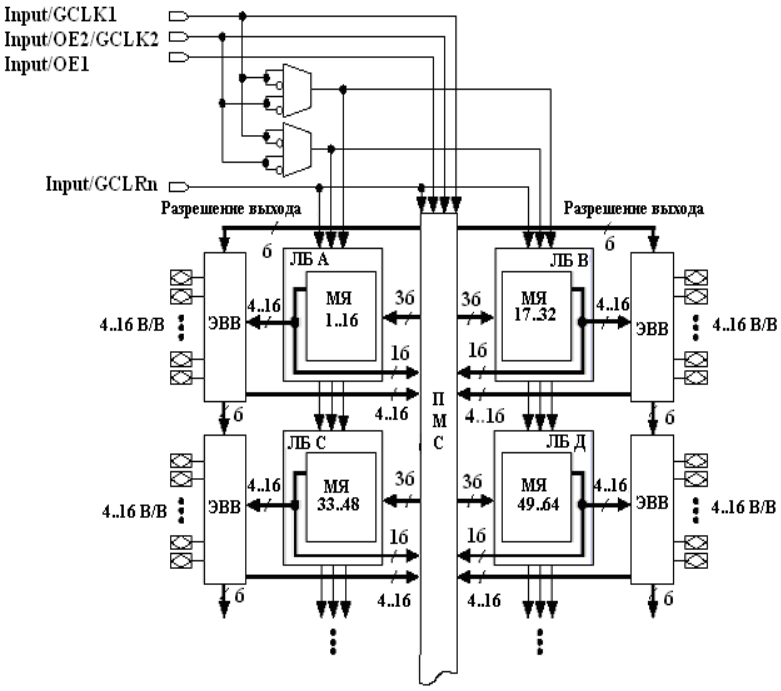


Рис.1.16. Функциональная схема ПЛИС семейства MAX3000

Как видно из рис.1.16, в основе архитектуры ПЛИС семейства MAX3000 лежат логические блоки, состоящие из 16 макроячеек каждый. Логические блоки соединяются с помощью программируемой матрицы соединений. Каждый логический блок имеет 36 входов с ПМС. На рис.1.17 приведена структурная схема макроячейки ПЛИС семейства MAX3000.

МЯ ПЛИС семейства MAX3000 состоит из трех основных узлов: локальной программируемой матрицы (LAB local array); матрицы распределения термов (product-term select matrix); программируемого регистра (Programmable register).

Комбинационные функции реализуются на локальной программируемой матрице и матрице распределения термов, позволяющей объединять логические произведения либо по ИЛИ (OR), либо по исключающему ИЛИ (XOR). Кроме того, матрица распределения термов позволяет скоммутировать цепи управления триггером МЯ.

Режим тактирования и конфигурация триггера выбираются автоматически во время синтеза проекта в САПР MAX+PLUS II в зависимости от выбранного разработчиком типа триггера при описании проекта.

В ПЛИС семейства MAX3000 доступно 2 глобальных тактовых сигнала, что позволяет проектировать схемы с двухфазной синхронизацией.

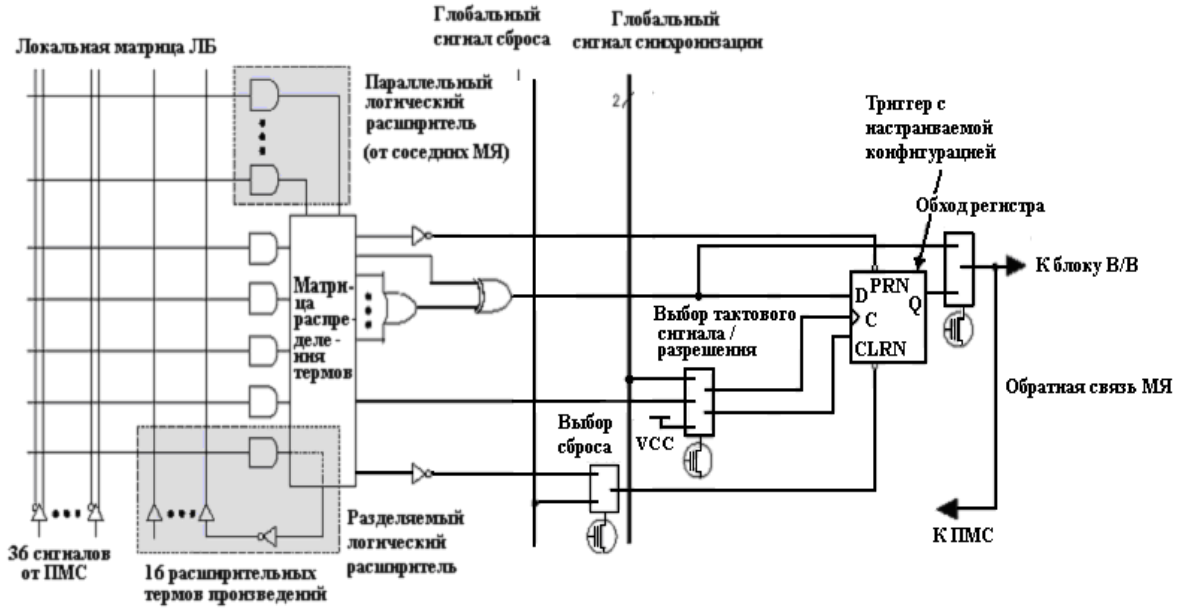


Рис.1.17. Структурная схема макроячейки ПЛИС семейства MAX3000

Для реализации логических функций большого числа переменных используются логические расширители. Разделяемый логический расширитель позволяет реализовать логическую функцию с большим числом входов, позволяя объединить МЯ, входящие в состав одного ЛБ. Таким образом, разделяемый расширитель формирует терм, инверсное значение которого передается матрицей распределения термов в локальную программируемую матрицу и может быть использовано в любой МЯ данного ЛБ. Как видно из рис.1.8, имеются 36 сигналов локальной ПМС, а также 16 инверсных сигналов с разделяемых логических расширителей, что позволяет в пределах одного ЛБ реализовать функцию до 52 термов ранга 1.

Параллельный логический расширитель позволяет использовать локальные матрицы смежных МЯ для реализации функций, в которые входят более 5 термов. Одна цепочка параллельных расширителей может включать до 4 МЯ, реализуя функцию 20 термов. На ПМС выводятся сигналы от всех возможных источников: ЭВВ, сигналов обратной связи ЛБ, специализированных выделенных выводов. В процессе программирования только необходимые сигналы “заводятся” на каждый ЛБ.

ПЛИС семейства MAX3000 полностью поддерживают возможность программирования в системе в соответствии со стандартом IEEE Std. 1149.1 –1990 (JTAG) с использованием соответствующих инструментальных средств.

Повышенное напряжение программирования формируется специализированными схемами, входящими в состав ПЛИС семейства MAX3000, из напряжения питания 3.3 В. Во время программирования в системе входы и выходы ПЛИС находяться в третьем состоянии и “слегка” подтянуты к напряжению питания. Сопротивления внутренних подтягивающих резисторов порядка 50 кОм.

Семейство MAX (Multiple Array matriX) 7000 объединяет семь серий ПЛИС. ПЛИС этого семейства позволяют заменить устройство, содержащее до сотни корпусов микросхем средней степени интеграции, и обеспечивают: задержку распространения сигнала от любого входа до выхода ПЛИС не более 5 нс; устойчивую работу на частотах до 151 МГц; возможность регулирования скорости переключения выходных буферов; возможность использования четырех режимов работы выходных буферов: вход, выход, двунаправленный, открытый коллектор; возможность задания режима пониженного энергопотребления (Turbo-off) как для всей СБИС в целом, так и для цепей распространения отдельных сигналов; возможность программирования и репрограммирования после распайки на плате; возможность задания режима секретности разработки (Design Security); работу с пониженным (3.3 В) напряжением питания.

ПЛИС семейства MAX7000 являются первыми CPLD фирмы ALTERA, выполненными по КМОП ЭСПЗУ технологии. В настоящее время выпускаются ПЛИС MAX7000, MAX7000A, MAX7000B, MAX7000E, MAX7000S. Семейства MAX7000A и MAX7000B расчитаны на работу в системах с напряжением питания 3.3 и 2.5 В соответственно, ПЛИС MAX7000S является дальнейшим развитием 5 вольтового MAX7000, допуская возможность программирования в системе. ПЛИС допускают не менее 100 циклов стирания-записи, чего для отладки изделия более чем достаточно. Это гарантированная фирмой цифра, хотя реально количество циклов может доходить до 250.

ПЛИС семейства MAX 7000 содержат от 32 до 256 макроячеек, которые объединены в группы по 16 макроячеек, называемые LAB (Logic Array Block). Каждая макроячейка содержит программируемую матрицу “И” и фиксированную матрицу “ИЛИ”, а также конфигурационный триггер с

независимыми сигналами тактового импульса, разрешения тактового импульса, сброса и предустановки. Для реализации сложных логических функций к каждой макроячейке могут быть подключены дополнительные термы произведений (за счет специальных ресурсов - так называемых “expander terms”), что увеличивает количество термов в макроячейке до 32.

ПЛИС семейства MAX7000S отличаются от MAX7000 возможностью программирования в системе (ISP) с использованием JTAG-интерфейса через специальный кабель Byteblaster (Bitblaster) непосредственно из САПР, а также пониженным по сравнению с MAX7000 потреблением.

Для оптимизации соотношения “быстродействие/потребление” используется так называемый программируемый “турбо-бит” который, будучи запрограммированным в состояние “ON” позволяет цепи достичь максимального быстродействия. В случае программирования его в состояние “OFF” - ограничивается максимальное быстродействие цепи (приблизительно в 2-2,5 раза), при этом сокращается потребление примерно в том же соотношении. “Турбо-бит” может быть запрограммирован для каждого конкретного сигнала независимо от других.

ПЛИС MAX 7000 выпускаются в различных типах корпусов (PLCC, QFP, PGA) с количеством выводов от 44 до 208.

Семейство MAX 7000A. Технология EEPROM с возможностью программирования в системе (ISP). Напряжение питания 3,3 В, и, следовательно, пониженное на 40 % потребление. Возможность установки режима пониженного потребления (около 50 %) для каждой макроячейки. Программируемый бит секретности, позволяющий защитить ПЛИС от несанкционированного считывания. Совместимость по выходам с уровнями 5,0/3,3/2,5 В.

ПЛИС MAX7000AE способны корректно работать в режиме "горячего включения" (Hot Socketing), т.е. допускают подачу входных сигналов на контакты и отсутствие сигналов на выходе до или во время включения питания.

ПЛИС EPM7032AE, EPM7064AE, EPM7128A и EPM7256AE по выводам совместимы с соответствующими микросхемами семейства MAX 7000S. Схожесть внутренней архитектуры позволит разработчикам, применяющим сегодня микросхемы MAX 7000S и MAX 9000A, легко перевести свои проекты на MAX 7000A.

Семейство MAX 7000S. Все ПЛИС MAX7000s поддерживают технологию программирования в системе (ISP, In-system programmability) и периферийного сканирования (boundary scan) в соответствии со стандартом IEEE Std. 1149.1 JTAG. Элементы ввода-вывода (ЭВВ) позволяют работать в системах с уровнями сигналов 5 В или 3.3 В. Матрица соединений имеет непрерывную структуру, что позволяет реализовать время задержки распространения сигнала до 5 нс. ПЛИС MAX7000s имеют возможность аппаратной эмуляции выходов с открытым коллектором (open - drains pin) и удовлетворяют требованиям стандарта PCI. Имеется возможность индивидуального программирования цепей сброса, установки и тактирования триггеров, входящих в макроячейку. Предусмотрен режим пониженного энергопотребления. Программируемый логический расширитель позволяет реализовать на одной макроячейке функции до 32 переменных. Имеется возможность задания бита секретности (security bit) для защиты от несанкционированного тиражирования разработки.

Семейство MAX (Multiple Array matriX) 9000 объединяет четыре серии ПЛИС. ПЛИС этого семейства позволяют заменить устройство, занимающее десятки плат, выполненных на микросхемах средней степени интеграции, и обеспечивают возможность: устойчивой работы на частотах до

117 МГц; независимого использования логической части и триггера макроячейки; задания режима пониженного энергопотребления (power-saving mode) как для всей ПЛИС в целом, так и для цепей распространения отдельных сигналов; программирования и репрограммирования после распайки на плате; работы в системах со смешанным напряжением питания (3.3 В, 5.0 В); регулирования скорости переключения выходных буферов; использования трех режимов работы выходных буферов: вход, выход, двунаправленный.

ПЛИС как семейства MAX9000, так и MAX9000A обладают возможностью программирования в системе (ISP) и полностью совместимы между собой по корпусам функциональным возможностям и по файлу прошивки. ПЛИС MAX9000A изготовлены по более совершенной технологии, что позволило повысить быстродействие микросхем до 7.5 нс (178 МГц) от входа до выхода.

1.4.2. Семейство ПЛИС FLEX6000 и FLEX10K

По своим характеристикам ПЛИС FLEX6000 является промежуточным между семействами FLEX8000 и FLEX10K. ПЛИС FLEX6000 выпускаются по технологии 0.5 мкм СОЗУ (SRAM), FLEX6000A по 0.35 мкм с тремя слоями металлизации и обладают удачными характеристиками цена-производительность для реализации не очень сложных алгоритмов ЦОС. Отличительной особенностью архитектуры ПЛИС FLEX6000 является технология OptiFLEX, представленная на рис.1.18.

В основе архитектуры OptiFLEX лежат логические блоки (ЛБ) (LABs, Logic array blocks), каждый из которых объединяет по 10 логических элементов (ЛЭ) (Logic elements) с помощью локальной матрицы соединений. Особенностью архитектуры OptiFLEX является то, что каждый логический

элемент может коммутироваться как на локальную матрицу соединений собственного логического блока, так и смежных (рис.1.18), тем самым расширяются возможности для трассировки.

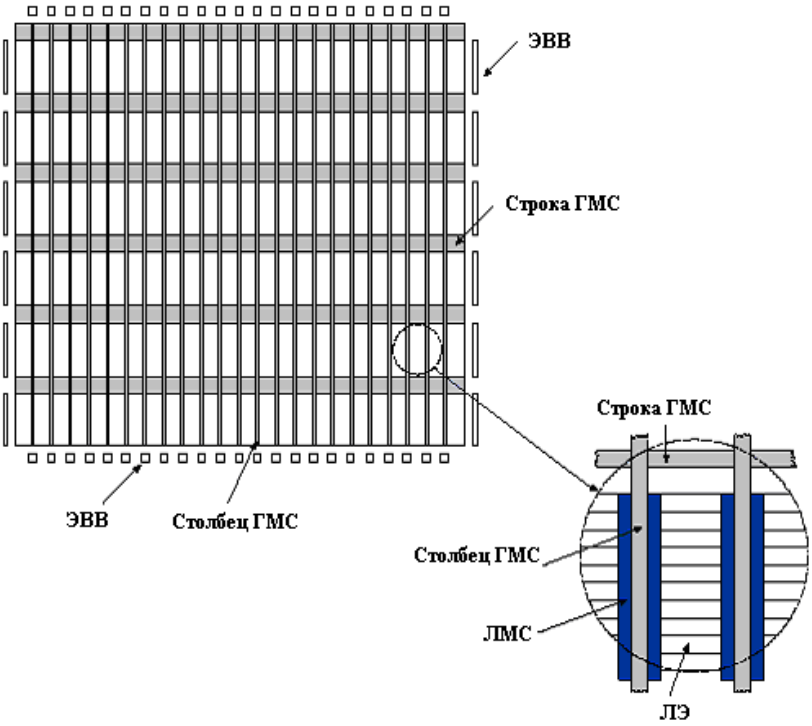


Рис.1.18. Технология OptiFLEX

На рис.1.19 приведена структура ЛБ ПЛИС семейства FLEX6000. Как видно из рис.1.19, ЛБ имеет чередующуюся структуру (interleaved structure), объединяя на локальной матрице соединений (ЛМС) (lockal interconnect) сигналы с двух смежных ЛБ. Кроме того, сигналы с ЛЭ и ЛМС могут коммутироваться на строки и столбцы глобальной матрицы соединений, которые имеют непрерывную структуру, обеспечивающую минимальные задержки. Каждый ЛБ и ЛЭ

управляется выделенными глобальными сигналами (Dedicated inputs), являющимися сигналами сброса, установки и синхронизации триггеров ЛЭ (рис.1.12).

На рис.1.20 приведена структура ЛЭ ПЛИС семейства FLEX6000. В основе ЛЭ лежит четырехвходовая таблица перекодировок (ТП, LUT, Look-up Table). Кроме того, в состав ЛЭ входят цепи ускоренного цепочечного переноса (Carry-in, carry-out) и каскадирования (cascade-in, cascade-out). Триггер ЛЭ может быть сконфигурирован с помощью логики сброса-установки (clear/preset logic), тактируется одним из сигналов, выбираемых логикой тактирования (clock select). При необходимости сигнал с выхода ТП может быть подан на выход ЛЭ в обход триггера (register bypass).

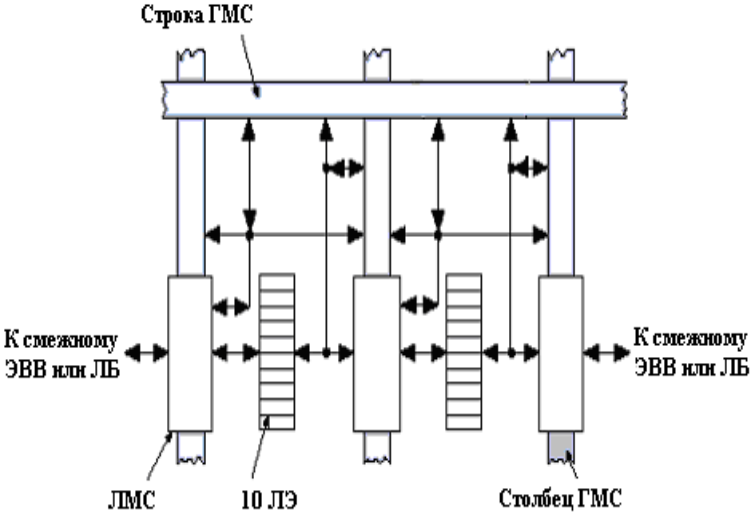


Рис.1.19. Структура ЛБ FLEX6000

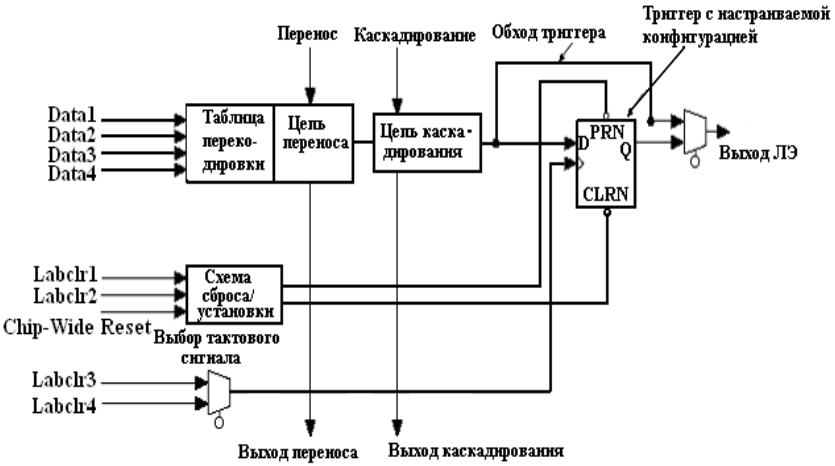


Рис.1.20. Структура ЛЭ ПЛИС семейства FLEX6000

Для обеспечения минимальной задержки при реализации сложных арифметических функций, таких как счетчики, сумматоры, вычитатели и т.п., используется организация ускоренных цепочечных переносов (carry chain) между ЛЭ. При организации цепочечных переносов первый ЛЭ каждого ЛБ не включается в цепочку цепочечных переносов, поскольку он формирует управляющие сигналы ЛБ. Вход первого ЛЭ в каждом ЛБ может быть использован для формирования сигналов синхронной загрузки или сброса счетчиков, использующих цепочечный перенос. Цепочка переносов длиннее чем 9 ЛЭ автоматически формируется путем объединения нескольких ЛБ вместе, причем перенос формируется не в соседний ЛБ, а через один, то есть из четного в четный, из нечетного ЛБ – в нечетный. Например, последний ЛЭ в первом ЛБ в ряду формирует перенос во второй ЛЭ в третьем ЛБ в том же ряду.

Семейство FLEX (Flexible Logic Element matriX) 10K объединяет семь серий ПЛИС. ПЛИС этого семейства позволяют заменить устройство, занимающее сотни плат, выполненных на микросхемах средней степени интеграции, и

обеспечивают возможность: устойчивой работы на частотах до 450 Мгц; реализации на кристалле статической памяти и ПЗУ объемом до 24 Кбит; независимого использования логической части и триггера каждого логического элемента; эмуляции внутренней шины с тремя состояниями; умножения внутренней тактовой частоты; работы в системах со смешанным напряжением питания (3.3 В, 5.0 В); реализации неограниченного числа циклов репрограммирования, в том числе "на лету", т.е. без выключения питания ПЛИС; регулирования скорости переключения выходных буферов; использования четырех режимов работы выходных буферов: вход, выход, двунаправленный, открытый коллектор.

ПЛИС семейства FLEX10K изготовлены по технологии SRAM с загрузкой при включении питания. Интеграция до 250000 вентилей. Особенностью этого семейства является возможность реализации на одном кристалле логических функций и памяти (RAM, ROM, FIFO), при этом память реализуется без затрат основной логики. Время доступа к памяти до 6 нс.

Память реализуется на специализированных ресурсах микросхемы, которые также могут быть использованы для эффективной реализации различных алгоритмов, в т.ч. цифровой обработки сигналов. Эти ПЛИС позволяют достичь высоких характеристик при разработке устройств, совместимых со стандартом PCI.

ПЛИС семейства FLEX10K имеют двухуровневую архитектуру матрицы соединений (рис.1.21). 8 ЛЭ объединяются в группы - логические блоки (ЛБ). Внутри логических блоков ЛЭ соединяются посредством локальной программируемой матрицы соединений (ЛМС), позволяющей соединять любой ЛЭ с любым. Логические блоки связаны между собой и с элементами ввода/вывода посредством глобальной программируемой матрицы соединений (ГПМС). Локальная и глобальная матрицы соединений имеют

непрерывную структуру - для каждого соединения выделяется непрерывный канал.

Встроенный блок памяти (ВБП) представляет собой ОЗУ емкостью 2048 (4096) бит и состоит из ЛМС, модуля памяти, синхронных буферных регистров и программируемых мультиплексоров. Наличие синхронных буферных регистров и программируемых мультиплексоров позволяет конфигурировать ВБП как ЗУ с организацией 256x8, 512x4, 1024x2, 2048x1.

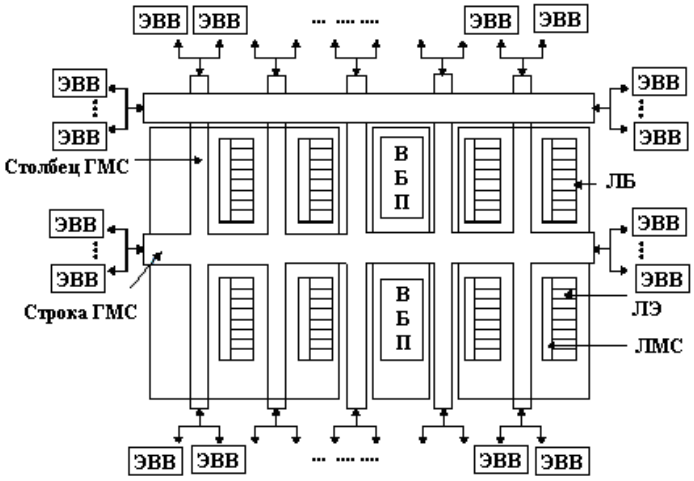


Рис.1.21. Архитектура ПЛИС семейства FLEX10K

На рис.1.22, укрупненным планом показана коммутация сигналов в ПЛИС FLEX10KE фирмы Altera. Осуществляется коммутация с ГМС строки, выделенных информационных входов (не показаны), выделенных тактовых входов (не показаны) на ЛМС; коммутацию столбца на строку ГМС; коммутацию строки на столбец ГМС; коммутацию выходов ЛЭ на строку и столбец ГМС; коммутацию КЛБ с КЛБ через соседний КЛБ.

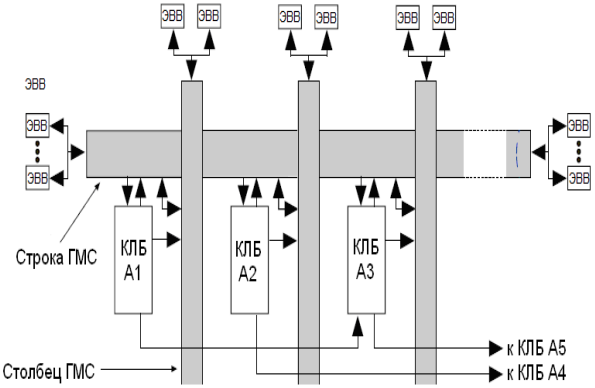


Рис.1.22. Программируемая коммутация сигналов в ПЛИС FLEX10KE фирмы Altera

На рис.1.22, б) показано, как осуществляется коммутация сигналов с дорожек ГМС на ЛМС КЛБ и с ЛМС на входы ЛЭ.

Коммутаторы в ПЛИС могут быть реализованы на мультиплексорах (рис.1.23) или с использованием одного n-МОПТ ключа (рис.1.24), на каждое track-to-pin соединение. В первом случае требуется 4 ячейки конфигурационной памяти (24 транзистора), во втором случае 16 ячеек памяти (96 транзисторов), однако, появляется критический путь из четырех последовательно соединенных n-МОПТ-ключей. В мультиплексорах могут быть использованы как n-МОПТ ключи (рис.1.23, *б*), так и КМОП-ключи (рис.1.23, *в*).

Использование мультиплексоров дает меньшую площадь, но наносит существенный удар по быстродействию, так как сигнал должен проходить через серии ключей. При использовании n-МОПТ-ключей высокий уровень в цепочке снижается после каждого элемента на величину порогового напряжения.

На рис.1.25 показана детализированная схема программируемой коммутации выходных сигналов ЛЭ КЛБ на строки и столбцы ГМС и обратно на ЛМС, а также

коммутация столбца на строку ГМС в ПЛИС FLEX10KE фирмы Altera. Коммутация выхода ЛЭ 1 показана на рис.1.26.

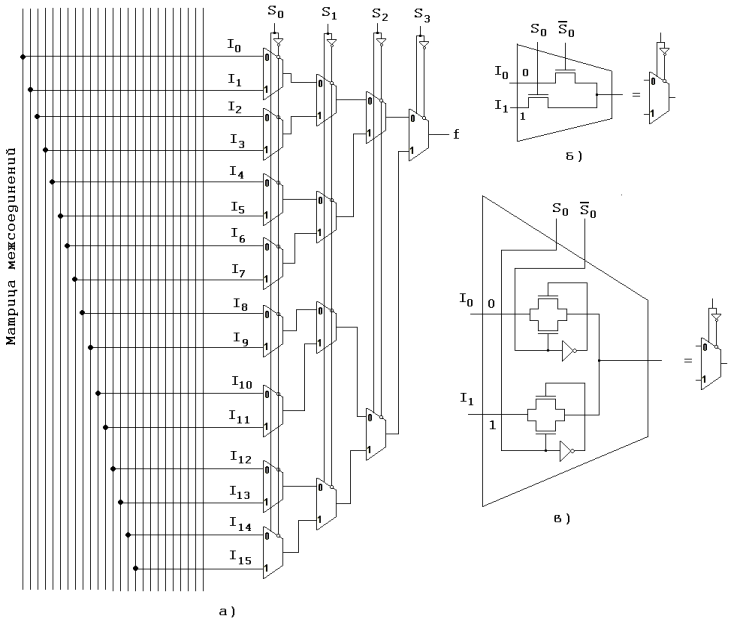


Рис.1.23. Коммутатор сигналов с ЛМС на вход КЛБ с использованием мультиплексоров: S0-S3 – конфигурационные биты памяти (а); б – мультиплексор на n-МОПТ ключах; в – мультиплексор на КМОП-ключах

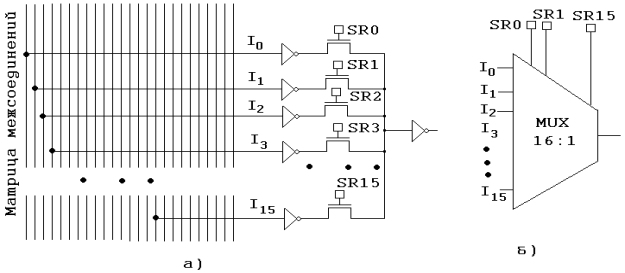


Рис.1.24. Коммутатор сигналов 16 в 1 с ЛМС на вход КЛБ с использованием n-МОПТ ключей на каждое соединение (а); б) условное обозначение

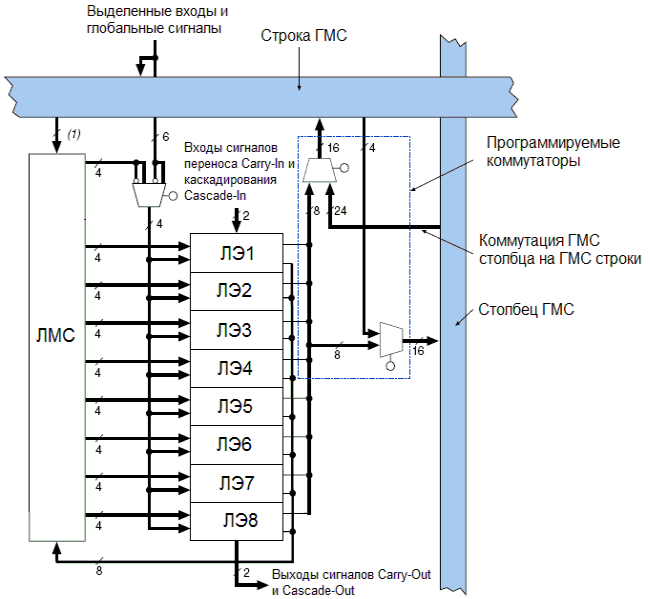


Рис.1.25. Программируемая коммутация сигналов в ПЛИС FLEX10KE фирмы Altera

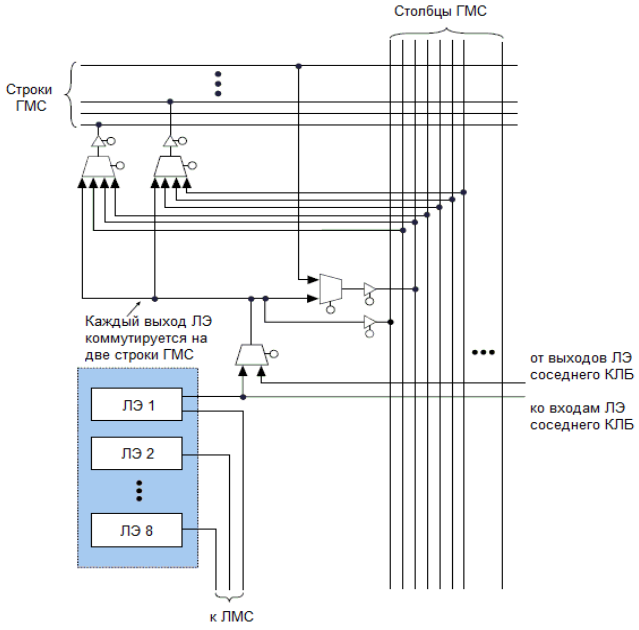


Рис.1.26. Коммутация выхода ЛЭ 1 на строки, столбцы ГМС и на ЛМС в ПЛИС FLEX10KE фирмы Altera

1.4.3. Семейство ПЛИС APEX20K

Развитие и разнообразие архитектур функциональных преобразователей, лежащих в основе базовых узлов ПЛИС, привели к тому, что в последние годы ПЛИС становятся основой для “систем на кристалле” (system-on-chip, SOC). В основе идеи SOC лежит интеграция всей электронной системы в одном кристалле (например, объединение на одном кристалле процессора, памяти и др. цифровых устройств). Компоненты этих систем разрабатываются отдельно и хранятся в виде файлов параметризируемых модулей. Окончательная структура SOC-ИС выполняется на базе этих “виртуальных компонентов”, называемых также “блоками интеллектуальной собственности”, с помощью САПР БИС. Благодаря стандартизации в одно целое можно объединять “виртуальные компоненты” от разных разработчиков.

Примером новых семейств ПЛИС, пригодных для реализации “систем - на - кристалле”, является семейство APEX20K фирмы Altera (рис.1.27).

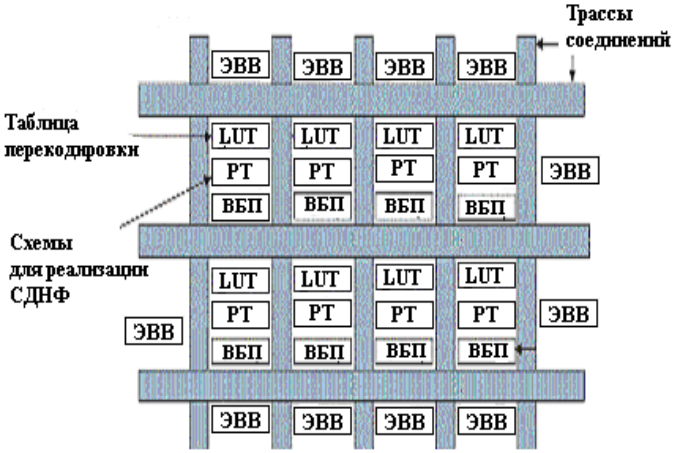


Рис.1.27. Архитектура APEX20K ПЛИС фирмы Altera

Архитектура APEX20K сочетает в себе как достоинства FPGA ПЛИС с их таблицами перекодировок (ТП) или LUT (Look-up table), входящими в состав логического элемента ПЛИС, так и логику вычисления СДНФ, характерную для ПЛИС CPLD - ПЛИС с высокой степенью интеграции элементов на кристалле (рис.1.27).

1.4.4. Семейство ПЛИС Stratix III

Stratix III, новое поколение ППВМ (FPGA) компании Altera, выполнено по технологии 65-нм. Главный упор сделан на снижение потребляемой мощности. ПЛИС Stratix III потребляют вдвое меньше энергии, работают на 25 % быстрее и обладают вдвое большим количеством логических элементов по сравнению с предыдущим поколением - Stratix II.

Семейство Stratix III представлено двумя главными направлениями: **Stratix III L** (Logic) - сбалансированы по количеству логических узлов (вентилей), памяти и блоков цифровой обработки сигналов (ЦОС) для приложений общего назначения; **Stratix III E** (Enhanced или улучшенные) - количество интегрированной памяти и блоков ЦОС увеличено для работы с соответствующими приложениями.

Основой семейства ПЛИС Stratix III являются адаптивные логические блоки (ALM), которые объединяются в логические блоки (logic array block, LAB). Таблица перекодировки (LUT) адаптивного логического блока (Memory LAB) может быть сконфигурирована как 16 x 2 двухпортовое ОЗУ. Всего 10 таких ALM могут образовать блок двухпортового ОЗУ емкостью 16 x 20 бит (320 бит конфигурационной памяти). LUT логических блоков (LAB) такой возможностью не обладают. Блоки Memory LAB и LAB сосуществуют парами и подключены к локальным матрицам межсоединений и к строкам и столбцам C4, C12, R4, R20

обладающими различной скоростью распространения сигналов и длинной межсоединений (рис.1.28). Локальные матрицы позваляют напрямую подключать соседние блоки LAB, блоки памяти с архитектурой TriMatrix, встроенные блоки цифровой обработки сигналов (DSP-блоки) или блоки ввода/вывода. Архитектура межсоединений MultiTrack обеспечивает большую доступность ко всем окружающим LAB с помощью меньшего числа связей, что позволяет увеличить производительность, снизить энергопотребление и оптимизировать упаковку логики.

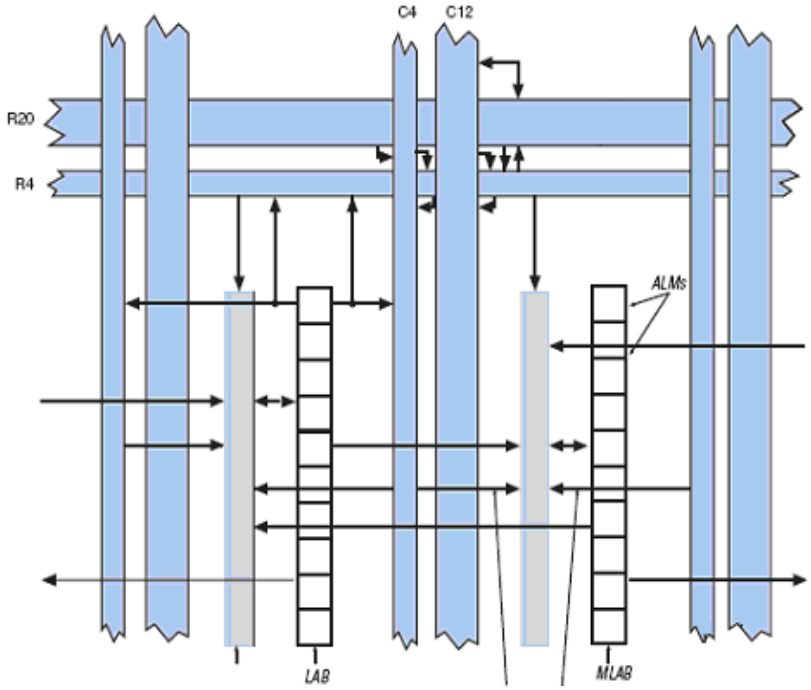


Рис.1.28. Фрагмент архитектуры ПЛИС семейства Stratix III

ALM обеспечивают эффективное управление логическими функциями, т.е. позволяют упаковывать большее число комбинационных и последовательностных логических

элементов. Каждый блок способен реализовывать различные комбинационные функции на 8-ми входовой таблице перекодировки, которая аппаратно реализуется на двух 4-х входовых адаптивных LUT-таблицах и последовательностную логику на двух программируемых D-триггерах, а также арифметические операции на двух специализируемых сумматорах (рис.1.29). С помошью такой LUT-таблицы можно реализовать булевы функции от 7, 6 и две независимые функции с меньшим числом переменных. Триггеры reg0 и reg1 могут быть использованы независимо от LUT-таблиц.

ALM способны работать в различных режимах: нормальный, расширенный LUT-режим, арифметический, общий арифметический и LUT-reg (в этом режиме используются три триггера). В каждом режиме ресурсы ALUT используются различно.

Например, ALM способен реализовать две независимые 4-входовые функции (без общих входов). На рис.1.30 показаны примеры реализации функций как без каких-либо общих входов, так и с ними. Нормальный режим работы позволяет обеспечить полную совместимость с 4-х входовыми LUT-таблицами предыдущих серий ПЛИС, такими как APEX II, FLEX.

Для реализации булевой функции 6-ти переменных могут быть использованы входы dataa, datab, datac, datad а также входы datae0, dataf0 или datae1, dataf1 (рис.1.31). Если используются входы datae0, dataf0 то выход функции регистерный или триггер reg0 может быть обойден. Если доступны входы datae1 и dataf0, то используется выход триггера reg1 или триггер reg1 может быть обойден.

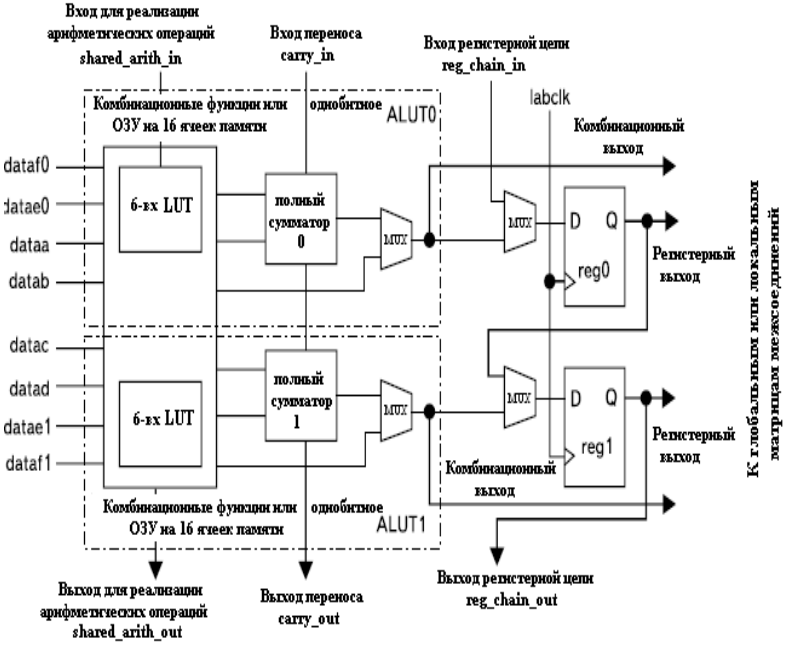


Рис.1.29. Адаптивный логический блок ПЛИС Stratix III

На рис.1.32 показан типовой шаблон реализации булевой функции 7-ми переменных для комбинационной схемы. Неиспользуемый вход dataf1 доступен для упаковки триггера. Данный шаблон используется для реализации конструкции if-else языка VHDL. Работа ALUT в арифметическом режиме показана на рис.1.33. Арифметический режим используется для реализации сумматоров, счетчиков и др. Встроенные сумматоры в ALM могут работать в двух режимах: как два независимые 2-х входовые сумматоры (арифметический режим) или один как 3-х входовой для реализации сложных арифметических операций (общий арифметический режим).

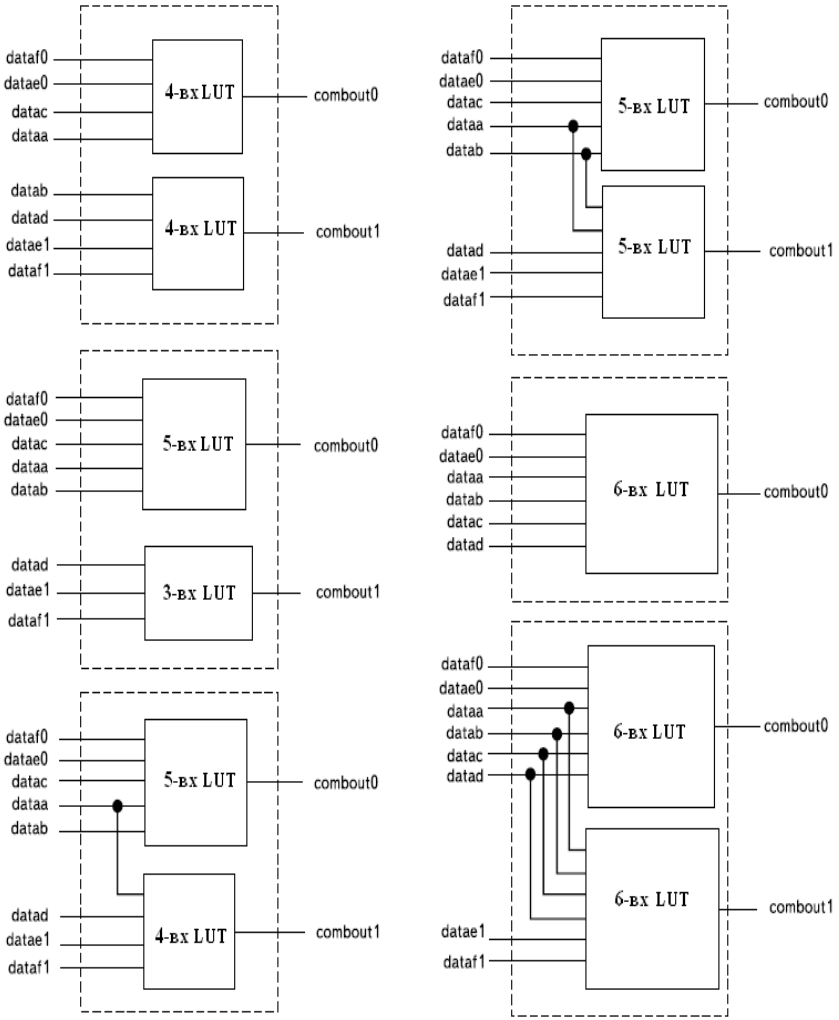


Рис.1.30. Реализация булевых функций в нормальном режиме работы ALM: а – двух независимых функций от 4-х переменных; б – независимой функции 5-ти и 3-х переменных; в – функции 5-ти и 4-х переменных с одним общим входом; г – функции 5-ти и функции 5-ти переменных с 2-мя общими входами; д –функции 6-ти переменных; е – функции 6-ти и функции 6-ти переменных с 4-мя общими входами

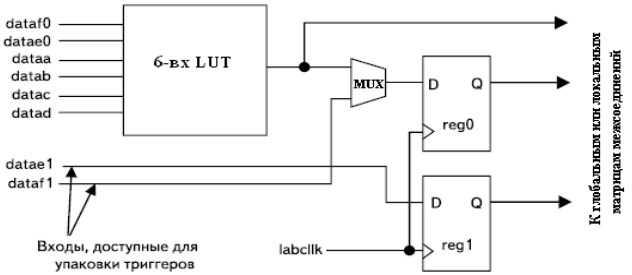


Рис.1.31. Реализация булевой функций 6-ти переменных в нормальном режиме работы ALM

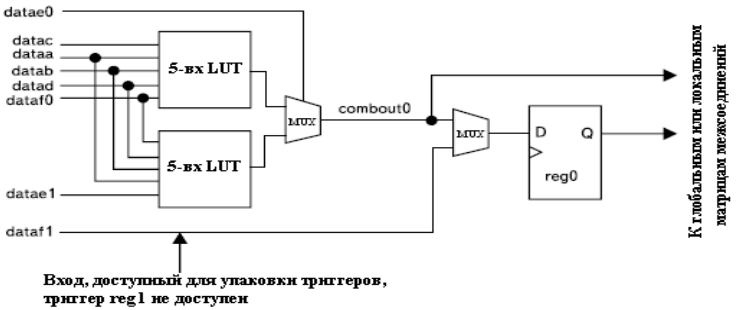


Рис.1.32. Работа ALUT в расширенном режиме

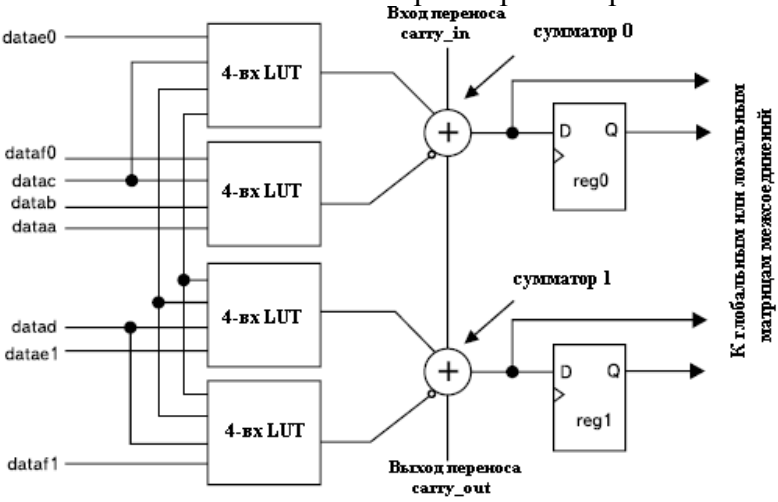


Рис.1.33. Работа ALUT в арифметическом режиме

ПЛИС Stratix включают в себя до 28 блоков ЦОС, которые повышают эффективность устройства при реализации приложений, требующих арифметических операций. Блоки ЦОС могут быть включены как умножители, или как умножители с накоплением, что обеспечивает увеличение эффективности при одновременном увеличении скорости обработки данных и существенно сберегает ресурс, занимаемый на кристалле проектом пользователя. На кристалле также может находиться до 12 схем автоподстройки частоты (PLLs). Цепи синхронизации могут иметь до 40 системных синхрочастот. ПЛИС Stratix поддерживают множество стандартов ввода – вывода, как для передачи сигналов по однопроводным соединительным линиям, так и по дифференциальным линии. Таким образом, данные устройства находятся на новом уровне системной интеграции для system-on-a-programmable-chip проектов (SOPC).

1.5. Программные средства проектирования ПЛИС

Рассмотрим программные инструменты T-Vpack и VPR, разработанные в университете Торонто (Канада, Торонто, http://www.eecg.utoronto.ca/vpr) для проектирования академических ПЛИС типа ППВМ с одноуровневой структурой межсоединений под технологические проектные нормы 22 – 180 нм КМОП технологии с минимальной площадью кристалла и шириной трассировочного канала, нахождением критического пути в трассировочных ресурсах ПЛИС. Успехи в области исследования и создания новых архитектур ПЛИС с использованием T-Vpack и VPR привели к созданию в Торонто технологического центра фирмы Altera (Altera Toronto Technology Centre).

Основные функциональные блоки ПЛИС (рис.1.34): логический блок (ЛБ), соединительные блоки (С-блоки),

коммутатор-маршрутизатор (S-блок). 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, т.к. позволяет получать существенный выигрыш по сравнению с технологией multi-driver по быстродействию (задержка распространения сигнала в трассировочных ресурсах ПЛИС уменьшается на 9 %) и по площади кристалла (экономия площади кристалла до 25 %). Технология соединений single-driver, известная как DirectDrive™ широко используется в современных сериях ПЛИС серий Stratix. Технология single-driver также распространяется на входные мультиплексоры и на демультплексоры в соединительных блоках (рис.1.34).

Рассмотрим типовой маршрут проектирования гетерогенных академических ПЛИС, который предполагает использование следующих программных инструментов: ODIN, ABC, T-Vpack, VPR. ODIN конвертирует схемное описание некоторого сложно-функционального устройства (“бенч марк”, тестовая схема на языке Verilog HDL) в специальный файл в .blif формате, в котором выделяет логические вентили для описания логики устройства и “черные ящики” для гетерогенных блоков.

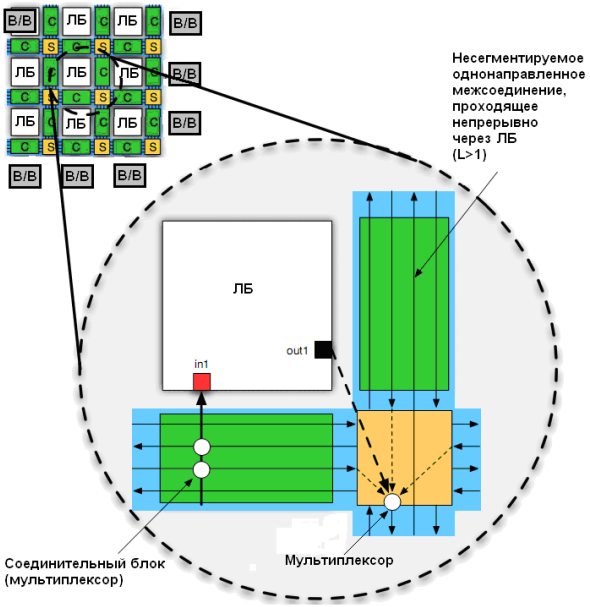


Рис.1.34. Распространение технологии соединений single-driver на соединительные блоки и непосредственное подключение выходов логических блоков к мультиплексорам коммутаторов-маршрутизаторов

Далее с использованием инструмента ABC (существуют и другие программные инструменты минимизации булевых функций в базис ПЛИС типа ППВМ с использованием 4-х входовой LUT-таблицы и D-триггера логического блока, такие как SIS и FlowMap) проводится логическая оптимизация схемы с использованием специального стиля описания независимого от технологии проектирования БИС и ее размещение в логические блоки академической ПЛИС. Выходным так же является файл в .blif формате, в котором выделяются LUT-таблицы, D-триггеры логических блоков и гетерогенные блоки (пример 1).

 # Benchmark "iir1" written by ABC

…

```
.latch      n279 hetero_REGISTER_61_1838_out re iir1_clk_i_0  0
.latch      n284 hetero_REGISTER_61_1839_out re iir1_clk_i_0  0
.latch      n289 hetero_REGISTER_61_1840_out re iir1_clk_i_0  0
.latch      n294 hetero_REGISTER_61_1841_out re iir1_clk_i_0  0
.latch      n299 hetero_REGISTER_61_1842_out re iir1_clk_i_0  0
.latch      n304 hetero_REGISTER_61_1843_out re iir1_clk_i_0  0
...
.names iir1_dat_i_4 iir1_rst_i_0 hetero_REGISTER_61_1906_out n1533 n999
00-10
-0000
.names iir1_dat_i_5 iir1_rst_i_0 hetero_REGISTER_61_1907_out n1533 n1004
00-10
-0000
.names iir1_dat_i_6 iir1_rst_i_0 hetero_REGISTER_61_1908_out n1533 n1009
00-10
-0000
.names iir1_dat_i_7 iir1_rst_i_0 hetero_REGISTER_61_1909_out n1533 n1014
00-10
-0000
.end
.model mult_36
.inputs a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 b0 b1 \
 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15 b16 b17
.outputs c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18 \
 c19 c20 c21 c22 c23 c24 c25 c26 c27 c28 c29 c30 c31 c32 c33 c34 c35
.blackbox
.end
```

Пример 1. Выходной файл инструмента ABC в формате blif

Инструмент T-Vpack перепаковывает файл в .blif-формате (LUT-таблицы и D-триггеры) в кластеры логических блоков (типовой размер кластера 2-12 ЛБ; число входов LUT-таблицы 2-7; длина межсоединений 1-8), которые аналогичны кластерам в ПЛИС FLEX8, 10K или кластерам ПЛИС Virtex, и формирует выходной файл в .net формате для VPR (пример 2), который размещает кластеры логических блоков и гетерогенные блоки 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 трассировочного канала,

быстродействию, экономии площади кристалла и др. (рис.1.35). 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 с учетом задержек распространения сигналов в трассировочных ресурсах ПЛИС.

```
.clb hetero_REGISTER_38_960_out
pinlist: hetero_H_SKEL_14_101_4 iir1_valid_0 n1324 n1325 \
hetero_H_SKEL_14_102_4 hetero_H_SKEL_14_102_5 hetero_REGISTER_38_961_out
\
hetero_H_SKEL_14_102_6 hetero_REGISTER_38_962_out hetero_H_SKEL_14_102_7
\
hetero_REGISTER_38_963_out
hetero_H_SKEL_14_102_8 hetero_REGISTER_38_964_out \
hetero_H_SKEL_14_102_9 hetero_REGISTER_38_965_out open open open open open
\
open open hetero_REGISTER_38_960_out n1414 n1416 n1420 n1422 n1426 n1428 \
n1432 n1434 n1438 iir1_clk_i_0
subblock: hetero_REGISTER_38_960_out 0 1 2 3 22 32
subblock: n1414 4 5 ble_0 6 23 open
subblock: n1416 4 5 ble_0 6 24 open
subblock: n1420 7 8 ble_1 open 25 open
subblock: n1422 7 8 ble_2 open 26 open
subblock: n1426 9 10 ble_3 open 27 open
subblock: n1428 9 10 ble_4 open 28 open
subblock: n1432 11 12 ble_5 open 29 open
subblock: n1434 11 12 ble_6 open 30 open
subblock: n1438 13 14 ble_7 open 31 open
```

Пример 2. Выходной файл инструмента T-Vpack в формате net

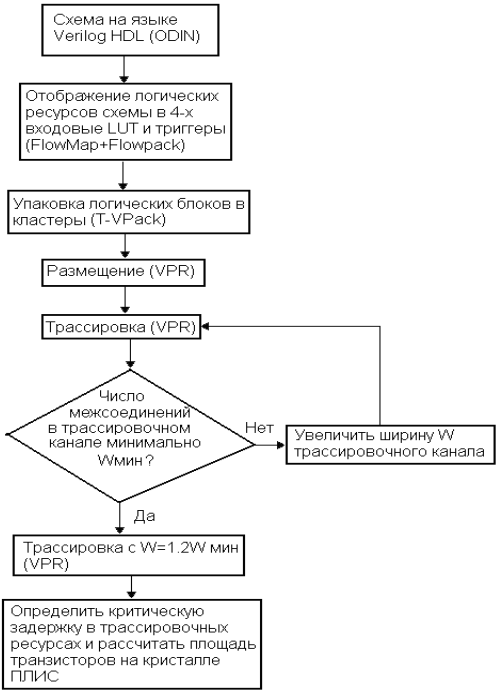


Рис.1.35. Маршрут проектирования ПЛИС типа ППВМ с одноуровневой структурой межсоединений

Для работы этих програм в среде Windows необходимо 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 инсталяции необходимо начать на сайте http://www.cygwin.com и загрузить программу setup.exe, далее с помощью этой программы необходимо скачать следующие приложения: GCC, GDB, X Windows (http://cygwin.com/xfree) и др. Более подробную информацию можно получить на сайте http://www.intuit.ru/department/security/issec/5/.

Ниже показано использование инструментов T-Vpack и VPR для размещения и трассировки тестовой схемы БИХ-

фильтра y[n] = b10*x[n] + b11*x[n-1] + b12*x[n-2] + a11*y[n-1] + a12*y[n-2] (http://opencores.org) в базис гетерогенных (со встроенными блоками перемножителей 36x36) ПЛИС типа ППВМ с одноуровневой структурой межсоединений.

Пример использования программных инструментов.

../T-VPACK_HET/t-vpack.exe iir1.map4.latren.blif iir1.map4.latren.net -inputs_per_cluster 22 -cluster_size 10 -lut_size 4

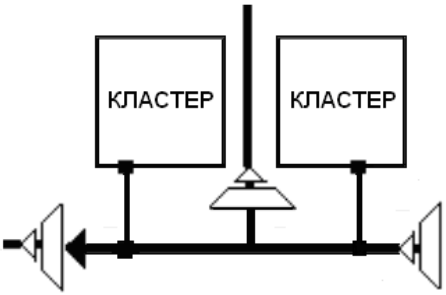
../VPR_HET/vpr.exe iir1.map4.latren.net k4-n10.xml place.out route.out

Информация об архитектуре ПЛИС содержится в файле k4-n10.xml (пример 3). В этом файле содержится: информация о требуемых размерах кристалла; об 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; об соединительных блоках; об 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.

<architecture>
<layout auto="**1.000000**" />
<device>
<sizing R_minW_nmos="**5726.870117**" R_minW_pmos="**15491.700195**" ipin_mux_trans_size="**1.000000**" />
<timing C_ipin_cblock="**1.191000e-14**" T_ipin_cblock="**1.482000e-10**" />
<area grid_logic_tile_area="**30000.000000**" />
< ! Спецификация на трассировочные каналы>
<Задается ширина периферийного трассировочного канала относительно самого широко канала ядра и устанавливается равномерное распределение межсоединений в каналах ядра ПЛИС>
<chan_width_distr>
<io width="**1.000000**" />
<x distr="**uniform**" peak="**1.000000**" />

<y distr="**uniform**" peak="**1.000000**" />
</chan_width_distr>
<! Задается маршрутизатор трассировочных ресурсов типа Wilton с коэффициентом разветвления по выходу Fs=3>
<switch_block type="**wilton**" fs="**3**" />
</device>
<! Пример спецификации ключа для однонаправленных межсоединений >
< Задается соединительный блок на мультиплексорах, сопротивление, входные и выходные емкости, внутренняя задержка ключа, геометрический размер мультиплексора и буфера восстановления уровня в условных единицах площади, привязанных к минимальной ширине (канала) транзистора для получения минимальной площади >
<switchlist>
<switch type="**mux**" name="**0**" R="**94.841003**" Cin="**1.537000e-14**" Cout="**2.194000e-13**" Tdel="**6.562000e-11**" mux_trans_size="**10.000000**" buf_size="**1**" />
</switchlist>

<! Пример спецификации на однонаправленные сегментированные межсоединения >
<! Задаются частота сегментации межсоединения в канале, длина сегмента (сколько кластеров проходит межсоединение без сегментации), направление передачи сигнала (однонаправленное или двунаправленное межсоединение), сопротивление и емкость межсоединения, схема депопуляции (удаление) маршрутизаторов на заданную длину сегмента (1 - есть, 0 – нет маршрутизатора), схема депопуляции подключений межсоединений ко входам кластера с помощью соединительных блоков на заданную длину сегмента >



<segmentlist>

<mux name="**0**" />

<sb type="**pattern**">**1 1 1**</sb>
<cb type="**pattern**">**1 1**</cb>

</segmentlist>
<! Пример спецификации для блоков ввода/вывода >
<! Задается число блоков ввода/вывода на строку/столбец из кластеров ЛБ, входные/выходные задержки, коэффициенты объединения по входу/выходу>
<typelist>
<io capacity="**3**" t_inpad="**7.734000e-11**" t_outpad="**4.395000e-11**">
<fc_in type="**frac**">**0.250000**</fc_in>
<fc_out type="**frac**">**1.000000**</fc_out>
</io>
<! Пример спецификации кластера N=10, K=4, I=22 >
<type name="**.clb**">
<subblocks max_subblocks="**10**" max_subblock_inputs="**4**">
<timing>
<T_comb>
<trow>**1.679000e-10**</trow>
<trow>**1.679000e-10**</trow>
<trow>**1.679000e-10**</trow>
<trow>**1.679000e-10**</trow>
</T_comb>
<T_seq_in>
<trow>**-3.990000e-11**</trow>
</T_seq_in>
<T_seq_out>
<trow>**1.261000e-10**</trow>
</T_seq_out>
</timing>
</subblocks>
<fc_in type="**frac**">**0.250000**</fc_in>
<fc_out type="**frac**">**1.000000**</fc_out>

Пример 3. Фрагмент xml-файла с описанием архитектуры ПЛИС типа ППВМ

На рис.1.36 показана гетерогенная архитектура ПЛИС типа ППВМ с одноуровневой структурой межсоединений размером 10x10 кластеров и встроенными блоками перемножителей 36x36 (9 шт) в VPR5.0. Каждый кластер

состоит из 10 логических блоков (ЛБ), каждый блок состоит из 4-х входовой LUT-таблицы и триггера. По периферии кристалла располагаются блоки ввода/вывода. В базис ПЛИС размещена тестовая схема БИХ-фильтра. Задействованные блоки перемножителей для реализации БИХ-фильтра отображены оранжевым цветом. Трассировочные ресурсы ПЛИС не подвергаются оптимизации. На рис.1.37 показаны электрические связи между функциональными блоками ПЛИС.

На рис.1.38 показана архитектура ПЛИС после оптимизации (размещение и трассировка). Зеленым цветом показан выделенный кластер из ЛБ (блок 89 n1071 с координатами (6,8)). Синим цветом – функциональные блоки (1 перемножитель, 3 кластера и 5 блоков ввода/вывода) которые связанны с коэффициентом объединения по входу F_{Cin} (показывается число межсоединений в трассировочном канале, которые могут быть подключены ко входу функционального блока, задается в долях от W) выделенного кластера; красным цветом – блоки которые связаны с коэффициентом разветвления по выходу F_{Cout} выделенного кластера (1 перемножитель, 7 кластеров). На рис.1.39 показана ПЛИС с общими трассировочными ресурсами после размещения и трассировки связей между блоками.

На рис.1.40 показаны общие трассировочные ресурсы, которые заложены в структуру программируемых межсоединений. Выделенный кластер показан ярко зеленым цветом. Темно-зеленым цветом показаны направления соединений межсоединений в маршрутизаторах. Синим цветом (синие линии и квадраты) – подключение трассировочных ресурсов ко входам кластера с помощью соединительных блоков (мультиплексоров). Соединительные блоки подключают входы кластера к трассировочным каналам.

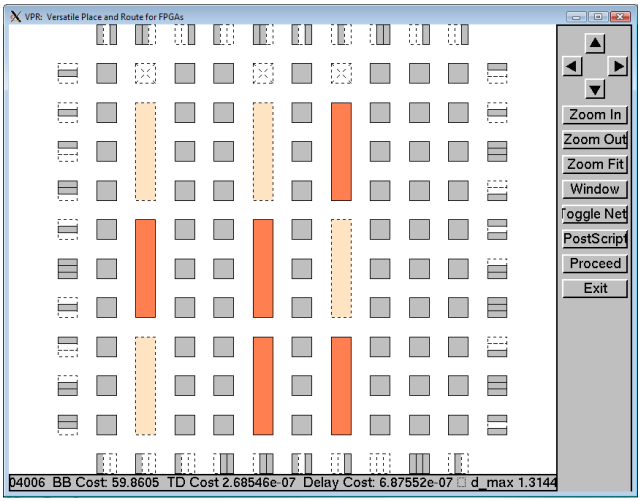


Рис.1.36. Интерфейс инструмента VPR. Гетерогенная архитектура (со встроенными перемножителями) ПЛИС с одноуровневой структурой межсоединений (4-х входовая LUT-таблица, размер кластера 10 логических блоков)

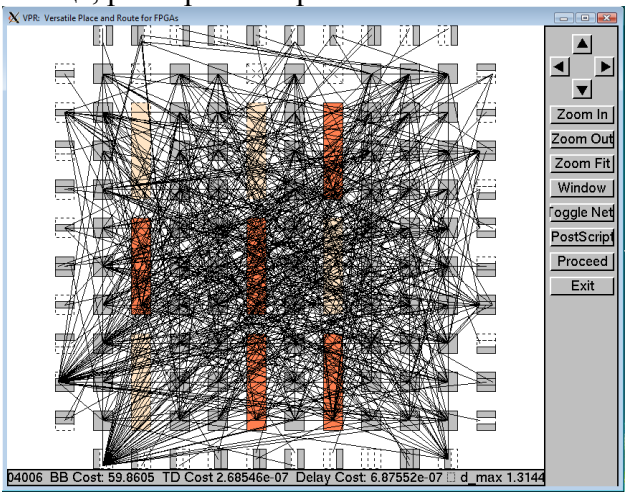


Рис.1.37. Гетерогенная архитектура ПЛИС с одноуровневой структурой межсоединений (4-х входовая LUT-таблица, размер кластера 10 логических блоков) со связями между функциональными блоками

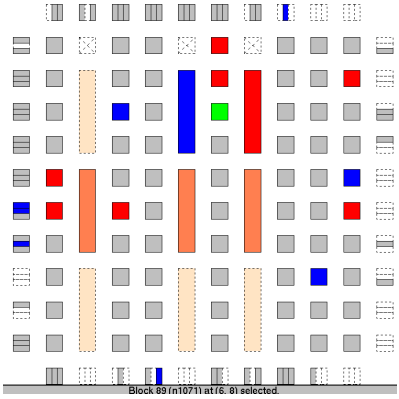


Рис.1.38. Гетерогенная архитектура ПЛИС с одноуровневой структурой межсоединений (4-х входовая LUT-таблица, размер кластера 10 логических блоков) после операций размещения и трассировки

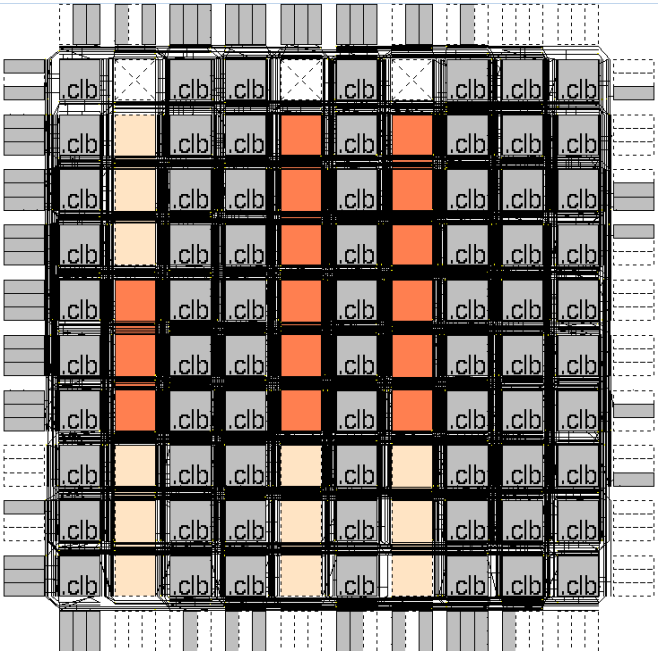


Рис.1.39. Гетерогенная ПЛИС с общими трассировочными ресурсами

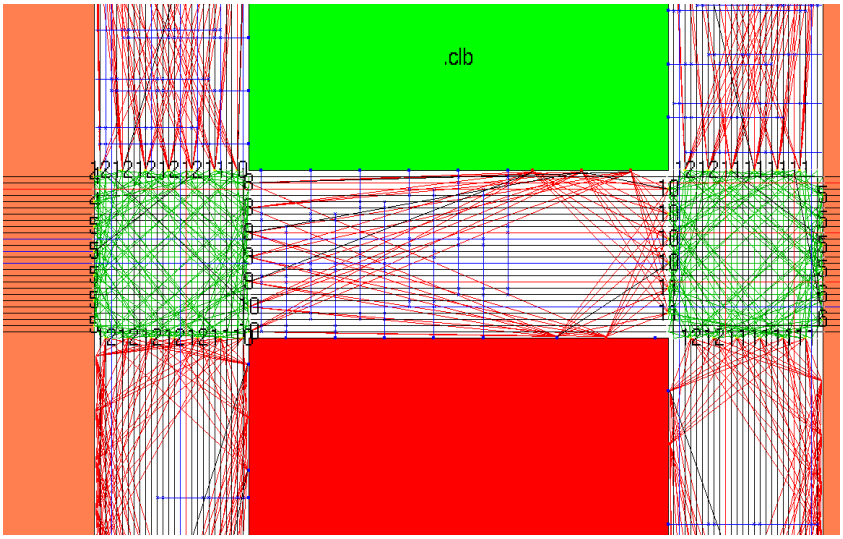


Рис.1.40. Общие трассировочные ресурсы гетерогенной ПЛИС

Выходы кластера по методологии соединений single-driver switch block напрямую подключаются к мультиплексорам маршрутизаторов. Для соединительных блоков (реализуются на мультиплексорах) задается коэффициент разветвления по входу $F_{Cin} = 0.25$. Для кластеров ядра ПЛИС выходы логических блоков подключаются к 13 мультиплексорам маршрутизаторов находящиеся слева (7 мультиплексоров) и с права (6 мультиплексоров) от выделенного кластера с коэффициент разветвления по выходу $F_{Cout} = 1$ (рис.1.40).

Красным цветом (красные линии и квадраты) показаны выходы кластера, межсоединения связанные с коэффициентом разветвления по выходу с выделенным кластером и избыточные (неиспользуемые) трассировочные ресурсы.

На рис.1.41 показаны только те трассировочные ресурсы, которые необходимы для реализации БИХ-фильтра в базисе гетерогенной ПЛИС. Синие линии (соединительные блоки и канальные трассировочные ресурсы) связаны с

коэффициентом объединения по входу (синие крестики), а красные линии и красные квадраты (выходы кластера) связаны с коэффициентом разветвления по выходу. Выходы кластера подключаются к трассировочным ресурсам с помощью буферов с третьим состоянием (тристабильная логика). На рис.1.41 связь красных квадратов (выходов) и красных линий (канальные трассировочные ресурсы) осуществляется черными линиями и черными треугольниками (буферы). Черным цветом показаны входы/выходы кластеров не связанные с выделенным кластером, так же этим цветом показываются канальные трассировочные ресурсы и различные межсоединения (в маршрутизаторах и в соединительных блоках) относящиеся к задаче размещения БИХ-фильтра в базис ПЛИС.

Горизонтальный и вертикальный трассировочный канал состоит из 26 однонаправленных межсоединений, т.е. ширина канала W=26 (рис.1.42). Канал разделяет коммутатор-маршрутизатор типа Wilton с коэффициентом разветвления $F_S = 3$, который обладает лучшей разводимостью (большее число путей маршрутизации в пределах заданного направления) и позволяет организовать длинные межсоединения. VPR 5.0 позволяет использовать 3 типа маршрутизаторов: subset, wilton, universal.

Треугольники (серые и черные) по периферии маршрутизатора показывают направление передачи сигналов по межсоединениям. Серые треугольники по периферии маршрутизатора указывают на то, что пара разнонаправленных межсоединений проходит через маршрутизатор непрерывно (без использования ключей в горизонтальном или вертикальнм направлениях). В рассматриваемом примере, в каналах, межсоединения непрерывно проходят через 2 кластера, в этом случае $L = 2$.

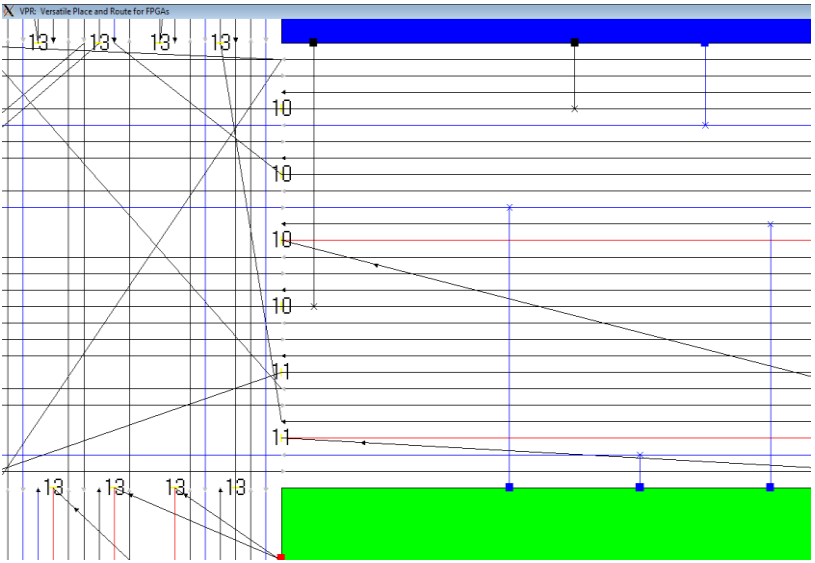


Рис.1.41. Фрагмент коммутатора-маршрутизатора (левый верхний от блока 89) типа Wilton, горизонтальный трассировочный канал из однонаправленных сегментов и трассировочные ресурсы задействованные для реализации БИХ-фильтра в базисе ПЛИС

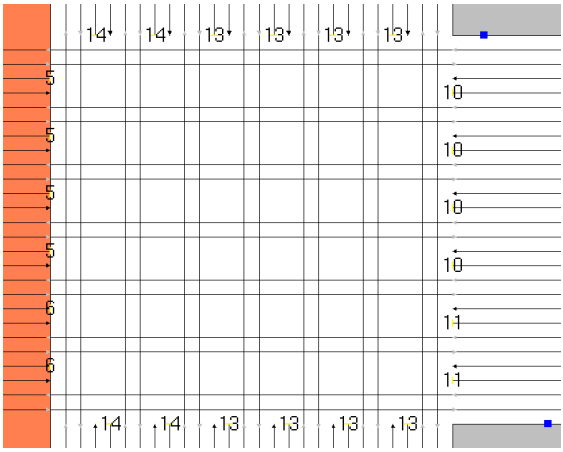
Желтой черточкой и черной стрелкой показаны места сегментации пары разнонаправленных межсоединений на периферии маршрутизатора. В позиции желтых черточек осуществляется подключение выходов соседних кластеров и выходов гетерогенных блоков (перемножителей). Цифрами обозначена разрядность мультиплексоров. Например, цифра 13, это мультиплексор 13 в 1 (рис.1.42).

Рассмотрим вертикальный трассировочный канал и левый верхний маршрутизатор от блока 89, находящиеся в ядре кристалла (рис.1.42, а). С каждой стороны маршрутизатора имеется 13 входящих межсоединений, из них 7 входящих непрерывных (серые треугольники, обращенные острием в маршрутизатор) и 6 входящих сегментированных

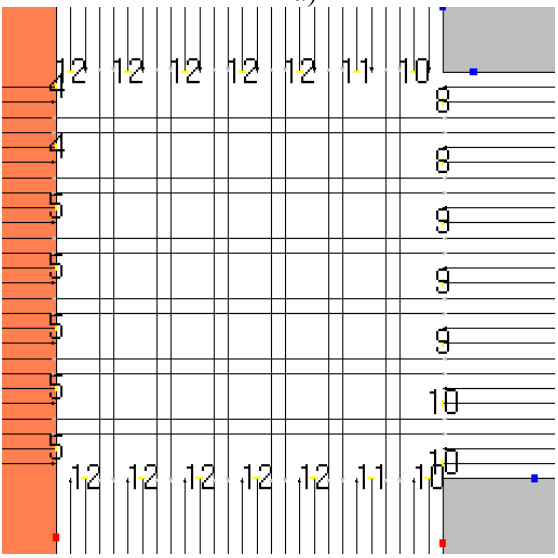
межсоединений (черные треугольники, обращенные острием в маршрутизатор) и 13 исходящих межсоединений: 7 исходящих непрерывных (серые треугольники, обращенные острием от маршрутизатора) и 6 исходящих сегментированных (желтые черточки).

Все входящие соединения имеют коэффициент разветвления $F_S = 3$, т.е. входящие межсоединение в маршрутизаторе разветвляется на 3 направления, однако, вслучае сегментированных входящих соединений в каждом из направлений имеется буферизованный ключ, а в случае входящих несегментированных межсоединений, горизонтальные или вертикальные направления несегментированы ключами (рис.1.42, *б*). В левом нижнем маршрутизаторе ситуация меняется на противоположную. С каждой стороны маршрутизатора имеется 13 входящих межсоединений, из них 6 входящих непрерывных и 7 входящих сегментированных межсоединений и 13 исходящих межсоединений: 6 исходящих непрерывных и 7 исходящих сегментированных. Таким образом в первом случае с каждой стороны по 6 а во втором по 7 мультиплексоров по методологии single-driver.

Рассмотрим коммутаторы-маршрутизаторы располагающиеся на периферийных трассировочных каналах ПЛИС. На рис.1.43, *а* показан коммутатор-маршрутизатор, расположенный в центре периферийного вертикального трассировочного канала. Слева кластеры логических блоков, а справа блоки ввода/вывода. В 13 мультиплексоров небольшой разрядности (4 и 5) левой стороны маршрутизатора осуществляется подключение выходов кластеров и блоков ввода/вывода. На рис.1.43, *б* показан коммутатор-маршрутизатор в углах пересечения горизонтального и вертикального периферийного трассировочных каналов.

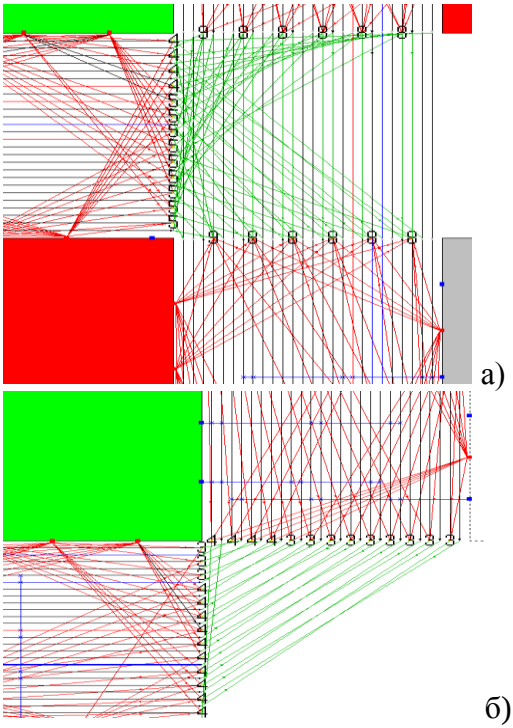


а)



б)

Рис.1.42. Коммутатор-маршрутизатор (расположен в центре ядра кристалла ПЛИС) построенный по методологии single-driver: а) -левый верхний от блока 89 (6 мультиплексоров с каждой стороны); б) - левый нижний от блока 89 (7 мультиплексоров с каждой стороны)



а)

б)

Рис.1.43. Коммутатор-маршрутизатор с общими трассировочными ресурсами: а) – расположен в центре периферийного вертикального трассировочного канала; б) – на пересечении горизонтального и вертикального периферийного трассировочного канала

Ведущие мировые дизайн-центры (более 200) и учебные образовательные центры (более 1000) широко используются программные инструменты T-Vpack и VPR как для проектирования, так и для исследования новых архитектур ПЛИС типа ППВМ.

В настоящее время разработчики как коммерческих, так и академических ПЛИС пришли к выводу о целесообразности использования однонаправленных сегментированных межсоединений различной длины в трассировочных каналах и использования мультиплексорных структур в соединительных блоках и коммутаторах-маршрутизаторах, что позволяет получать существенный выигрыш по быстродействию и по площади кристалла.

2. ТРЕХМЕРНЫЕ ИНТЕГРАЛЬНЫЕ СХЕМЫ

2.1. Проблемы, связанные с проектированием БИС по субмикронным проектным нормам, и методы их решения

С наступлением эры субмикронных технологий БИС стали работать на высоких частотах, потреблять больший ток и мощность при меньших напряжениях питания. Обострились паразитные эффекты (паразитная емкость связи между проводниками, приводящая к перекрестным искажениям, электромиграция, времязависимый пробой подзатворных оксидов, паразитное падение напряжения в цепях питания и заземления, паразитные индуктивные эффекты), которые не учитывались при конструировании БИС предыдущего поколения. В субмикронных условиях проблема взаимосвязи таких параметров, как скорость, потреблемая мощность, целостность сигналов и надежность стала столь же актуальной, как и проблема снижения площади кристалла для БИС предыдущего поколения.

Это привело к более яркому проявлению эффекта паразитной емкостной связи (рис.2.1). Кроме того, масса других паразитных эффектов, которые можно было не учитывать в проектах предыдущего поколения, стали ключевыми факторами для обеспечения правильного функционирования и высокой производительности новых БИС повышенной плотности. Например, для субмикронных БИС характерны такие паразитные эффекты, как, например, преобладание задержек распространения сигналов по токопроводящим дорожкам над задержками распространения сигналов в вентилях из за наличия собственных сопротивлений и емкостей (RC-характеристики).

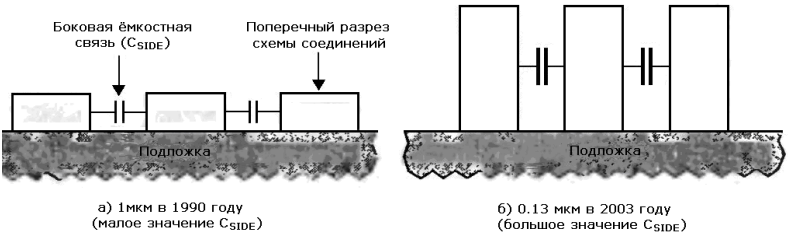


Рис.2.1. Увеличение боковой емкости с уменьшением размеров токопроводящих дорожек субмикронных БИС

Уменьшение геометрических проектных норм привело к значительному увеличению емкости C_{SIDE} между боковыми стенками соседних дорожек (емкость боковой связи) по сравнению с емкостью между основанием проводника и подложкой кристалла C_{AREA} и емкостью между боковой стенкой проводника и подложкой C_{FRINGE}. Более того, для БИС с 6 и более слоями металлизации характерно появление существенной по величине емкостной связи между соседними слоями C_{CROSS} (рис.2.2).

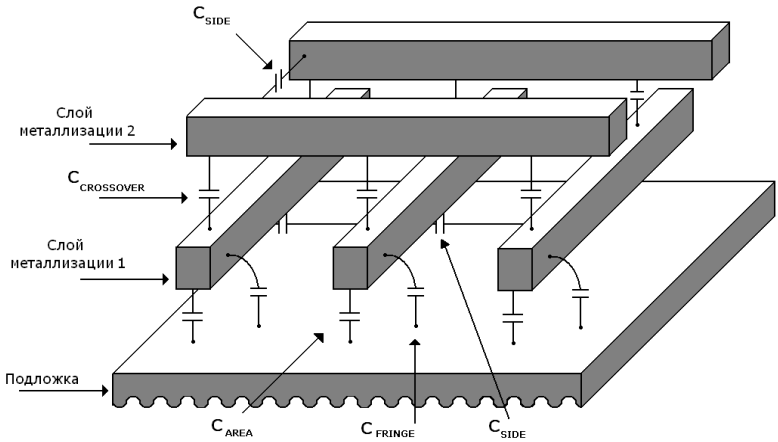


Рис.2.2. Емкостные явления, связанные с внутренними проводниками БИС

Мерой оценки эффекта перекрёстных искажений является отношение ёмкости боковой связи, возникающей между проводниками, расположенными на одном слое C_{SIDE}, к ёмкости межслойной связи C_{CROSS}, возникающей между проводниками, расположенными на разных слоях. Эффект взаимосвязи становится более видным, поскольку развитие технологии приводит к использованию геометрических объектов меньшего размера. Согласно докладам, представляемым на регулярно проходящей международной конференции International Technology Roadmap for Semiconductors, ёмкость боковых связей для технологий 1999 года превышала ёмкость межслойных связей почти в три раза, а к 2006 году это соотношение достигло пяти (рис.2.3).

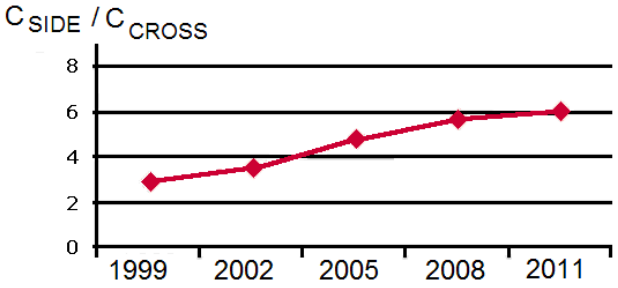


Рис.2.3. Рост отношения ёмкости боковой связи, возникающей между проводниками, расположенными на одном слое C_{SIDE}, к ёмкости межслойной связи C_{CROSS} в субмикронных БИС

На рис.2.4 представлена современная КМОП-структура с двумя n- и p-карманами по 0.18 мкм проектным нормам с одним уровнем поликремния и шестью уровнями алюминиевой металлизации (AlCu (0.5 % Cu) с подслоем Ti) и напряжением питания ядра 1.8 В кремниевой фабрики X-FAB Semiconductor Foundries работающей в режиме “foundry”. XFAB Semiconductor Foundries AG (Германия) - ведущая

группа предприятий полупроводникового производства специализирующаяся на выпуске кристаллов смешанных аналого-цифровых БИС по субмикронным проектным нормам (табл.2.1).

В технологическом маршруте используются поликремниевые затворы и глубокая изоляция канавками. В n-кармане формируются p-МОПТ, а в p-кармане – n-МОПТ. По КТТ минимальная длина n- и p-МОПТ с индуцированными каналами составляет 0.18 мкм, а минимальная ширина 0.22 мкм.

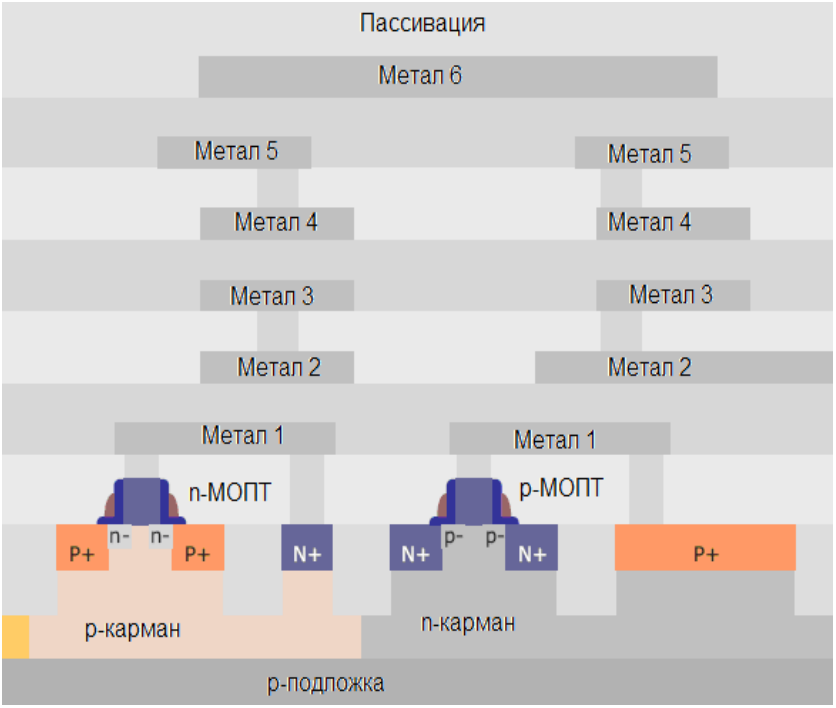


Рис.2.4. Сечение КМОП структуры с двумя карманами n- и p-типа проводимости в p-подложке

Таблица 2.1

Конструктивно-технологические требования кремниевой фабрики XFAB в КМОП-технологическом процессе XC018

Топологический слой	Технология 0.18 мкм		
	Ширина проводника, мкм	Толщина проводника, мкм	Минимальное расстояние, мкм
Металл1	0.23	0.17	0.23
Металл2	0.28	0.22	0.28
Металл3	0.28	0.25	0.28
Металл4	0.28	0.25	0.28
Металл5	0.28	0.25	0.28
Металл6	0.44	0.35	0.46

Сопротивление токопроводящей дорожки определяется по формуле:

$$R = \frac{rl}{S} = \frac{rL}{HW} = \rho\frac{L}{W},$$

где r – удельное сопротивление, при $20\ ^0C$ для Al-металлизации $r = 2.7*10^8$ Ом*м; H – константа технологии (толщина проводника); ρ – удельное поверхностное сопротивление, Ом/квадрат .

Согласно упрощенным представлениям, для субмикронных БИС, модель емкости токопроводящих дорожек складывается из емкости параллельных пластинок (относительно нижележащих токопроводящих дорожек или относительно земли) и краевой емкости (рис.2.5). Сосредоточенная емкость проводника определяется по следующей формуле: $C_i = \frac{\varepsilon_{ox}}{t_{ox}}WL$, где W – ширина; L – длина проводника; t_{ox} – толщина окисла (межслойного диэлектрика). Из формулы следует, что емкость прямопропорциональна

перекрытию проводников и обратно пропорциональна расстоянию между ними. В субмикронных БИС отношение $W/H<1$ поэтому модель параллельных пластинок становится не точной и емкость между боковыми стенками токопроводящих дорожек и подложкой (краевая емкость) уже нельзя игнорировать.

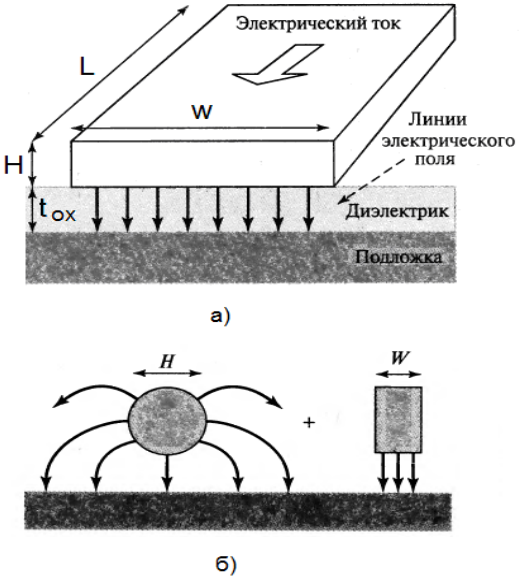


Рис.2.5. Токопроводящая дорожка (а) и модель емкости токопроводящей дорожки: емкость параллельных пластинок и краевая емкость, моделируемая цилиндрическим проводником, диаметр которого равен толщине дорожки

Для расчета паразитной емкости прямоугольных токопроводящих дорожек в субмикронных БИС используют следующую аппроксимацию:

$$C_{wire} = C_s + C_p = \frac{w\varepsilon_{ox}}{t_{ox}} + 2\frac{\pi\varepsilon_{ox}}{\log(t_{ox}/H)},$$

где C_s – удельная поверхностная емкость проводника на единицу длины и пластиной заземления; C_p – краевая емкость; $w = W - H/2$.

На практике используют более простую формулу для вычисления емкости токопроводящей дорожки:

$$C = C_s + C_p = (\sigma_s \cdot l \cdot w) + 2 \cdot l \cdot \sigma_{p66},$$

где σ_s – удельная поверхностная ёмкость с нижележащим металлом, аф/мкм^2; σ_{p66} – краевая емкость или емкость периметра. Множитель 2 в формуле учитывает две стороны токопроводящей дорожки при расчете краевой емкости, а ее толщиной пренебрегают. Удельные и краевые емкости берутся из технологических файлов кремниевых фабрик. Если рассматриваемая токопроводящая дорожка находится в двух верхних слоях металлизации, например, в шестом и пятом слое металлизации, то:

$$C = C_s + C_p = (\sigma_s \cdot l \cdot w) + 2 \cdot l \cdot \sigma_{p66} + 2 \cdot l \cdot \sigma_{p65}.$$

Если перекрытие по площади с нижележащим металлом составляет от 10 до 50 %, то это учитывается коэффициентом в емкости параллельных пластинок:

$$C = 0.5 \cdot C_s + C_p .$$

Если расстояние между проводниками 2 мкм, а минимальное расстояние по КТТ 0.46 мкм, то влияние краевой емкости ослабляется в 4.35 раза:

$$C = 0.5 \cdot C_s + C_p / 4.35.$$

Рассмотрим пример, алюминиевая токопроводящая дорожка длиной 10 см и шириной 1 мкм располагается на кристалле размером порядка 1-2 см. Например, для 6 слоя AlCu-металлизации $\sigma_s = 34$ аф/мкм^2; краевая емкость $\sigma_{p66} = 116$, аф/мкм. Общая емкость составит:

$$C = C_s + C_p = 34\ \text{аф/мкм}^2 * 0.1 * 10^6\ \text{мкм}^2 + 2 * 0.1 * 10^6\ \text{мкм} * 116\ \text{аф/мкм} =$$
$$= 3.4\ \text{пФ} + 23.2\ \text{пФ} = 26.6\ \text{пФ}$$

Из расчета следует, что краевой емкостью в субмикронных БИС пренебрегать нельзя.

При увеличении длины шин синхронизации, паразитная емкость может вносить существенный вклад в перекос значений времен t_{LH} и t_{HL}. Рассчитаем задержку распространения тактового сигнала в RC-цепи первого порядка. Предположим, что паразитная емкость RC-цепи C заряжена до уровня напряжения питания U_{CC}, а на входе цепи действует перепад напряжения от U_{CC} до 0, то переходной процесс разряда в емкости описывается экспоненциальной функцией вида ($\tau = RC$):

$$U_{\text{вых}} = U_{CC} e^{-t_{HL}/RC},\ t_{HL} = -RC \ln \frac{U_{\text{вых}}}{U_{CC}} = -0.69\tau .$$

Для шины синхронизации, между двумя блоками, при $U_{вых} = U_{CC}/2$, с параметрами RC-цепи $R = 216$ Ом, $C = 1904$ фФ задержка времени спада фронта синхросигнала t_{HL} составляет 0.29 нс.

Аналогично рассчитывается время нарастания фронта сигнала, когда на входе действует перепад напряжения с 0 до U_{CC}, а паразитная емкость разряжена до напряжения нуля:

$$U_{\text{вых}} = U_{CC}(1 - e^{-t_{LH}/RC}),\ t_{LH} = -RC \ln \frac{U_{\text{CC}} - U_{вых}}{U_{CC}} = 0.69\tau .$$

Технология ученых из NIST является вариантом технологии, известной, как «дамасская металлизация», которая часто используется для изготовления сложной трехмерной разводки в БИС, системы соединения электрических элементов в многослойных структурах.

Технология, названная по имени известной древней арабской технологии изготовления высокоуглеродистой стали для клинков, предполагает вытравливание на поверхности подложки горизонтальных бороздок и вертикальных перемычек, которые затем заполняются медью методом

гальванопластики. Размеры бороздок могут колебаться от десятков нанометров до сотен микронов. После заполнения подложка полируется для удаления излишков меди.

Основным достоинством метода дамасской металлизации является то, что металл полностью заполняет бороздки, не оставляя пустот. Для этого в электролит добавляется вещество, которое препятствует слишком быстрому оседанию молекул металла на стенках бороздок. Однако картина нанесения становится иной, когда наносится не медь, а магнитный материал. Ученым из NIST удалось оптимизировать эту технологию для создания ферромагнитных наноструктур. С помощью данной технологии станет возможным изготовление сложных микроэлектромеханических устройств, в которых для соединения магнитных материалов с немагнитными компонентами могут использоваться широко распространенные в электротехнике методы.

Внутренними называются задержки, которые свойственны внутренней архитектуре логических функций. Под внешними задержками подразумевают задержки, относящиеся к межблочным соединениям внутри кристалла БИС. В БИС предыдущего поколения внутренние задержки преобладали над внешними. Например, в 2 мкм технологии внутренние задержки составляли величину порядка двух третей от общей задержки (рис.2.6). Но с уменьшением технологических проектных норм внешняя задержка стала расти и приобрела доминирующее влияние (рис.2.6). При переходе на новые технологические поколения рост доли внешних задержек сохранится и в современных БИС по субмикронным и нанометровым проектным нормам доля внешних задержек может составлять до 80 % и более от величины общей задержки.

Задержки типа вывод-вывод и точка-точка являются современными названиями внутренних и внешних задержек

соответственно. Задержка вывод-вывод характеризует собой время от прихода воздействия на вход вентиля до появления соответствующей реакции на его выходе, а точка-точка описывает время распространения сигнала между выходом источника сигнала и входом нагрузки (рис.2.7).

Задержка вывод-вывод обычно равна времени от момента достижения входным сигналом порога срабатывания до начала соответствующей реакции (отклика) на выходе вентиля, а задержка точка-точка определяется от начала соответствующей посылки источника сигнала до достижения ею порога срабатывания нагрузки.

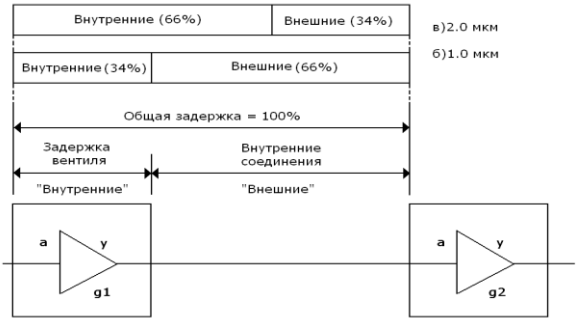


Рис.2.6. Внутренние и внешние задержки в БИС различного технологического поколения

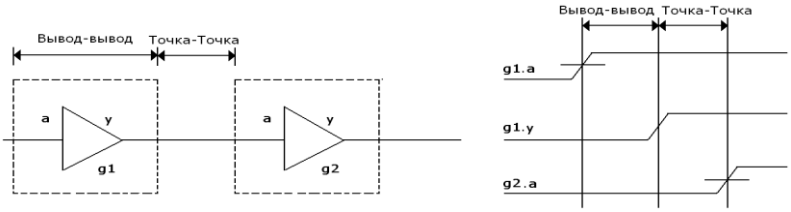


Рис.2.7. Задержки вывод-вывод и точка-точка в БИС

Для упрощения расчетов задержек вывод-вывод будем считать, что отсчитываются они от времени, когда входной сигнал достигает уровня 50 % от разницы между 0 и 1, исходя

из предположения, что порог срабатывания у вентилей составляет 50 % (рис.2.8).

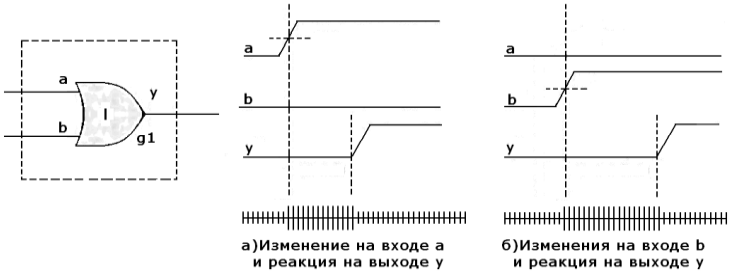


Рис.2.8. Зависимость задержки вывод-вывод от пути прохождения сигнала

Данный пример показывает, что в современных субмикронных БИС каждый путь прохождения сигнала от входа до выхода характеризуется своим значением задержки вывод-вывод.

Крутизна (наклон) импульса синхронизации представляет скорость изменения сигнала при его переходе с логического 0 на уровень логической 1 и наоборот (рис.2.9). При мгновенном переходе обеспечивается максимально возможное значение крутизны.

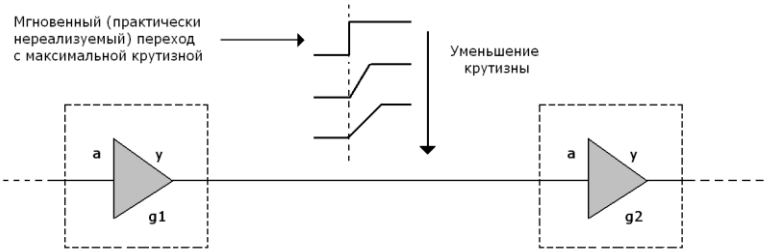


Рис.2.9. Крутизна сигнала характеризует время, необходимое для перехода сигнала с уровня на уровень

В субмикронных БИС и ПЛИС фронт сигнала с большой крутизной может вызвать быстроменяющийся отклик на выходе (рис.2.10, *а*), а фронт сигнала с малой крутизной может привести к появлению медленноменяющегося выходного сигнала (рис.2.10, б).

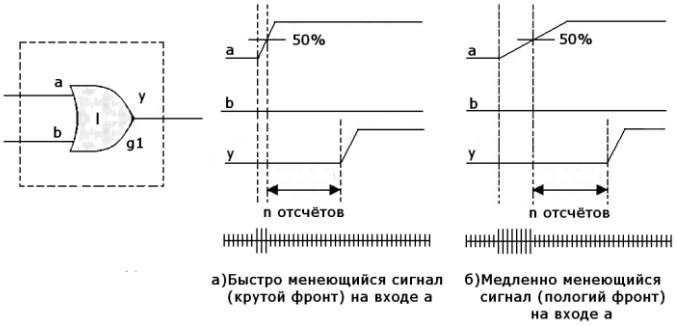


Рис.2.10. Зависимость выходной характеристики вентиля от крутизны входного сигнала

Уменьшение размеров (идеальное масштабирование, $\alpha > 1$) при постоянном электрическом поле в канале МОПТ приводит не только к уменьшению мощности рассеяния, площади вентиля и времени задержки, но и к ухудшению, например, плотности тока через соединительную дорожку, и контакт при масштабировании возрастают в α^2 раз, возрастает также и сопротивление соединений при уменьшении размеров.

Чтобы оказать противодействие паразитным явлениям при уменьшении размеров элементов на кристалле, напряжения питания КМОП БИС не снижались в строгом соответствии с законом масштабирования при постоянном электрическом поле. Однако сильные электрические поля, возникающие при уменьшении размеров МОПТ, при сохранении питающего напряжения приводят к появлению горячих носителей (Hot Carrier Injection, HCI). Эти горячие носители могут вызвать проблемы, связанные с необходимостью обеспечения

надежности в течение длительной работы. В соответствии с принципами масштабирования КМОП ИС толщина подзатворного оксида T_{OX} непрерывно уменьшается. Для технологического поколения 0.05-0.25 мкм напряженность электрического поля в оксиде не превышает 5 МВ/см. Практически допустимая минимальная толщина термического оксида составляет 3.5 нм, при этом плотность дефектов достигает уровня 0.5 $см^{-2}$ при напряженности поля в оксиде $E_{OX} = 7.5$ МВ/см.

Для субмикронных КМОП БИС характерны: сдвиг во времени порогового напряжения ΔV_t, деградация крутизны Δg_m, деградация тока стока $\Delta I_C / I_C$, лавинный пробой между истоком и стоком, наличие туннельной компоненты тока утечки стокового перехода при высокой напряженности вертикального электрического поля в области перекрытия стока затвором (GIDL-эффект) и др.

Снижение напряжения питания V_{DD} в значительной степени уменьшает влияние эффекта горячих носителей на долговременную стабильность параметров КМОП БИС. Величина максимального напряжения питания, при котором напряженность продольного электрического поля вблизи стока не превышает критической величины E_{cr}, снижающей срок службы прибора ниже 10 лет, дается соотношением:

$$V_{DD} = V_{СИнас}(L_{eff}) + E_{cr}(0.2T_{OX}^{1/3} X_j^{1/2} + l_{LDD}),$$

где $V_{СИнас}$ – напряжение насыщения стока; L_{eff} – эффективная длина канала; l_{LDD} – эффективная длина LDD-области (слаболегированные области стока); X_j – глубина залегания $p-n$ – перехода. При уменьшении L_{eff} в 2 раза напряжение насыщения стока может быть снижено не более чем на 0,5 В, следовательно, V_{DD} также будет снижено на 0,5 В. Наличие LDD- области дает возможность увеличить напряжение

питания на 30 % и более, т.е. на величину $E_{cr} l_{LDD}$. Допустимая с точки зрения надежности (обусловленной горячими носителями) величина E_{cr} составляет $3*10^5$ В/см.

Установлено, что использование LLD-областей для субмикронных КМОП БИС позволяет: ослабить чувствительность порогового напряжения V_t к эффективной длине канала L_{eff}; ослабить величину DIBL-эффекта, характеризующего значение сдвига порогового напряжения ΔV_t, вызываемого большим напряжением на стоке V_C; подавить GIDL-эффект; увеличить напряжение питания V_{DD} для обеспечения высокого быстродействия БИС.

Сквозной маршрут проектирования БИС с учетом надежности требует разрушения барьеров между уровнями проектирования: логическим проектированием, синтезом, физическим проектированием и физической верификацией. Вариант размещения топологических ячеек, явившийся результатом учёта одной проблемы (например, нарушений, выявляемых при временном анализе), может обострить и другую (например, вывести за пределы допустимых отклонений параметры целостности сигналов). Большая точность любого отдельно взятого анализа не может быть достигнута за счёт ухудшения других аспектов всего проекта. Каждый инструмент проектирования должен быть снабжён определёнными данными от других инструментов, чтобы хорошо выполнять свою задачу, не нарушая других критериев. И в то же время эти инструменты должны обеспечивать соответствующий уровень обмена данными друг с другом без повторного проведения вычислений и генерации данных.

Например, анализ эффектов электромиграции на постопологическом этапе проектирования является в субмикронных БИС запоздалым. Электромиграция традиционно рассматривалась как проблема ИС с широкими

шинами питания и решалась на заключительных этапах проектирования. В современных БИС обострилась проблема электромиграции, связанная с повышенной плотностью тока в сигнальных цепях и цепях синхронизации и межслойных соединений. Экспертиза схемы и ее топологической реализации увеличивает время цикла проектирования. Увеличение ширины проводников в случае выявленных проблем может быть просто невозможным, а это означает возвращение на этап разработки топологии, переработку с учетом результатов, полученных в предыдущем цикле проектирования. Наилучшим решением проблем электромиграции является определение правильной ширины проводников в процессе их размещения. В субмикронных БИС необходим учет тепловых эффектов, связанных с электромиграцией, известных как эффекты саморазогрева, особенно в цепях сигналов и синхронизации.

Проблемы, связанные с целостностью сигналов, также необходимо решать до топологического проектирования или во время трассировки и размещения. Целостность сигналов охватывает физические эффекты, проявляющиеся при проектировании БИС и приводящие к неправильному функционированию. Наиболее заметным является эффект паразитной емкостной связи между проводниками, приводящий к возникновению в них перекрестных искажений. Эти эффекты почти не проявлялись при использовании технологии с шириной проводников 0.5 мкм и больше. По мере повышения плотности схем и перехода к глубоким субмикронным проектным нормам емкостная связь начинает расти вследствие сближения проводников и роста их относительной толщины.

В САПР БИС используются дополнительные программные модули, позволяющие решать проблемы питания, целостности сигналов, электромагнитной интерференции, электромиграции металла, долговечности

оксидов и т.д. Сквозной маршрут проектирования БИС с учетом надежности поддерживают САПР фирм Cadence Design System, Synopsys, Mentor Graphics, Avante! (Compass Design Automation).

Экспериментальную интенсивность отказов БИС и ПЛИС ведущие фирмы-изготовители (например, Siemens AG, Analog Devices (ADI), Atmel, Xilinx, Altera, QuickLogic, Actel) рекомендуют оценивать по результатам ЭТТ (предпочтительно, динамической) по формуле: $\lambda = \frac{\chi^2(P^*, m)}{2N\Delta t K_y} * 10^9 = \frac{U}{N\Delta t K_y} * 10^9$, $U = \frac{\chi^2(P^*, m)}{2}$, где χ^2 – распределение хи-квадрат (табулированная величина, зависящая от доверительной вероятности и числа отказов); P^* – доверительная вероятность (обычно выбирается из диапазона 0,5-0,95), связанная с уровнем значимости CL соотношением $P^* = 1 - \frac{CL\,\%}{100}$; $m = (2n+2)$-число степеней свободы, где n – количество отказавших ИС; N – общее число испытуемых ИС; K_y – обобщенный коэффициент ускорения; Δt – время испытаний; $N\Delta t K_y$ – приведенное полное время испытаний или эквивалентные приборо-часы (в зарубежной литературе принята аббревиатура EDH Equivalent Device Hours). Использование этой формулы предполагает рассмотрение величины U как случайной величины, распределенной по закону χ^2. Табл.2.2 показывает результаты испытаний ПЛИС фирмы Altera по методике HTOL (V_{CC}+20 %, 1000 ч, температура испытаний: минимально 125 ^0C, максимально 140 ^0C).

Таблица 2.2

Результаты испытаний ПЛИС фирмы Altera (методика HTOL, MIL-STD-883)

Семейство ПЛИС	Проектные нормы, мкм, технология	Напряжение питания при испытаниях, V_{CC}, В	Интенсивность отказов λ, ФИТ
FLEX8000, FLEX10K	0.5/0.42 3 слоя мет., по техн. СОЗУ	6.0+20 %	35.1
FLEX10KA, FLEX6000A	0.3/0.35 4 слоя мет., по техн. СОЗУ	4.0+20 %	48.3
FLEX10KB,10KE, APEX20K	0.22 5 слоев мет., по техн. СОЗУ	3.01+20 %	24
APEX20KE	0.18 до 8 слоев мет., по техн. СОЗУ	2.3+25 %	56.3
APEX20KC, APEXII	0.15 до 8 слоев Cu-мет.	2.3+25 % 1.8+20 %	85.9
CPLD / MAX3000A	0.35 4 слоя мет. по техн. ЭСППЗУ	4.0+20 %	25.8
CPLD / MAX7000B	0.22 5 слоев мет. по техн. ЭСППЗУ	3.0+20 %	30
Стандартное (EP910,610)	по техн. СППЗУ УФ	6.0+20 %	26.8

По результатам тестирования среднее время наработки на отказ составляет для СБИС семейств FLEX приблизительно 5000 лет, а для семейств MAX — 3000 лет, причем для последних почти половина отказов связана с потерей заряда в элементах памяти, в конфигурирующих электрически стираемых перепрограммируемых ПЗУ (ЭСППЗУ, EEPROM)-транзисторов.

2.2. Стековые 3D БИС

Увеличение плотности трехмерных ИС (3D ИС), которое становится возможным благодаря вертикальному размещению элементов, будет способствовать многократному сокращению затрат на производство ИС по сравнению с традиционными 2 D ИС при той же технологии производства. 3D ИС могут масштабироваться с той же скоростью, какую предусматривает закон Мура, тем самым давая потребителям возможность со временем использовать все преимущества трехмерной технологии (рис.2.11).

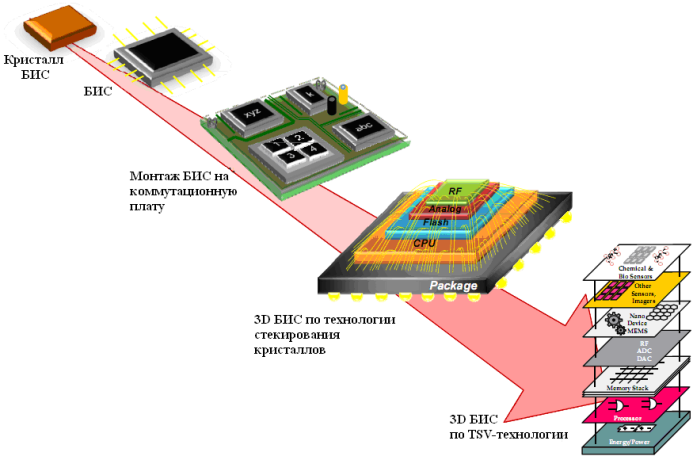


Рис.2.11. Вектор эволюционного развития БИС

Еще одна выгода от использования трехмерных ИС по субмикронным технологиям - уменьшение общей длины межсоединений на кристалле, что положительно сказывается на задержках распространения сигналов. Рис.2.12 показывает, что при переходе на новые технологические поколения задержки сигналов в межсоединениях 2D БИС неуклонно увеличиваются, а задержки в вентилях (логических элементах)

уменьшаются. Задержки в вентилях для субмикронных БИС лежат в диапазоне единиц пикосекунд.

Первые 3D ИС собирались по идеологии PoP (Package on Package – корпус на корпусе или укладка в трехразмерную сборку корпусированных ИС с использованием межсоединений и коммутационных слоев, размещенных на самих корпусах). Позднее операция монтажа кристаллов друг на друга с последующей упаковкой в единый корпус («stacked die», 3D-SIC, 3D stacked IC, штабелированные/стекированные трехмерные ИС) и последующего межсоединения методом разварки проволочных выводов получила более широкое распространение (рис.2.13 – рис.2.15).

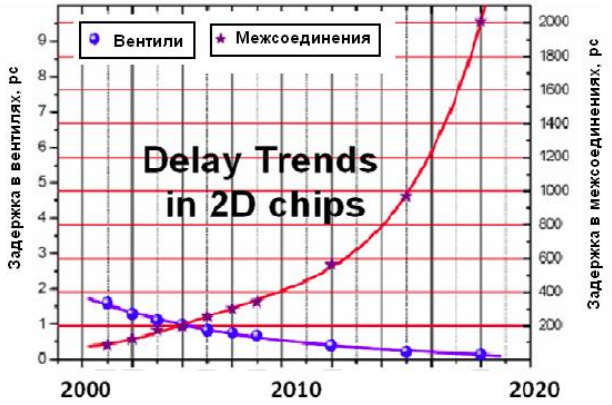


Рис.2.12. Задержки в вентилях и межсоединениях 2D БИС

МСМ ИС (многокристальный модуль на гибком полиимидном основании, МКМ 3D) появились в 1980 г. и характеризовались длинными межсоединениями между кристаллами, такими как цифровая логика и память. Наибольшая длина межсоединений достигала 10 мм. Использование стековых структур позволило уменьшить длину межсоединений до 70 мкм.

Так, на базе кристаллов 1537РУ30 (память 256 Кбит – разработка ОАО "НИИМЭ и завод "Микрон") спроектированы и изготовлены многокристальные модули памяти 1 Мбит в 3D-исполнении. На основе кристаллов 1645РУ1У (память 1 Мбит – разработка ЗАО "НПЦ Миландр") спроектированы и изготовлены МКМ памяти 4 Мбит. При проектировании использовались пакеты DxDesigner, Expedition PCB и Hyper Lynx САПР Mentor Graphics.

Рис.2.13. Стековые (stacked die) 3D БИС по технологии Non TSV с разваркой выводов и последующей упаковкой в единый BGA-корпус

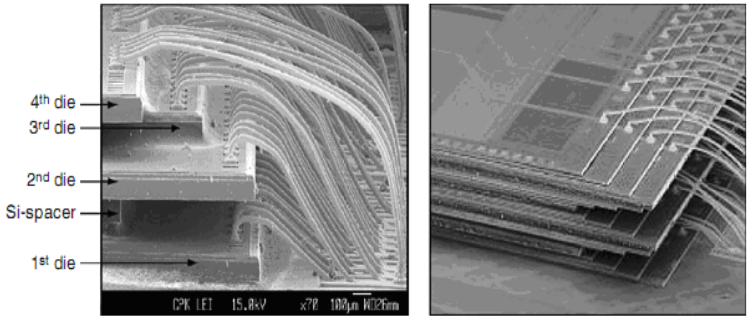
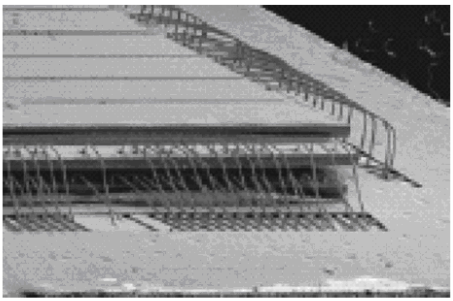


Рис.2.14. Монтаж кристаллов друг на друга и соединения их с помощью разварки выводов: а) 4 кристалла (технология фирмы ChipPAC); б) 20 кристаллов с толщиной 25 мкм по технологии фирмы Hynix



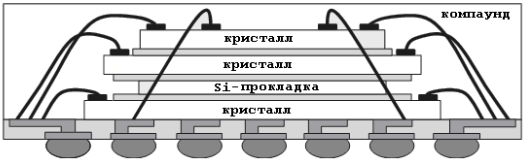


Рис.2.15. Стековые 3D БИС фирмы Intel (логика + два кристалла памяти) по технологии Non TSV с разваркой выводов. Логика – верхний кристалл

На сайте EE Times опубликован список 10 перспективных технологий электроники, которые, по мнению ресурса, получат свое развитие в 2010 году. Девятую позицию занимает TSV-технология. Ожидается, что данная технология приведет к созданию реальных 3D БИС.

В настоящее время, использование технологии TSV (through silicon vias – сквозные кремниевые межсоединения или переходные отверстия в кремнии), позволило убрать операцию разварки из технологической цепочки, обеспечив максимально возможный на сегодняшний день уровень интеграции ИС. TSV-технология включает процессы формирования соединений, осаждения, заполнения, удаления металла с поверхности, утонения пластин, соединения/стекирования, инспектирования, тестирования.

Зарубежным компаниям удалось достичь определенных успехов в области создания новых средств САПР для проектирования 3D ИС. Компании Atrenta, AutoESL, Qualcomm разработали верхнеуровневую систему

проектирования 3D ИС, в частности 3D-синтез высокого уровня, раннее разделение, разработку общей топологии кристалла и многодоменный анализ. Компания Qualcomm для своих 3D ИС разработала технологию PathFinding, которая способствует поэтапной оптимизации посадочной площади кристалла, тепловых характеристик и других функций.

Межуниверситетский центр по микроэлектронике IMEC (Interuniversity Microelectronics Centre), крупнейший европейский независимый институт по исследованиям в области наноэлектроники, использует в своих опытных партиях 3D ИС технологию штабелированных трехмерных ИС и TSV-технологию. В настоящее время выпускаются кристаллы, выполненные на 200-мм пластинах в рамках базового 0.13-мкм КМОП-техпроцесса с добавлением техпроцесса Cu-TSV. Для штабелирования верхний кристалл подвергают утонению до 25 мкм и присоединяют к кристаллу-основанию с помощью термокомпрессии Cu-Cu. Центр IMEC в настоящее время расширяет данный техпроцесс до операции присоединения кристалла непосредственно к полупроводниковой пластине и находится в процессе переноса данной технологии на платформу 300 мм. Для тестирования целостности и производительности 3D ИС используют кольцевые генераторы различной конфигурации, размещенные в двух кристаллах и связанные посредством Cu TSV.

Для проектирования 3D БИС может быть использована САПР Synopsys. StarRC™ - инструмент платформы Galaxy для экстракции значений паразитных сопротивлений, индуктивностей и ёмкостей (RLC) из топологических представлений ИС в формате GDSII, позволяет, учитывает все известные физические эффекты в полупроводниках для технологий вплоть до 28 нм и осуществлять 3D моделирование внутренних соединений из меди, локальных внутренних соединений, диэлектриков с низкой диэлектрической проницаемостью, кремния на изоляторе (SOI) и вариаций

параметров на кристалле. Ведущие кремниевые фабрики (TSMC, UMC, Chartered и др.), предоставляют технологические файлы процессов для работы с инструментом StarRC.

Продукты, которые необходимо использовать для проектирования 3D цифровых БИС: Encounter Digital Implementation System (EDI System), Encounter Power system (EPS), Encounter Timing System (ETS), First Encounter. EDI System позволяет достичь оптимальных результатов по быстродействию при проектировании с уровня RTL до уровня размещенных вентилей. Алгоритмы логического синтеза и оптимизации основаны на технологии RTL Compiler. Обеспечивается полная поддержка синтеза, планировки кристалла, функций размещения макроблоков и стандартных ячеек, планирование и разводка сетки питания, экстракция паразитных параметров, функции анализа параметров быстродействия и анализа сетки питания и т.д. ETS - является средством временного статического анализа, а также анализом взаимодействия проводников. ETS-XL может точно учитывать влияние перекрестных наводок и просадок питания на временные задержки логики и ее функциональность. В продукте First Encounter САПР Cadence реализована концепция виртуального прототипа.

Трехмерный метод производства имеет два очевидных преимущества. Во-первых, это существенное снижение цен при фиксированном числе транзисторов на кристалле и, во-вторых, увеличение числа транзисторов со скоростью, не меньшей, чем предусматривает закон Мура в расчете на количество устройств, приходящихся на единицу площади.

Максимально возможный на сегодняшний день уровень интеграции ИС обеспечивает использование технологии TSV, которая позволяет убрать операцию разварки из технологической цепочки, обеспечивает более высокую плотность монтажа, большую функциональность, лучшие

технические характеристики (минимальная длина соединений, межсоединения не ограничивают скорость распространения сигнала), более низкое энергопотребление, меньшую стоимость. Однако использование данной технологии требует решение ряда проблем, таких как снижение токов утечек ИС динамической памяти, снижение рассеиваемой мощности.

TSV-технология позволяет значительно увеличить количество линий ввода/вывода, что радикальным образом приводит к повышению скорости трансляции данных и уменьшить энергопотребление, а также вызвать появление принципиально новых видов высокоэффективных устройств. Для проектирования 3D БИС с TSV необходимо использовать новые инструменты САПР для нанометровых проектов.

2.3. ПЛИС типа ППВМ: от 2D к 3D

Цель данного раздела показать эволюционные изменения, происходящие в трассировочных ресурсах ПЛИС при переходе от 2D к 3D и какой выигрыш от этого может быть получен. Схемотехнические решения в трассировочных ресурсах и алгоритмы программирования электрических соединений являются важнейшими ноу-хау разработчиков индустриальных ПЛИС.

“Кросс-бары” (коммутаторы, маршрутизаторы) 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 (рис.2.16 и рис.2.17). Мультиплексор с полной коммутацией является наиболее гибким, однако, его недостатком является большое количество ключей

коммутации с соответствующим числом конфигурационной памяти.

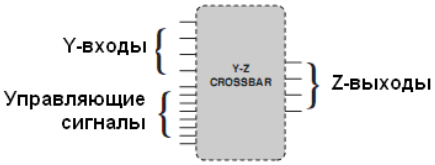


Рис.2.16. Мультиплексор с полной коммутацией (“кросс-бар”)

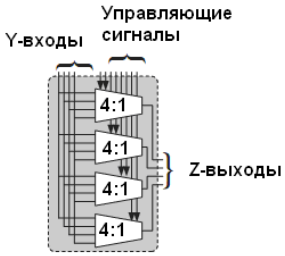


Рис.2.17. Определение мультиплексора с полной коммутацией 4 входа х 4 выхода (8 управляющих сигналов) на мультиплексорах 4 в 1

2D индустриальные ПЛИС. Под индустриальными ПЛИС будем подразумевать различные архитектуры ПЛИС выпускаемые зарубежными фирмами (например, хорошо известные в России - Altera, Xilinx и малоизвестные, такие как SiliconBlue Technologies Corporation (www.SiliconBlueTech.com) и Tabula (www.tabula.com)) для массовых коммерческих применений, а под академическими, ПЛИС разрабатываемые в ведущих учебных центрах (например, Университет Торонто и Университет Британской Колумбии в Канаде, в США - Университет штата Нью-Йорк в Стоуни-Брук, Массачусетский технологический институт, Калифорнийский технологический институт).

ПЛИС типа ППВМ фирмы Altera серии FLEX10K и FLEX6K по зарубежной классификации можно рассматривать как полуиерархические, а серии APEX иерархической архитектуры с многоуровневой структурой трассировочных каналов, в которых верхний уровень иерархии представляет

непрерывные (несегментированные) длинные линии (FastTrack – длинное непрерывное межсоединение в трассировочном канале), простирающиеся через весь кристалл, половину и четверть кристалла по вертикальным и горизонтальным направлениям. ПЛИС с FastTrack иногда называют “строковые” ПЛИС. Большинство же других коммерческих архитектур ПЛИС типа ППВМ по технологии СОЗУ (например, ПЛИС фирмы Xilinx серии Virtex) имеет одноуровневую структуру, когда кластеры из логических блоков окружены с четырех сторон межсоединениями горизонтальных и вертикальных трассировочных каналов, равномерно распределенных по всей площади кристалла.

На рис.2.18 показан фрагмент индустриальной ПЛИС типа ППВМ серии FLEX 8K компании Altera с многоуровневой структурой. Подключение кластера из 8 логических элементов (ЛЭ) с 4-х входовыми LUT-таблицами с числом входов 24 к горизонтальному трассировочному каналу из 168 межсоединений осуществляется с помощью соединительного блока (мультиплексора частичной коммутации 168 входов х 24 выхода (разреженный коммутатор на 1/12). Каждая строка (FastTrack) из канала может быть скоммутирована дважды на входы кластера. Внутри кластера коммутация межсоединений осуществляется с помощью мультиплексоров полной коммутации. ПЛИС серии FLEX 8K имеют несегментируемую трассировочную структуру.

На рис.2.19 показано подключение кластера из базовых логических элементов (BLE) к горизонтальным и вертикальным трассировочным каналам ($W_X = W_Y$) в ПЛИС с одноуровневой структурой межсоединений (академическая ПЛИС с симетричной структурой, известная под названием как ПЛИС с “островковой структурой” (Island-style)) с помощью мультиплексоров частичной коммутации. Подключение входов/выходов кластера к трассировочным каналам осуществляется с четырех сторон. Трассировочные

каналы сегментируются маршрутизаторами (матрица переключателей) типа Disjoint с коэффициентом разветвления $F_S = 3$. в ПЛИС Stratix III используется трехсторонняя, а в ПЛИС Virtex двухсторонняя коммутация кластера к трассировочным каналам.

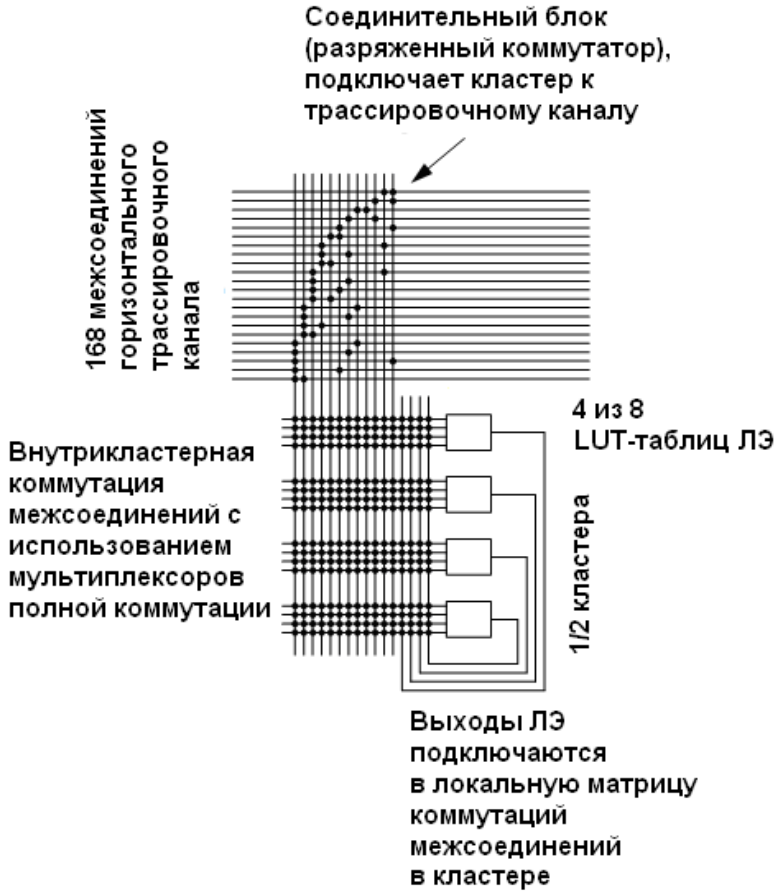


Рис.2.18. Частичная и полная коммутация в индустриальной ПЛИС типа ППВМ серии FLEX 8K компании Altera

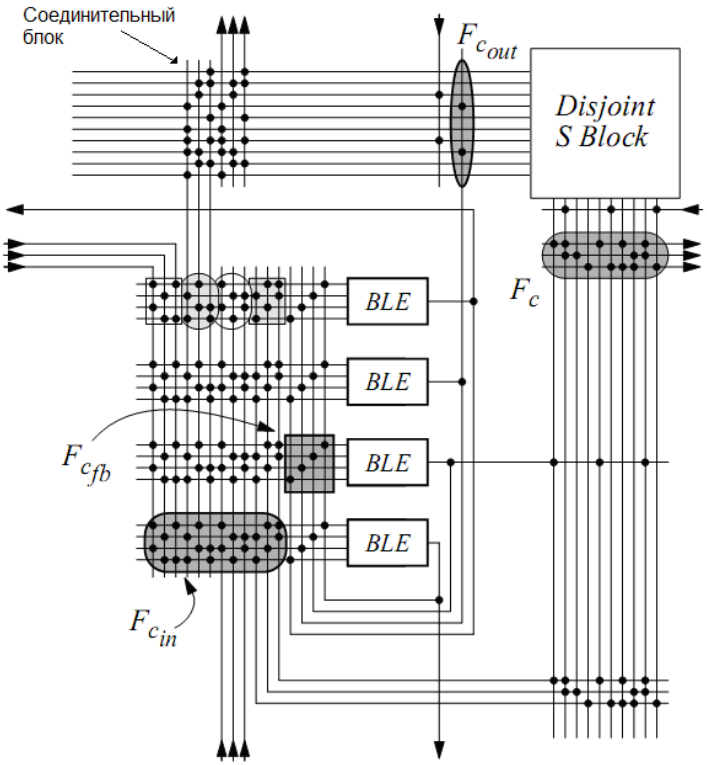


Рис.2.19. Подключение кластера к трассировочным каналам в ПЛИС с одноуровневой структурой межсоединений (академическая ПЛИС)

В ранних сериях ПЛИС типа ППВМ преимущество отдавалось использованию сегментов межсоединений короткой длины, а длинные линии набирались из коротких межсоединений разделенных между собой электронными ключами (проходные транзисторы или буферы с третьим состоянием), что приводило к возрастанию задержек распространения сигналов в длинных линиях, за счет внесения паразитных сопротивлений и емкостей проходными транзисторами.

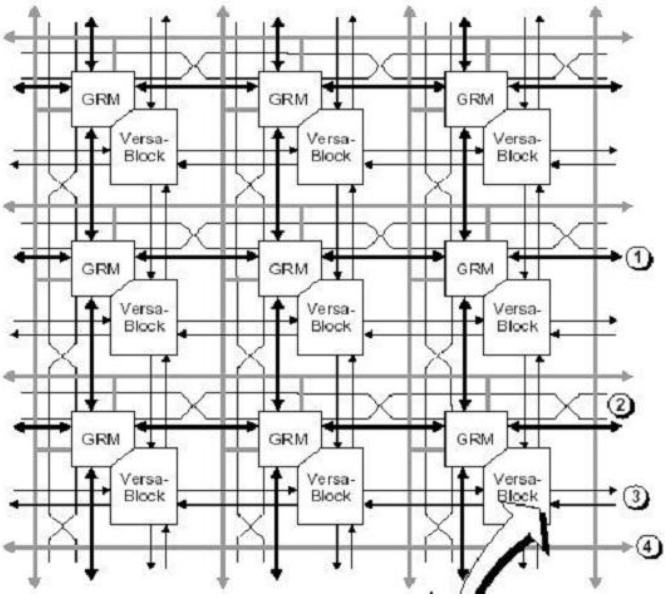
В современных индустриальных ПЛИС, например в ПЛИС XC5200 фирмы Xilinx используется 6 уровней

межсоединий в трассировочных ресурсах (рис.2.20): 1 - короткие линии (X1, длина сегментации межсоединений в одну глобальную трассировочную матрицу (GRM)); 2 – линии двойной длины (X2, длина сегментации через одну GRM); 3 – прямые соединения (direct connects) между кластерами (VersaBlock) из логических блоков без захода в GRM; 4 – длинные/глобальные линии, простирающиеся через весь кристалл, по ним передаются глобальные сигналы сброса/установки; 5 – локальная матрица межсоединений (LIM, входные коммутаторы для подключения кластера VersaBlock); 6 – вспомогательные межсоединия для логических ячеек (LC) в кластере (TS, коммутаторы на входы LC). ПЛИС Lattice ORCA IV серии содержит 23 % X1, 70 % X6 и 7 % длинных линий в половину кристалла. ПЛИС Xilinx серии Virtex4 содержит 22 % X1, 66 % X6 и 13 % X24 длинных линий в горизонтальных и вертикальных направлениях.

Прямые соединения имеют наименьшую задержку распространения сигнала и эффективны для реализации быстрых арифметических модулей, обладающих большим числом локальных соединений, критичных по скорости. Длинные линии предназначены для больших разветвлений по выходу и эффективны для реализации шин в проектах пользователя.

В настоящее время современные индустриальные ПЛИС типа ППВМ серии Stratix фирмы Altera и Virtex фирмы Xilinx имеют сегментируемую трассировочную структуру с использованием однонаправленных межсоединений (unidirectional). В архитектуре ПЛИС семейства Stratix фирмы Altera соединения между кластерами, TriMatrix памятью, DSP-блоками, и элементами ввода/вывода (ЭВВ) осуществляется 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 её месторасположения на кристалле ПЛИС, что обеспечивается использованием однонаправленных межсоединений и мультиплексорных структур типа single-driver. Каждая связь в пределах канала управляется единственным источником сигнала и впоследствии может быть выбрана мультиплексором.



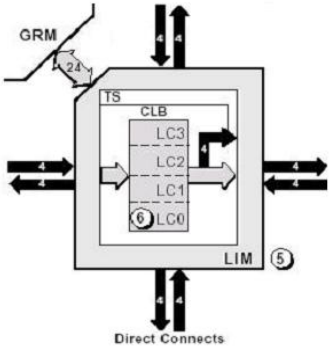


Рис.2.20. Уровни межсоединий в трассировочных ресурсах ПЛИС XC5200 фирмы Xilinx

Многоканальные соединения представляют собой непрерывный, оптимизированный набор шин различной длины и скорости, используемые для меж- и внутри- блоковой разводки, распространяются на фиксируемое расстояние. Например, горизонтальный канал R20 является самым быстрым каналом, распространяющимся через 20 кластеров, а R4 обладает наивысшей трассировочной способностью. Так же существуют и вертикальные трассировочные каналы C4 и C12, цепи ускоренных цепочечных переносов, цепи для обеспечения обще-арифметических операций, регистерные цепи. Указанные цепи, как правило, объединяют в вертикальные трассировочные каналы.

Существуют цепи глобальной синхронизации ПЛИС, которые работают совместно с блоками аналоговой фазовой автоподстройки частоты (PLL, Altera). Так в передовых зарубежных ПЛИС может находиться до 12 таких устройств, присутствуют так же блоки цифровой автоподстройки по задержке (DLL, Xilinx). PLL и DLL должны обеспечивать синфазность синхроимпульсов во всех точках кристалла, которые необходимы при работе ПЛИС на частотах свыше 200 Мгц. Цепи глобальной синхронизации ПЛИС обычно включают в трассировочные каналы.

В индустриальных ПЛИС используется технология wafer-level-processed stacked package с интеграцией на уровне коммутационной платы. Фирма Xilinx совместно с TSMC разработала новейшую серию ПЛИС FPGA (функциональная емкость 2 млн. эквивалентных вентилей) Virtex-7 с использованием интеграции на уровне коммутационной Si-пластины (рис.2.21). Кристаллы Virtex-7 реализуются по технологии 28 нм, а межсоединения кристаллов выполняются в коммутационной Si-пластине с использованием хорошо отработанного 65 нм технологического процесса с 4 слоями металлизации. Что обеспечивает десятки тысяч

межсоединений, высокую пропускную способность и малое время задержек распространения сигналов.

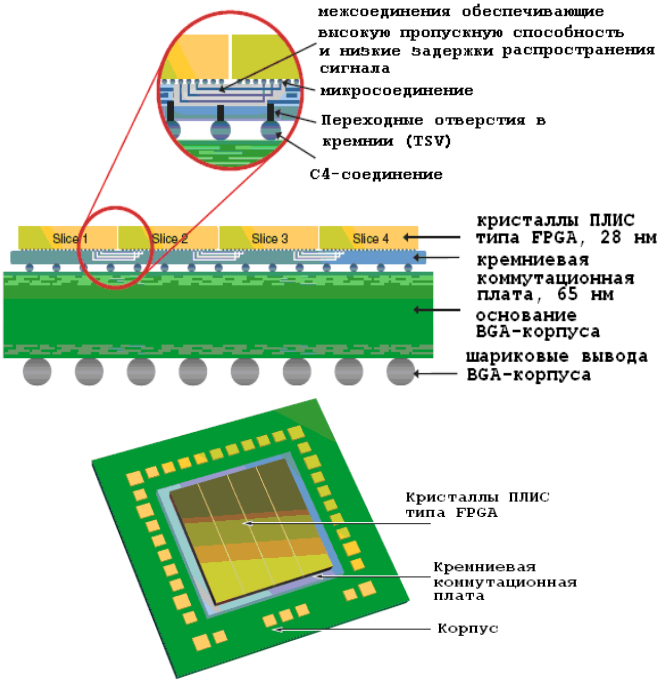


Рис.2.21. 3D ПЛИС FPGA серии Virtex -7 фирмы Xilinx с использованием комбинированных технологий: монтаж методом перевернутого кристалла в BGA-корпус; интеграция кристаллов на уровне кремниевой пластины по технологии 65 нм; TSV-технология; 28 нм КМОП-технологический процесс

3D академические ПЛИС. 3D интеграция позволяет снизить нагрузку на межсоединения, снизить мощность потребления и уменьшить задержки в межсоединениях. На рис.2.22 показана концепция создания 3D ПЛИС по технологии стекирования кристаллов. За основу взята одноуровневая структура ПЛИС типа ППВМ.

Конфигурационная память (ячейки СОЗУ) располагается в кристалле Memory Layer, трассировочные ресурсы (маршрутизаторы SB и соединительные блоки CB) в кристалле Switch Layer, матрица логических блоков в кристалле CMOS Layer.

В 3D ПЛИС необходима интеграция функциональных блоков, таких как маршрутизаторы и соединительные блоки из 2D ПЛИС. Предлагается модель коммутации трассировочных ресурсов под названием “фабрика маршрутизации”, которая предполагает наличие матрицы из блоков маршрутизации (RB), при этом каждый из блоков маршрутизации связан со своим логическим блоком (рис.2.23). Конфигурационная память располагается в кристаллах Memory Layer 2 и Memory Layer 1, матрица блоков маршрутизации (RB) в кристалле Switch Layer, матрица логических блоков (LB) в кристалле CMOS Layer.

Матрица из блоков маршрутизации покрыта горизонтальными и вертикальными трассировочными каналами, межсоединения в которых сегментируются через 1 (X1) или 2 коммутационных блока (X2) (рис.2.24 и рис.2.25).

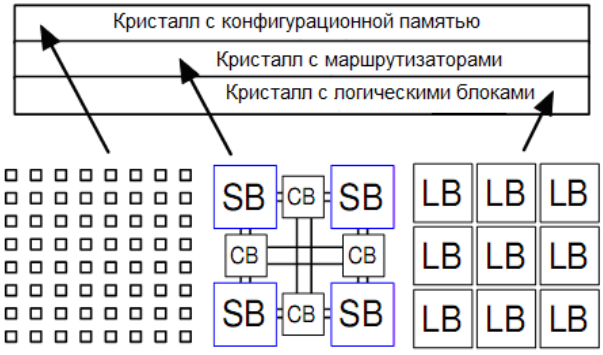


Рис.2.22. Концепция создания 3D ПЛИС по технологии стекирования кристаллов

Рис.2.23. Концепция “трассировочная фабрика” для создания 3D ПЛИС по технологии стекирования кристаллов

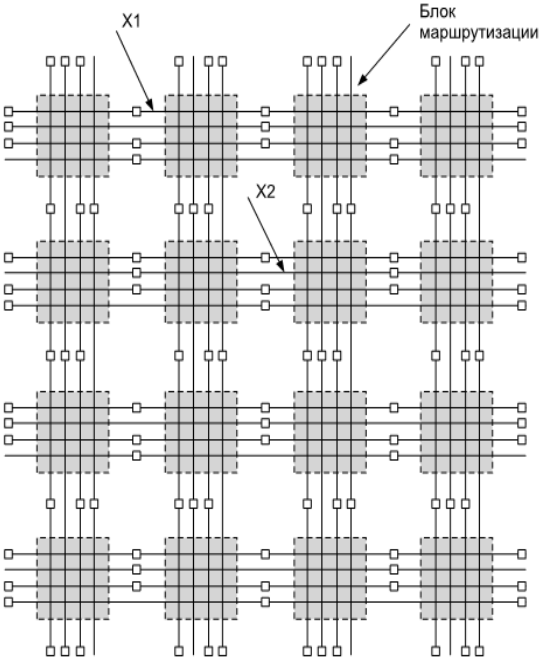


Рис.2.24. Матрица из блоков маршрутизации (RB) и горизонтальные и вертикальные трассировочны е каналы в модели “трассировоч-ная фабрика”

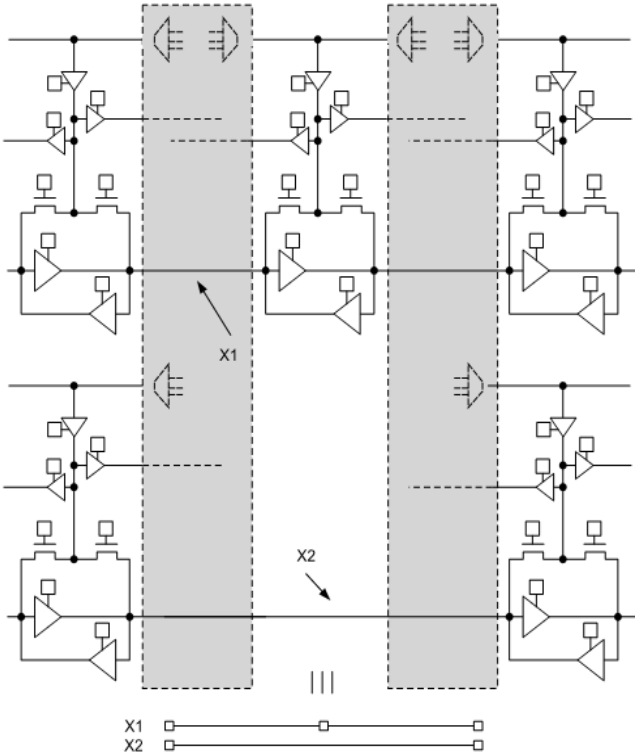


Рис.2.25. Подключение межсоединений с длиной сегментации в один (Single Interconnect) или два (Double Interconnect) RB к матрице RB

На рис.2.26 показана концепция создания 3D ПЛИС с использованием 3D маршрутизаторов. LUT-таблицы, входящие в состав логических блоков, подсоединяются к горизонтальным и вертикальным трассировочным каналам с четырех сторон. Межкристальные соединения осуществляются с помощью 3D маршрутизаторов. Коэффициент объединения по входу/выходу для соединительных блоков $F_C = W$ и коэффициент разветвления входящих межсоединений для маршрутизаторов $F_S = 5$. Число межкристальных соединений составляет 15-20 % от общего числа межсоединений в 3D ПЛИС логической емкостью 20K при числе кристаллов от 2-х до 4-х.

На рис.2.27 показано дальнейшее развитие концепции создания 3D ПЛИС с использованием 3D “плитки” (3D маршрутизатор с локальными трассировочными ресурсами) разработанной в Массачусетском технологическом институте (MIT) США. “Плитка” представляет собой конфигурируемый логический блок (CLB) как у ПЛИС серии Virtex II фирмы Xilinx, состоит из четырех секций (Slice), объединенных локальной матрицей коммутации, которая непосредственно связана с 3D маршрутизатором с возможностью межкристальной коммутации (Inter-strata via).

Однако, 3D маршрутизаторам присущи недостатки: требуется большее число ключей на n-МОП транзисторах и конфигурационных ячеек памяти СОЗУ. Преимущество – позволяют существенно снизить ширину трассировочного канала, по сравнению с 2D ПЛИС, что приводит к увеличению логической емкости ПЛИС; уменьшить задержки распространения сигналов, особенно в длинных линиях, и снизить мощность потребления.

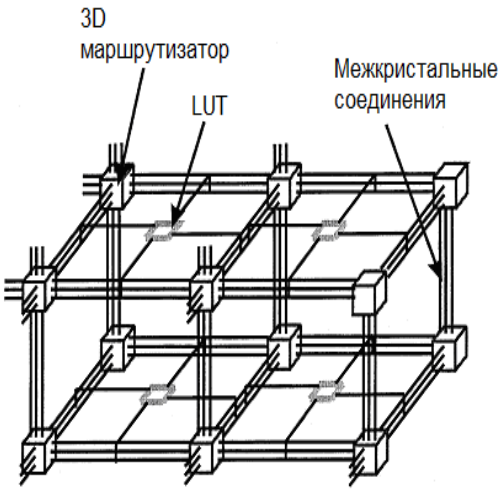


Рис.2.26. Концепция создания 3D ПЛИС с использованием технологии стекирования кристаллов и 3D маршрутизаторов

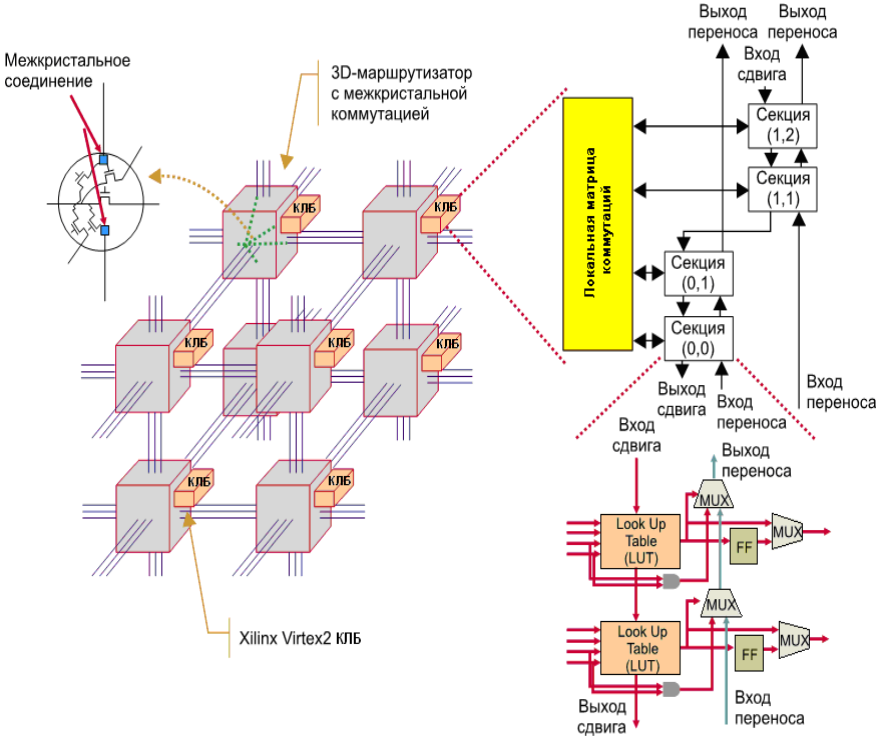


Рис.2.27. Концепция создания 3D ПЛИС с использованием технологии стекирования кристаллов (MIT-архитектура)

На рис.2.28 показано соединение трех кристаллов в стековую структуру с использованием оригинальной технологии FaStack фирмы Tezzaron (www.tezzaron.com). Такое соединение можно рассматривать как “истинную” 3D интеграцию. Используется стандартный КМОП технологический процесс с пятью слоями медной металлизации для реализации внутренних межсоединений и изоляцией активных структур мелкими канавками.

Два нижних кристалла соединяются “лицом к лицу” посредством внутренних контактных площадок, причем на обратной стороне второго кристалла, так же сформированы

внутренние контактные площадки. Третий кристалл (верхний) присоединяется к обратной стороне второго кристалла “лицом вниз”. На обратной стороне третьего кристалла формируется контактная площадка из алюминия для подключения периферийных устройств, например, микромеханического датчика. Рассмотрим кристалл 2. Например, сквозные межслойные контакты (Super-Contact, выполняют роль межкристальных соединений) соединяют первый уровень межсоединений (топологический слой металл М1) с внутренними контактными площадками, расположенными с обратной стороны пластины.

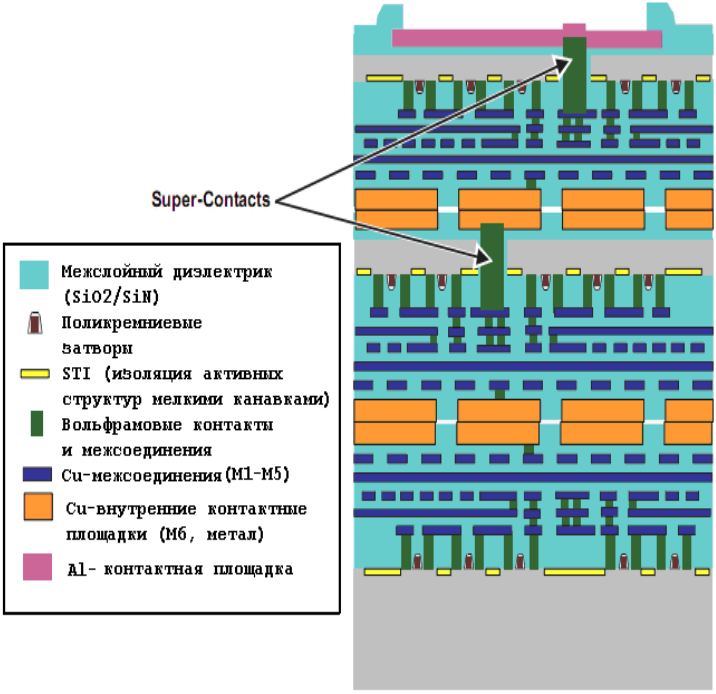


Рис.2.28. 3D ПЛИС типа ППВМ с использованием технологии FaStack фирмы Tezzaron

Ниже перечислены недостатки 3D ПЛИС, присущие и 3D БИС:

- переходные отверстия через кремний (through-silicon vias – TSV, ранее употреблялся термин Inter-strata via, межкристальное соединение) оказывают существенное влияние на систему питания и температурный режим многослойной структуры БИС. Чрезвычайно трудно рассчитать, в каком месте переходные отверстия должны быть сформированы и как они будут влиять на систему питания и температурный режим;

- неясны ответы на вопросы, связанные с методами соединения кристаллов, количеством кристаллов в структуре, тепловыми условиями в многослойной структуре, методами тестирования и т.д. Развитие этого направления требует совмещения технологических решений на уровне кристалла, корпуса и системы в целом чтобы минимизировать влияние помех;

- возможности повышения уровня интеграции с применением 3D-технологии ограничены оптимальным количеством стекируемых кристаллов, поскольку при увеличении количества стекируемых кристаллов, количество межкристальных соединений и сложность перечисленных проблем будет резко возрастать.

3D академические ПЛИС с использованием нанотехнологий. 3D NRAM ПЛИС. Рассмотрим один из вариантов реализации ПЛИС типа ППВМ с использованием памяти на углеродных нанотрубках и КМОП-технологического процесса, который можно отнести к “псевдо” 3D ПЛИС. Развивающаяся американская компания Nantero (www.nantero.com) активно занимается созданием новой технологии создания энергонезависимой оперативной памяти 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.

Запоминающее устройство состоит из двух кремниевых подложек, на которых особым образом размещены массивы нанотрубок. Используется два таких свойства, как эластичность (гибкость) нанотрубок и притягивание атомов углерода друг к другу под воздействием сил Ван-дер-Ваальса. Нанотрубки закрепляются на кремниевой подложке, а под ними на расстоянии примерно 120 нм располагается углеродный субстрат. Малое расстояние между соседними подложками вместе с ничтожными размерами нанотрубок позволяют достичь скоростей записи-чтения порядка половины наносекунды.

В предложенной архитектуре кристаллов слой нанотрубок наносится на подложку. Затем методом обычной литографии на нем “вычерчивают” электрические контакты, соединенные друг с другом “толстыми” лентами из нанотрубок. При этом пространственная ориентация нанотрубок и степень их идентичности не имеют значения. Главное, чтобы ленты проявляли нужные механические свойства. Электрический заряд небольшой силы, возникающий на нижней подложке, притягивает к последней группу нанотрубок, расположенных над ней. Далее притянутые нанотрубки удерживаются в таком состоянии под действием сил Ван-дер-Ваальса до появления следующего электрического заряда. Благодаря такому устройству свисающие нанотрубки могут играть роль битов памяти: “поднятое” состояние – логический 0, “опущенное” – логическая 1. Так как в каждом отдельном переходе между указанными состояниями участвует несколько десятков нанотрубок, создается избыточность, предохраняющая систему от случайных потерь информации. В “замкнутом” и “разомкнутом” состояниях система из нанотрубок имеет

различное электрическое сопротивление, за счет чего возможно считывание информации.

В настоящее время специалистами Nantero уже создан работающий прототип массива NRAM. Компания Nantero сообщила об изготовлении и удачном тестировании памяти на основе углеродных нанотрубок, произведенной по 22-нм технологическому процессу. Представители компании заявляют, что 22-нм NRAM-память может быть изготовлена на оборудовании, предназначенном для производства микропроцессоров по КМОП-технологии. Технологический процесс Nantero разрабатывала совместно с корпорацией LSI Logic (www.lsilogic.com), что и обеспечило совместимость с уже существующим оборудованием, требующего незначительной модификации.

Традиционные реконфигурируемые архитектуры, к которым относятся ПЛИС типа ППВМ, поддерживают только частичное динамическое реконфигурирование или крупно-зернистую мультиконтекстную реконфигурацию и не позволяют использовать мелко-зернистое логическое временное сворачивание (упаковку логики). Предположим, что необходимо реализовать булеву функцию на LUT-таблицах (рис.2.29). Для традиционных реконфигурируемых архитектур потребуется n-LUT-таблиц, а, следовательно, n-ЛЭ. Для временного сворачивания потребуется лишь 1 ЛЭ с памятью NRAM для временного хранения значений LUT-таблиц: LUT-1 в первом цикле, LUT-2 во втором цикле и т.д.

На рис.2.30 показан граф связанности 10 LUT-таблиц для реализации заданной булевой функции. Логическое сворачивание может быть произведено на различных уровнях разбиения схемы. 4 ЛЭ входящих в макроблок, достаточно чтобы построить граф связанности 10 LUT-таблиц, реализующий некую произвольную булеву функцию, за несколько временных циклов (рис.2.29), с другой стороны рис.2.30 показывает второй уровень сворачивания

рассматриваемого графа, но уже с использованием двух макроблоков. В первом случае 4 уровня вложенности требуют большее время вычислений, во втором 2 уровня требуют меньшее время, но большее число ЛЭ.

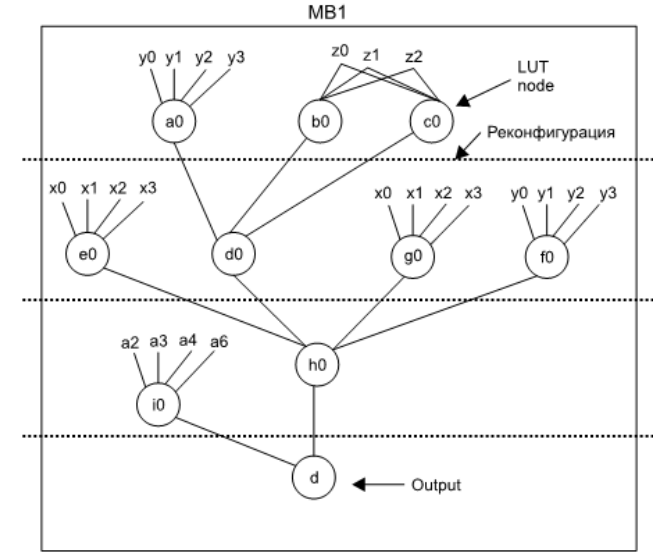


Рис.2.29. Граф связанности LUT-таблиц для реализации булевой функции в одном макроблоке. Первый уровень упаковки логики

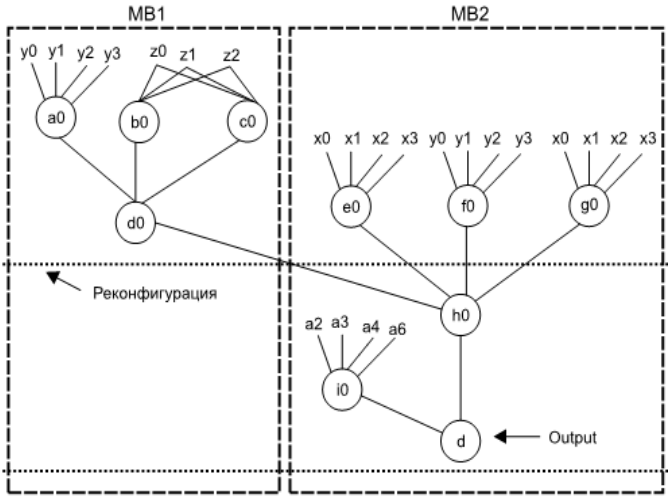


Рис.2.30. Второй уровень упаковки логики

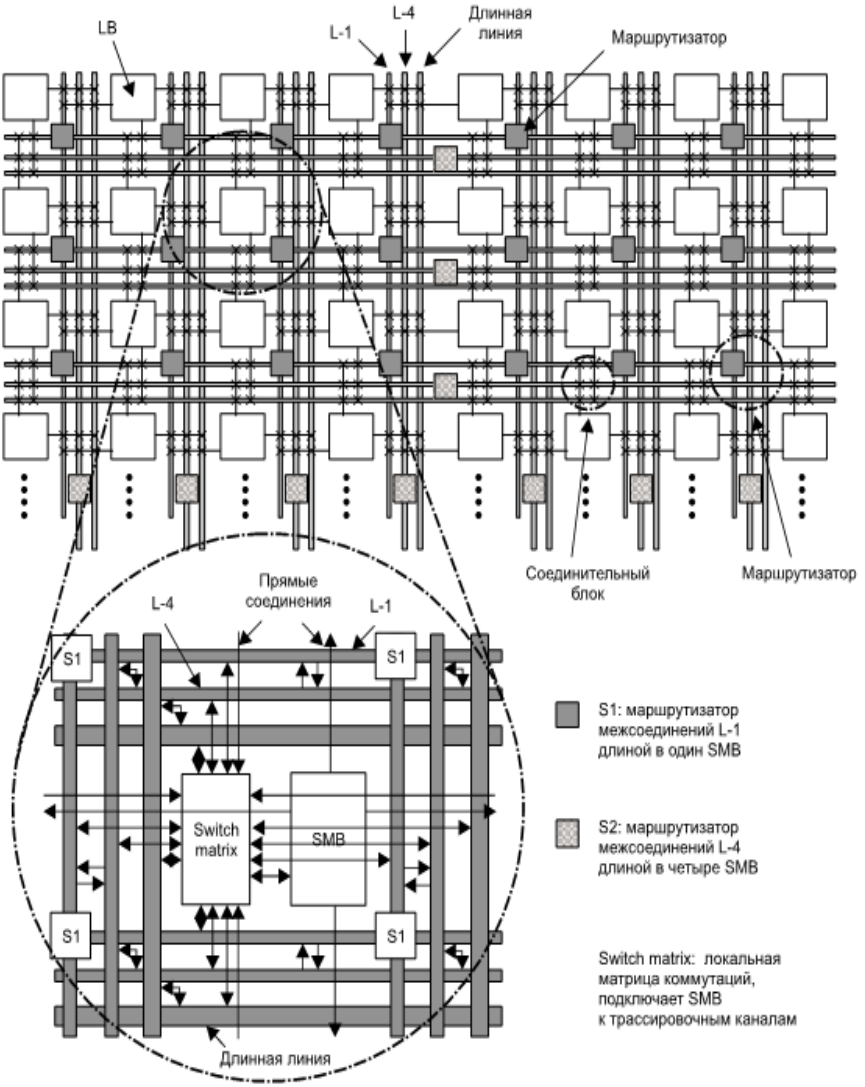
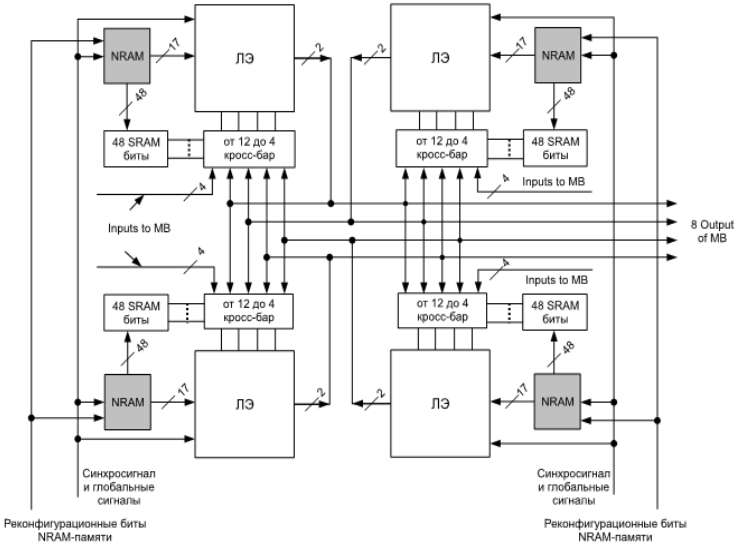


Рис.2.31. S1-маршрутизатор межсоединений L-1 с длиной сегментации в 1 ЛЭ; S2- маршрутизатор межсоединений L-4 с длиной сегментации в 4 ЛЭ; Switch matrix – локальная матрица коммутации межсоединений, подключает входы/выходы Супермакроблока к трассировочным каналам

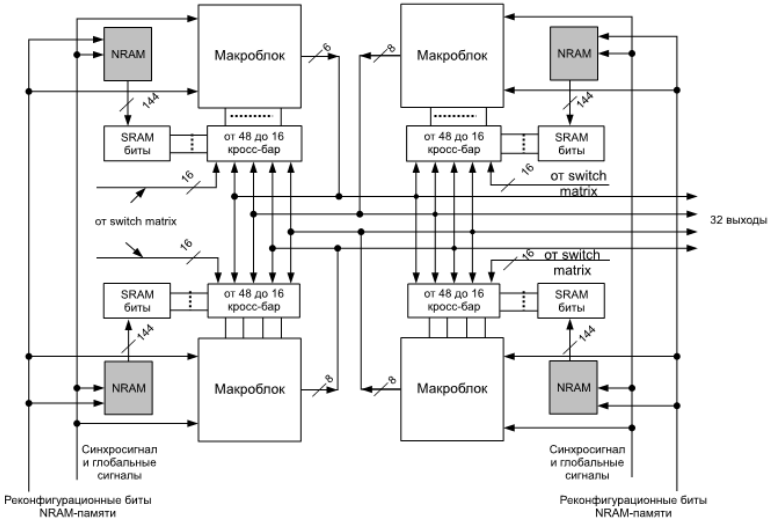
В основе ПЛИС NATURE (hybrid CMOS/NanoTUbe REconfigurable architecture, NATURE) типа ППВМ лежит одноуровневая структура межсоединений (рис.2.31). Для достижения компромисса задержка распространения сигнала/площадь кристалла используются L-1, L-4 и длинные линии, пересекающие весь кристалл. Предполагается, что используется 100 нм КМОП-технологический процесс, поэтому длина нанотрубок будет составлять 100 нм. Время реконфигурации одной LUT-таблицы составляет 160 пс, величина, значительно меньше времени задержки распространения сигнала в трассировочных ресурсах обычных ПЛИС.

Логический блок содержит локальную матрицу межсоединений и супермакроблок (рис.2.32). Соседние супермакроблоки связаны между собой прямыми межсоединениями (Direct link). Используется двухуровневая архитектура кластера (Супермакроблока). Кластер состоит из четырех макроблоков. Каждый макроблок состоит из 4 логических элементов (ЛЭ, LE). ЛЭ: 4-х входовая LUT-таблица и программируемый триггер. Для конфигурации ЛЭ требуется 17 бит конфигурационной NRAM-памяти, которые считываются последовательно. NRAM-память входящая в состав ЛЭ приблизительно занимает 21 % от площади макроблока при использовании 100 нм КМОП-технологии. До начала работы ПЛИС, конфигурационные данные в NRAM перезагружаются из внешней памяти.

Программируемый коммутатор 12x4 (соединительный блок) подключает собственные входы/выходы и выходы (комбинационный и регистерный) от трех соседних ЛЭ к своим собственным входам. Для программирования коммутатора требуются 48 бит памяти СОЗУ, которые перезагружаются из энергонезависимой NRAM-памяти, которая играет роль реконфигурируемой памяти. Емкость блока NRAM-памяти макроблока 65 бит.



а)



б)

Рис.2.32. а) Макроблок (MB) и б) Супермакроблок (SMB) ПЛИС с NRAM

Кластер состоит из 16 ЛЭ. Входами каждого из макроблоков являются 16 сигналов из локальной матрицы коммутаций и 32 выхода четырех макроблоков в кластере, которые объединяются во внутрикластерную локальную шину и подключаются к локальной матрице. В кластере используются коммутаторы 48x16.

В качестве альтернативы можно рассмотреть серию Tabula (www.tabula.com) компании Abax - псевдо-3D ПЛИС, работающие как 8 стекированных кристаллов. По оценкам разработчиков Tabula, её кристаллы более чем в три раза меньше по плоскости, чем эквивалентные ПЛИС, что делает их в 5 раз дешевле при производстве, предоставляя при этом удвоенную плотность логики и почти в 4 раза более высокую производительность. Как и в случае с ПЛИС типа ППВМ, ПЛИС компании Tabula содержат множество идентичных базовых логических блоков, которые могут быть запрограммированы на выполнение любой нужной логической функции. Встроенная память ПЛИС хранит конфигурацию слоёв, на которые он переключается. Однако энергопотребление таких ПЛИС будет относительно высоким.

3D мемристорные (CMOL FPGA, mrFPGA) ПЛИС. Рассмотрим один из вариантов реализации ПЛИС типа ППВМ с использованием энергонезависимой резистивной памяти (http://ru.wikipedia.org/wiki/Мемристор). Мемристор (англ. memristor, от memory - “память”, и resistor - “электрическое сопротивление”) - пассивный элемент в микроэлектронике, способный изменять свое сопротивление. Может быть описан как двухполюсник с нелинейной вольт-амперной характеристикой, обладающий гистерезисом.

Теория мемристора была создана в 1971 году профессором Леоном Чуа. Устанавливает отношения между интегралами по времени силы тока, протекающего через элемент, и напряжения на нем. Долгое время мемристор считался теоретическим объектом, который нельзя построить.

Однако лабораторный образец мемристора был создан в 2008 году коллективом ученых во главе с Р.С. Уильямсом в исследовательской лаборатории фирмы Hewlett-Packard.

В отличие от теоретической модели, устройство не накапливает заряд, подобно конденсатору, и не поддерживает магнитный поток, как катушка индуктивности. Работа устройства обеспечивается за счет химических превращений в тонкой (5 нм) двухслойной пленке двуокиси титана. Один из слоев пленки слегка обеднен кислородом, и кислородные дырки мигрируют между слоями под действием приложенного к устройству электрического напряжения. Данную реализацию мемристора следует отнести к классу наноионных устройств. Сопротивление мемристора можно существенно (на три порядка) изменять, пропуская через него ток. Изменение сопротивления эквивалентно переключению между единичным и нулевым состоянием, что и наделяет новый элемент свойством памяти. Энергия затрачивается только в момент переключения.

Наблюдающееся в мемристоре явление гистерезиса позволяет использовать его в качестве ячейки памяти. Блоки СОЗУ могут быть более емкими и быстрыми чем флеш-память. Их умение “запоминать” заряд позволит отказаться от загрузки системы. В памяти компьютера отключенного от питания будет храниться его последнее состояние. Его можно будет включить и начать работу с того места, на котором остановился. Это же свойство позволит отказаться от некоторых компонентов современного ПК, что позволит сделать компьютеры меньше и дешевле. HP предполагает, что к 2012 году мемристоры начнут заменять собою флеш-память, в 2014-2016 - оперативную память и жесткие диски. За последние три года фирма HP накопила около 500 патентов по мемристорам, охватывающих резистивные ОЗУ, память с изменением фазы и другие виды двухвыводных приборов памяти.

На рис.2.33 показан фрагмент мемристорной коммутационной матрицы емкостью 1К (нанокоммутатор) по совместимому КМОП-технологическому процессу: Ag (верхний) и p-Si (нижний) - наноэлектроды и a-Si (аморфный) в качестве активного переключающего слоя; SiO2 – оксид на подложке; SOG – изоляционное стекло. На рис.2.34 показано тестирование 400 бит информации с использованием специального программного обеспечения. Цвета на карте показывают различные сопротивления точек коммутации в матрице в режиме ON.

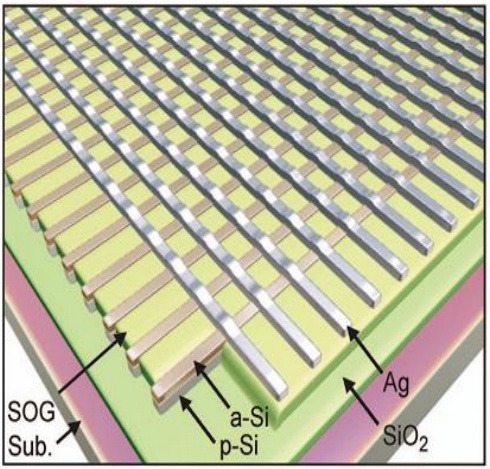


Рис.2.33. Фрагмент мемристорной коммутационной матрицы емкостью 1К (нанокоммутатор) по совместимому КМОП-технологическому процессу: Ag (верхний) и p-Si (нижний) - наноэлектроды и a-Si (аморфный) в качестве активного переключающего слоя; SiO2 – оксид на подложке; SOG – изоляционное стекло

Интеграция нанокоммутатора в ПЛИС позволит обеспечить более эффективную маршрутизацию сигналов и, в конечном счете, к повышению логической емкости и быстродействия. В HP Labs новая технология фигурирует под названием “программируемые межсоединения на базе нанопроводников” (Field Programmable Nanowire Interconnect, FPNI).

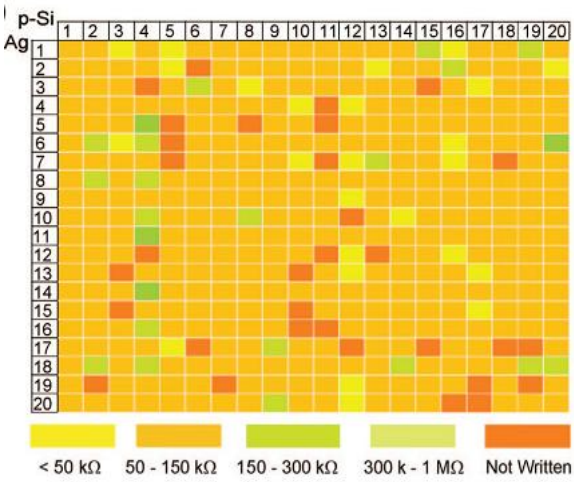


Рис.2.34. Тестируется 400 бит информации с использованием специального программного обеспечения. Цвета на карте показывают различные сопротивления точек коммутации в матрице в режиме ON

Гестеризесный резистивный переключатель, демонстрирующий нелинейную ВАХ, состоит из среды переключения (оксиды переходных металлов), зажатой между двумя электродами (бистабильный резистивный переключатель).

Сопротивление переключателя зависит не только от приложенного напряжения (тока), но и предистории его программирования. Значения сопротивлений мемристора в режимах Включено/Выключено зависит от того, прошло ли напряжение программирования пороговое напряжение записи $+V_{th}$ ($0 \to 1$, состояние 1) или пороговое напряжение стирания $-V_{th}$ ($1 \to 0$, состояние 0) в предыдущем цикле операции.

На рис.2.35 показана концепция создания гибридных КМОП/молекулярных устройств, в том числе и ПЛИС (CMOL FPGA, CMOS + MOLecular electronics) типа ППВМ. CMOL ПЛИС, как и обычная ПЛИС типа ППВМ с симметричной структурой состоит из плиток. В центре плитки находится триггерная ячейка, окруженная 12 базовыми ячейками. Базовая ячейка состоит из инвертора и двух ключей на проходных транзисторах, которые подключены к строкам и столбцам

системы адресации и передачи данных КМОП-структуры и имеют контакты к верхнему (синяя точка) и нижнему нанопроводникам (красная точка). В процессе конфигурации инверторы выключаются, и проходные транзисторы могут быть использованы для установки двоичных состояний каждого молекулярного устройства. Зеленой точкой обозначены молекулярные переключатели (наноприборы).

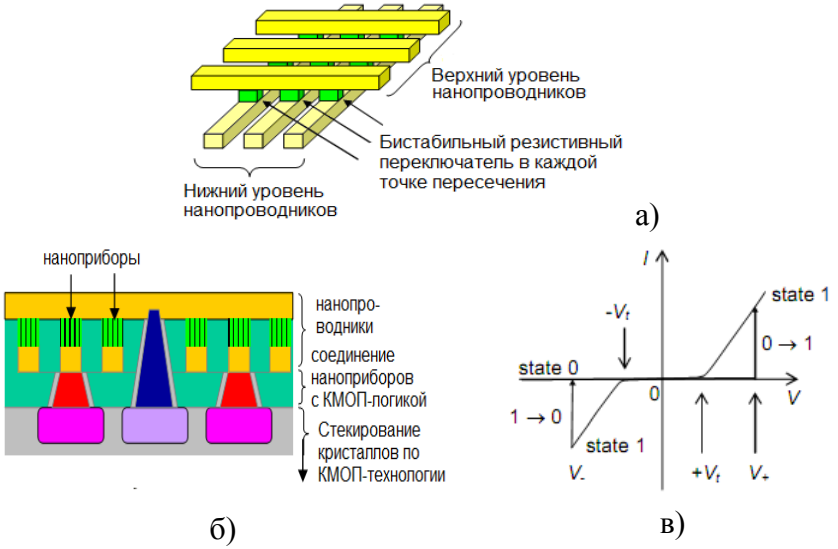
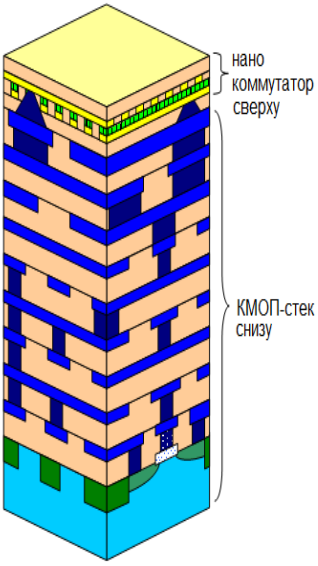
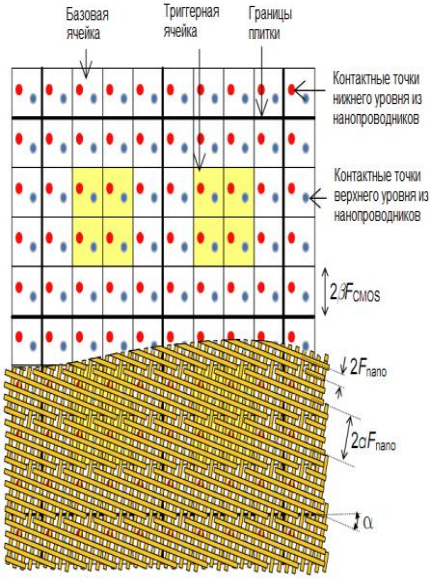


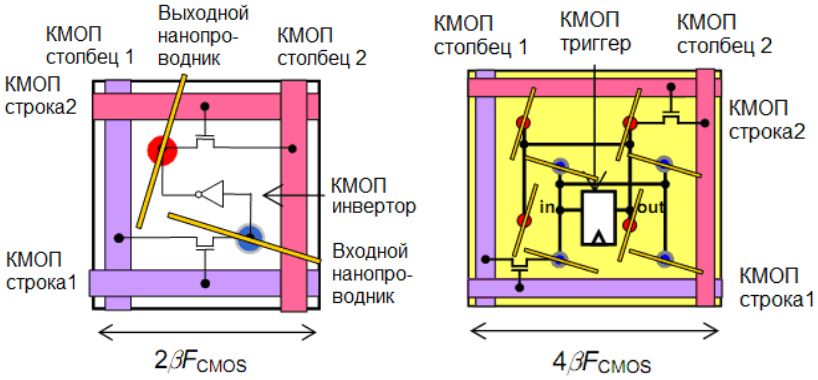
Рис.2.35. Нанокоммутатор (а); б) – приборная структура и ВАХ резистивного ключевого элемента с гистерезисным эффектом (в); г) КМОП-стек с нанокоммутатором; д) гибридная КМОП ПЛИС (CMOL FPGA); д) нескоммутированные элементы гибридной ПЛИС: базовая ячейка и триггерная ячейки; ж) – реализация логического элемента 2 ИЛИ-НЕ и эквивалентные схемы двух наноприборов в состоянии ON, проходного ключа и межсоединения



г)

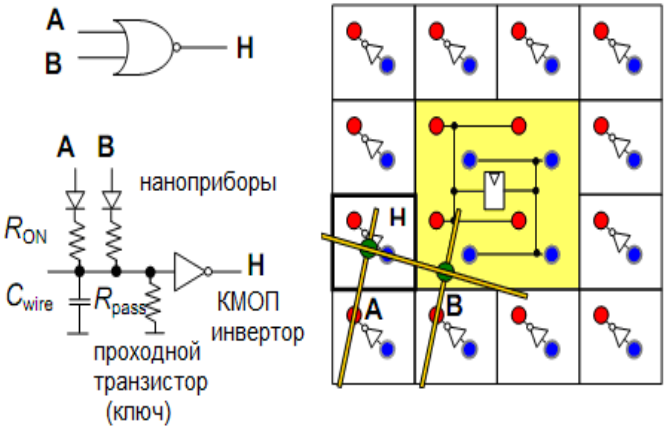


д)



е)

Рис.2.35. Продолжение



ж)

Рис.2.35. Окончание

Использование мемристоров в ПЛИС в качестве резистивной энергонезависимой памяти (resistive random-access memory, RRAM) значительно снижает задержки распространения сигналов в межсоединениях и позволяет экономить площадь кристалла, т.к. они располагаются в верхних слоях, над КМОП-логикой, что сравнимо с эффектом от использования 3D технологии стекирования кристаллов. Программируемые трассировочные ресурсы обычных ПЛИС занимают от 90 % всей площади ПЛИС, вносят 80 % вклад в задержки распространения сигналов, свыше 95 % потребляемой мощности приходится на коммутацию ключей в межсоединениях.

В настоящее время разработана новая архитектура mrFPGA ПЛИС типа ППВМ с использованием мемристоров в программируемых соединительных блоках и маршрутузаторах и КМОП-совместимого технологического процесса, позволяет полностью отказаться от использования электронного ключа (n-МОП транзистор) управляемого ячейкой памятью СОЗУ (пяти или шести транзисторная ячейка памяти) по КМОП-технологии, в качестве коммутирующего элемента (рис.2.36 и

рис.2.37). Мемристор сформирован пересечением электродов из платиновой нанопроволоки, разделенных пленкой диоксида титана.

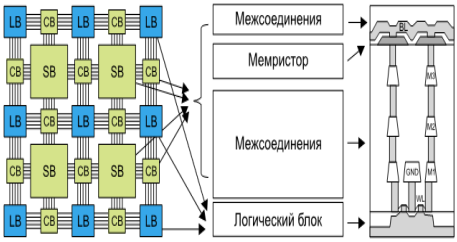


Рис.2.36. Академическая ПЛИС типа ППВМ с использованием мемристоров в маршрутизаторах и соединительных блоках

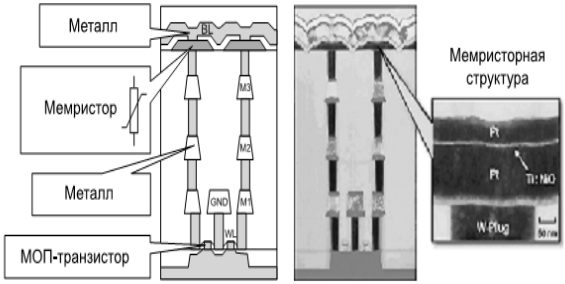


Рис.2.37. Схематический профиль структуры (а) и профиль, построенный с использованием просвечивающей электронной микроскопии

Анализ литературных данных показывает, что непрерывное усовершенствование структуры трассировочных ресурсов КМОП 2D ПЛИС исчерпало свои возможности, а дальнейший рост логической емкости ПЛИС возможен путем либо использования технологии стекирования кристаллов или путем перехода к использованию новейших достижений нанотехнологии: нанотрубки в качестве реконфигурационной

памяти ПЛИС или использование нанокоммутаторов (мемристорных структур) в маршрутизаторах и соединительных блоках.

Для субмикронных БИС такой паразитный эффект, как преобладание задержек распространения сигналов по токопроводящим дорожкам над задержками распространения сигналов в вентилях, из за наличия собственных сопротивлений и емкостей становится доминирующим. Так как удельное сопротивление токопроводящих дорожек снизить нельзя, то RC-нагрузка, создаваемая этими дорожками, может вызвать значительную задержку сигнала. С увеличением емкости нагрузки постоянная времени RC имеет большее значение и длительность переходных процессов существенно возрастает. Использование трехмерных БИС по субмикронным технологиям позволяет уменьшить общую длину межсоединений на кристалле, что положительно сказывается на задержках распространения сигналов.

Максимально возможный на сегодняшний день уровень интеграции БИС позволяет получить TSV-технлогия, которая обеспечивает более высокую плотность монтажа, большую функциональность, лучшие технические характеристики, более низкое энергопотребление, меньшую стоимость. Однако использование данной технологии требует решение ряда проблем.

TSV-технология позволяет значительно увеличить количество линий ввода/вывода, что радикальным образом приводит к повышению скорости трансляции данных и уменьшить энергопотребление, а также вызвать появление принципиально новых видов высокоэффективных устройств. Для проектирования 3D БИС с TSV необходимо использовать новые инструменты САПР для нанометровых проектов.

3. ПРОЕКТИРОВАНИЕ ПЛИС ТИПА ППВМ

3.1. Разработка функциональной модели ПЛИС типа ППВМ в САПР Quartus II с использованием технологии соединений multi-driver

Одноуровневая структура межсоединений ПЛИС типа ППВМ широко используется не только в индустриальных ПЛИС фирм Xilinx, Lucent Technologies, Vantis (рис.3.1), но и при разработке академических ПЛИС с архитектурой типа Island-style, например, 3D ПЛИС, комбинированных ПЛИС, в которых, в качестве массива конфигурационной памяти используются блоки памяти на нанотрубках. Многоуровневая структура межсоединений используется в ПЛИС Stratix, Cyclon и др. фирмы Altera.

Основные функциональные блоки: логический блок (ЛБ), соединительные блоки С1 и С2, коммутатор-маршрутизатор (S-блок или “свич-бокс”). Для ПЛИС типа ППВМ различных фирм характерно использование маршрутизаторов в трассировочных каналах. Быстродействие ПЛИС во многом определяется именно тем, как искустно спроектирована структура межсоединений.

При проектировании маршрутизатора могут быть использованы однонаправленные и двунаправленные программируемые межсоединения с применением в качестве ключа n-МОП транзистора и буфера с третьим состоянием (рис.3.2, 3.3). n-МОП ключ по своей сути является двунаправленным, а двунаправленный ключ на буферах с третьим состоянием может быть получен их параллельным соединением. На рис.3.2, *а* показана функциональная модель n-МОП шеститранзисторного ключа на логическом уровне, используемого в маршрутизаторах ПЛИС Xilinx серии XC4000 типа disjoint (рис.3.1). Для реализации модели шеститранзисторного ключа с использованием двунаправленных межсоединений применяются буферы для

восстановления уровня сигнала, буферы с третьим состоянием и мультиплексоры (рис.3.2, *а*).

Две схемы взаимоэквивалентны. Однако, второй вариант позволяет отказаться от использования буферов с третьим состоянием (рис.3.2, *б*), но приводит к увеличению числа межсоединений в канале, и как следствие, к увеличению разрядности мультиплексорных структур в соединительных блоках. В случае однонаправленных соединений, при движении сигнала в направлении стрелки, коммутация сигнала на противоположные направления осуществляется на соседние межсоединения, что приводит к появлению характерных “косичек” в трассировочных каналах.

Маршрутизатор построенный с использованием двунаправленных межсоединений и двунаправленных ключей получил название multi-driver, а с использованием однонаправленных межсоединений и мультиплексорных структур (коммутаторов) - single-driver switch block. При этом под термином “драйвер” подразумевается устройство, с помощью которого осуществляется коммутация межсоединений: n-МОП ключ, однонаправленный и двунаправленный ключ на буферах с третьим состоянием, мультиплексорная структура.

Рис.3.3 показывает идею использования двунаправленных и однонаправленных межсоединений в коммутаторах-маршрутизаторах. Рис.3.3, *а* показывает так же, что выходы логических блоков, объединенных в кластеры, могут быть подключены к трассировочным каналам с помощью выходных демультиплексоров на буферах с третьим состоянием (multi-driver) соединительных блоков.

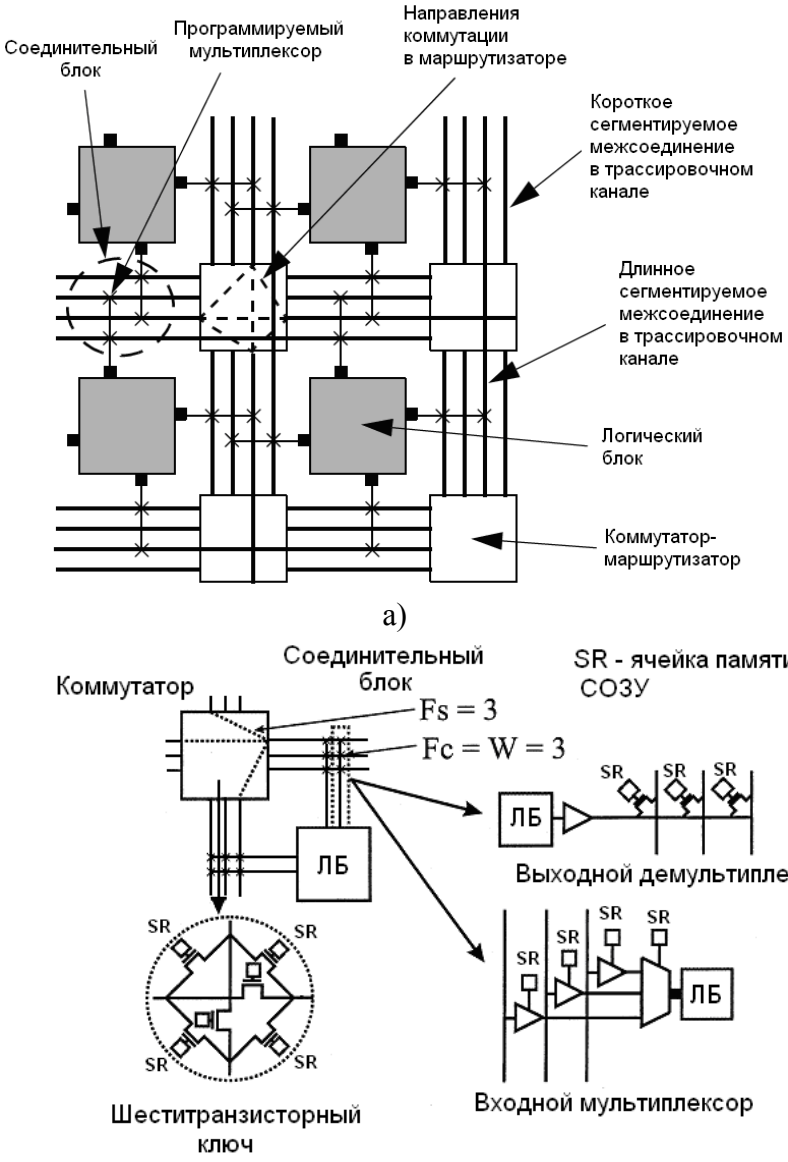


Рис.3.1. ПЛИС типа ППВМ с одноуровневой структурой межсоединений (а) и основные функциональные блоки (б)

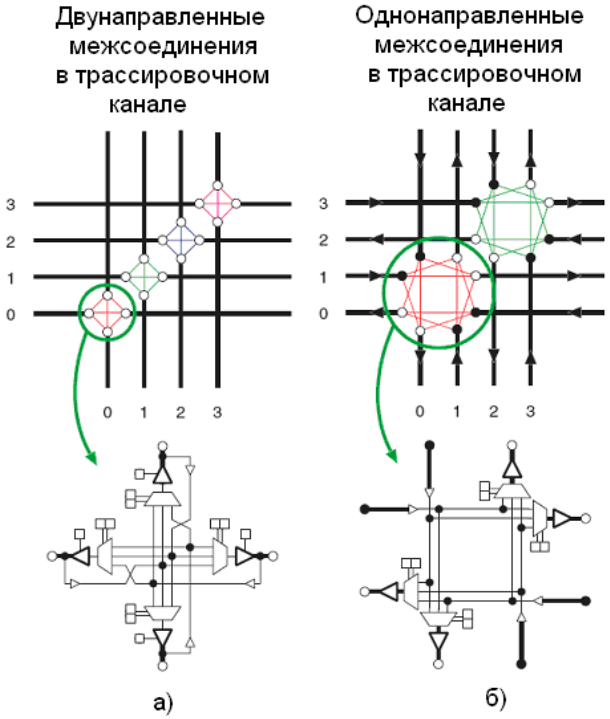


Рис.3.2. Коммутатор-маршрутизатор типа disjoint (S-блок) и функциональная модель шеститранзисторного ключа на логическом уровне: а) с использованием двунаправленных программируемых соединений; б) однонаправленные соединения

В настоящее время считается, что использование однонаправленных межсоединений в совокупности с мультиплексорными структурами в маршрутизаторах наиболее перспективно (рис.3.4), т.к. позволяет получать существенный выигрыш по быстродействию и по площади кристалла. Методология single-driver широко используется в современных сериях ПЛИС серий Stratix и Virtex. В современных ПЛИС серий Stratix используется патентованная технология

DirectDrive™, которая гарантирует идентичные соединительные ресурсы для любой реализуемой булевой функции, независимо от её месторасположения на кристалле, а в ПЛИС Virtex-5 для реализации логических функций и локальных межсоединений используется технология ExpressFabric™.

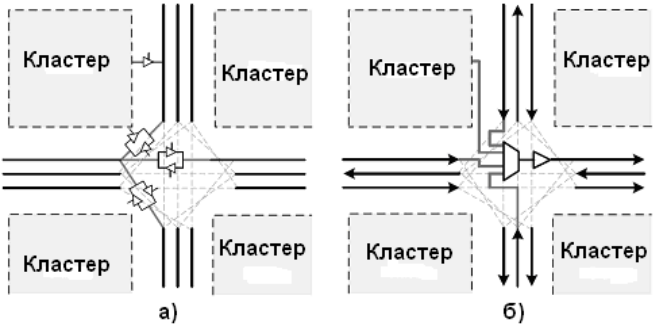


Рис.3.3. Коммутатор-маршрутизатор типа multi-driver (а) и single-driver (б)

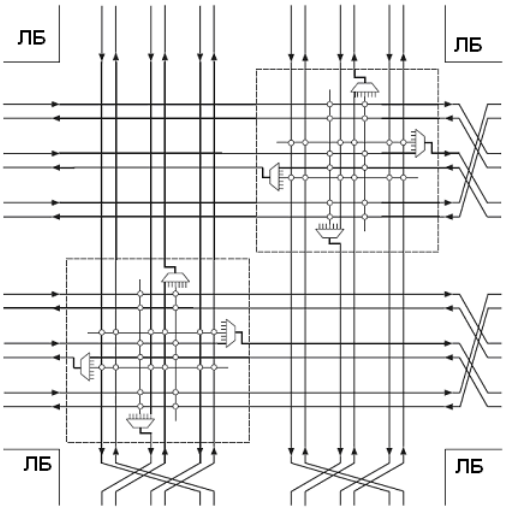


Рис.3.4. Два коммутатора-маршрутизатора типа single-driver с длинной сегмента L=3

Методология single-driver также распространяется на входные мультиплексоры и на демультплексоры соединительных блоков. На рис.3.3, *б* показано непосредственное подключение выходов логических блоков в мультиплексоры ближайших коммутаторов-маршрутизаторов, что позволяет отказаться от использования выходных демультиплексоров в соединительных блоках.

Для проектирования гомогенных академических ПЛИС с логическими блоками объединенных в кластеры может быть использован один из программных инструментов автоматизированного проектирования - VPR версии 4.3 (Versatile Place and Route), а для гетерогенных (с использованием гетерогенных блоков таких как, встроенные перемножители, блоки ОЗУ, блоки для реализации алгоритмов цифровой обработки сигналов) - VPR версии 5.0. VPR разработан в университете Торонто (Торонто, Канада, http://www.eecg.utoronto.ca/vpr), поддерживает маршрутизаторы обоих типов multi-driver и single-driver.

Кратко рассмотрим типовой маршрут проектирования гетерогенных академических ПЛИС, который предполагает использование следующих программных инструментов: ODIN, ABC, T-Vpack, VPR. ODIN конвертирует схемное описание некоторого сложно-функционального устройства (“бенч марк”, тестовая схема на языке Verilog HDL) в специальный файл в .blif формате, в котором выделяет логические вентили для описания логики устройства и “черные ящики” для гетерогенных блоков. Далее с использованием инструмента ABC (существуют и другие программные инструменты минимизации булевых функций в базис ПЛИС типа ППВМ с использованием 4-х входовой LUT-таблицы и D-триггера логического блока, такие как SIS и FlowMap) проводится логическая оптимизация схемы с использованием специального стиля описания независимого от технологии проектирования БИС (например, заказная специализируемая

БИС (ASIC), структурированная БИС (proASIC) или ПЛИС) и ее размещение в логические блоки академической ПЛИС. Выходным так же является файл в .blif формате, в котором выделяются LUT-таблицы, D-триггеры логических блоков и гетерогенные блоки. Инструмент T-Vpack перепаковывает файл в .blif-формате (LUT-таблицы и D-триггеры) в кластеры логических блоков, которые аналогичны кластерам в ПЛИС FLEX8, 10K или кластерам ПЛИС 5200 и Virtex, и формирует выходной файл в .net формате для VPR, который размещает кластеры логических блоков и гетерогенные блоки 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 трассировочного канала, быстродействию, экономии площади кристалла и др. Подобный маршрут проектирования существует и для коммерческих ПЛИС, например, индустриальный САПР ПЛИС Quartus II.

На рис.3.5 показана структура межсоединений в ПЛИС полученная с использованием VPR 5.0 для ширины трассировочного канала $W = 10$. В маршрутизаторе типа single-driver используются однонаправленные соединения с применением n-МОП ключей (кружки) и буферов с третьим состоянием (треугольники). Мультиплексоры соединительных блоков показаны, синим цветом, а выходные демультиплексоры – красным. Зеленым цветом показан S-блок. Если ЛБ заменяется на кластер (например, под кластером подразумевается 4 ЛБ), то структура межсоединений в S-блоке значительно усложняется (рис.3.6).

Разработаем модель гомогенной ПЛИС типа ППВМ с одноуровневой структурой межсоединений в САПР ПЛИС Quartus II компании Altera, для изучения особенностей программируемой коммутации.

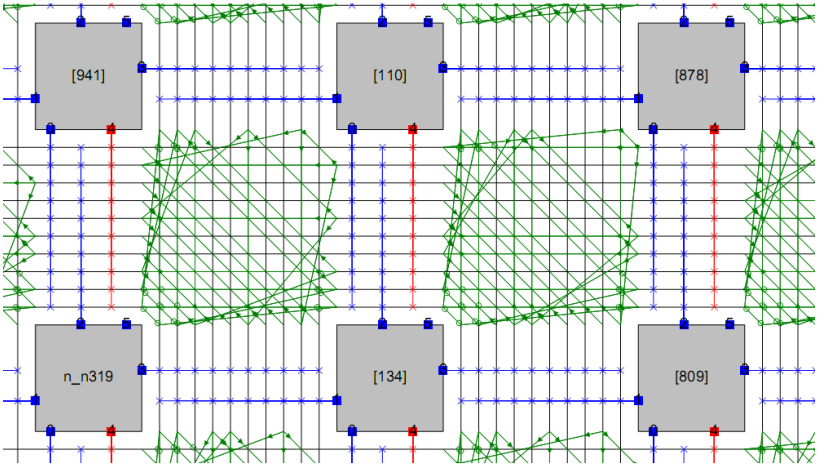


Рис.3.5. Структура межсоединений в коммутаторе-маршрутизаторе типа multi-driver при ширине канала $W = 10$, полученная с помощью VPR 5.0

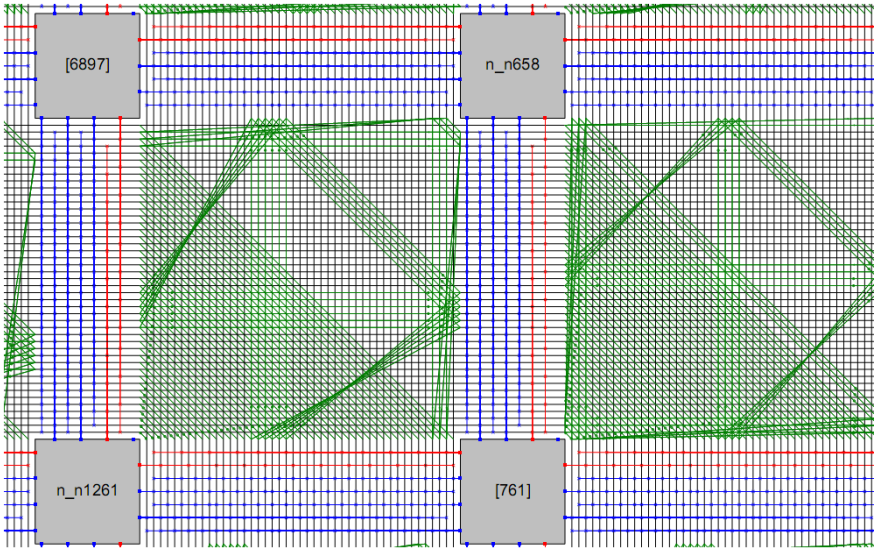


Рис.3.6. Структура межсоединений в коммутаторе-маршрутизаторе типа multi-driver при объединении логических блоков в кластеры, полученная с помощью VPR 5.0

При проектировании будем использовать следующие характеристики: коммутатор-маршрутизатор типа disjoint с коэффициентом разветвления по выходу $F_S = 3$; для соединительного блока: коэффициент объединения по входу логического блока $F_{C,in} = W$, коэффициент разветвления по выходу $F_{C,out} = W$, где W – ширина трассировочного канала. В нашем случае $W = 4$. На рис.3.1, *б* показан пример реализации ПЛИС с одноуровневой структорой с параметрами $F_S = 3$ и $F_{C,out} = F_{C,in} = W$. Для $F_S > 5$ уже необходимо использовать 3D коммутаторы-маршрутизаторы и переходить на 3D-технологии.

В разрабатываемой модели ПЛИС используется модель шеститранзисторного ключа показанная на рис.3.7. В качестве n-МОП ключа применяется буфер с третьим состоянием. Направление передачи сигналов в канале однонаправленное, с право на лево (рис.3.8). Коммутатор-маршрутизатор состоит из двух шеститранзисторных ключей, двух вертикальных и двух горизонтальных не сегментированных межсоединений.

Для программирования конфигурационных ячеек памяти СОЗУ необходимо два универсальных (сдвиговых) регистра, один 4–х разрядный (регистр столбца) и один 12-ти разрядный (регистр строки), 12 строк на 4 столбца, итого требуется запрограммировать 48 бит конфигурационной памяти одной плитки ПЛИС LEGO. Поэтому бит 20 в логическом блоке необходим для кратности 4. С этой же целью необходимы биты 43, 44 в соединительном блоке С1. На рис.3.9 показано, как можно сконфигурировать S-блок.

Все сигналы в горизонтальном трассировочном канале передаются с право на лево, в вертикальном канале – снизу вверх (рис.3.10). Все межсоединения в каналах сегментированы. На рис.3.10 так же показана схема сегментации межсоединений в каналах: 1, 2, 4 и 4.

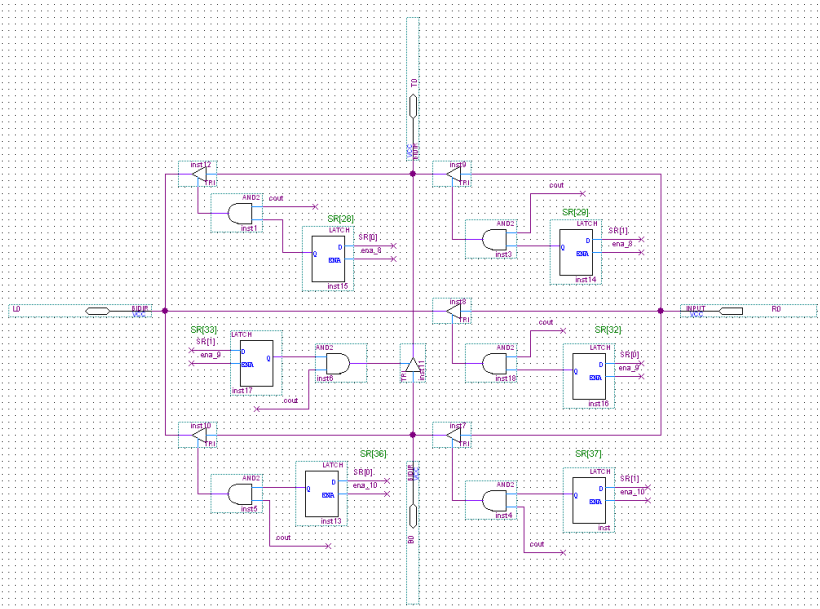


Рис.3.7. Шеститранзисторный ключ на буферах с третьим состоянием

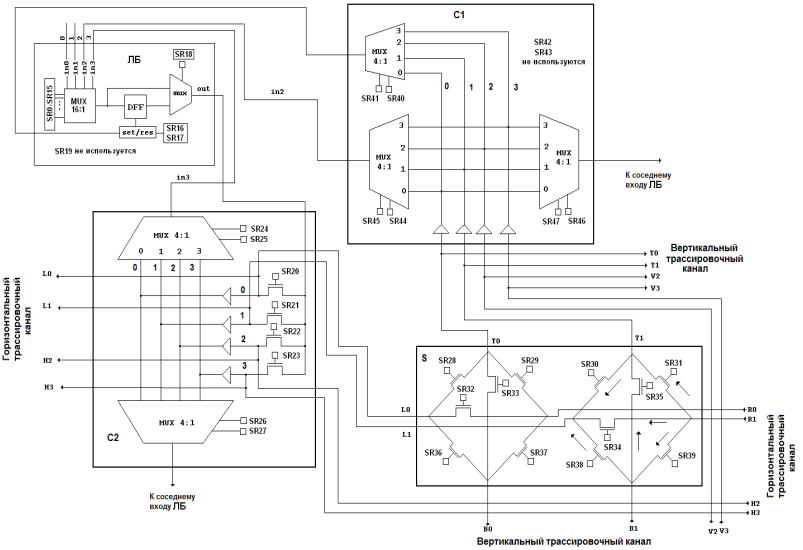


Рис.3.8. Принципиальная схема одной плитки ПЛИС

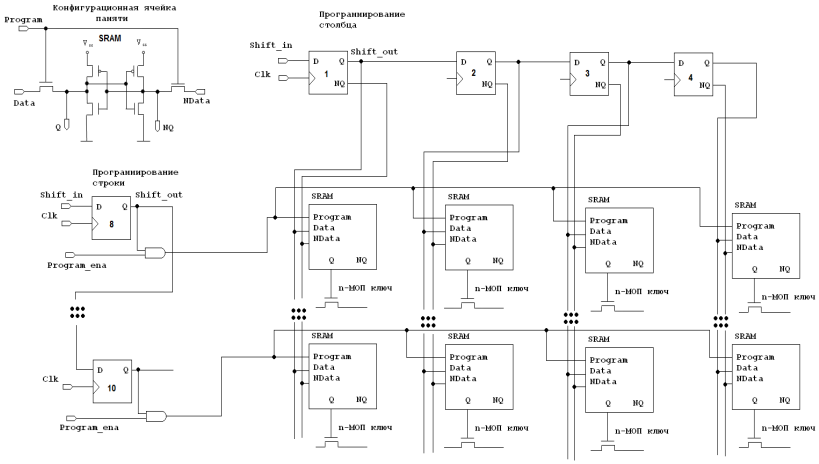


Рис.3.9. Использование 3 - х разрядов сдвигового 12 –ти разрядного регистра строки и 4 - х разрядного регистра столбца для конфигурирования S-блока, состоящего из 12 –ти n-МОП ключей

Межсоединение в трассировочном канале может проходить непрерывно через 1, 2, 4 и 4 логических блока, с право на лево и снизу вверх. Логический блок, соединительные блоки С1 и С2 и маршрутизатор объединяются в плитку. Для обеспечения периодичности сегментации потребовалось разработать схему в 16 плиток, назовем такую схему как “супер плитка”. Последующее ее тиражирование позволяет легко наращивать функциональную емкость проектируемой ПЛИС. На рис.3.11 показана детальная структура сегментации межсоединений 16 плиток и схема подключения соединительных блоков С1 и С2.

Электрическая схема одной плитки показана на рис.3.12. Для конфигурирования одной плитки требуется 48 бит памяти. Рассмотрим схему “загрузчика”. Загрузка конфигурации в плитку осуществляется с помощью блока downloader_rom (рис.3.13), который состоит из 4-х разрядного суммирующего счетчика с модулем счета 13, ПЗУ емкостью 4

бита на 16 слов на базе мегафункции lpm_rom и дешифратора 4 в 12, который выбирает нужную строку с четырьмя ячейками памяти в плитке, в зависимости от состояния счетчика. По окончании загрузки счетчиком вырабатывается сигнал cout, высокий уровень которого разрешает перезапись информации из ячеек памяти в выходном мультиплексоре соединительного блока С2 и шеститранзисторных ключей на буферах с третьим состоянием в S-блоке, остальные ячейки (16 ячеек для настройки LUT-таблицы и все для управления адресными входами мультиплексоров в соединительных блоках) памяти доступны сразу же после записи в них конфигурационной информации. Это позволяет избежать открывание буферов с третьим состоянием во время конфигурации плитки.

В качестве ячеек памяти используются защелки (однотактный триггер или триггер тактируемый уровнем синхросигнала). На рис.3.14 показаны 16 конфигурационных ячеек памяти 4-х входовой LUT-таблицы. Ячейки памяти объединены в строки по 4 бита. Каждая строка выбирается с помощью дешифратора строки “загрузчика”. Всего 12 строк по 4 бита.

До начала работы необходимо сформировать конфигурационный файл с расширением mif. На рис.3.15 показан файл конфигурации ПЗУ (адреса строк начинаются с нуля). LUT-таблица запрограммирована на выполнение булевой функции $f = x3 \oplus x2 \oplus x1 \oplus x0$. Таких файлов должно быть заполнено 16 шт. При включении осуществляется параллельная конфигурация каждой плитки с помощью своего “загрузчика”. Для упрощения будем использовать 16 одинаковых конфигурационных файлов.

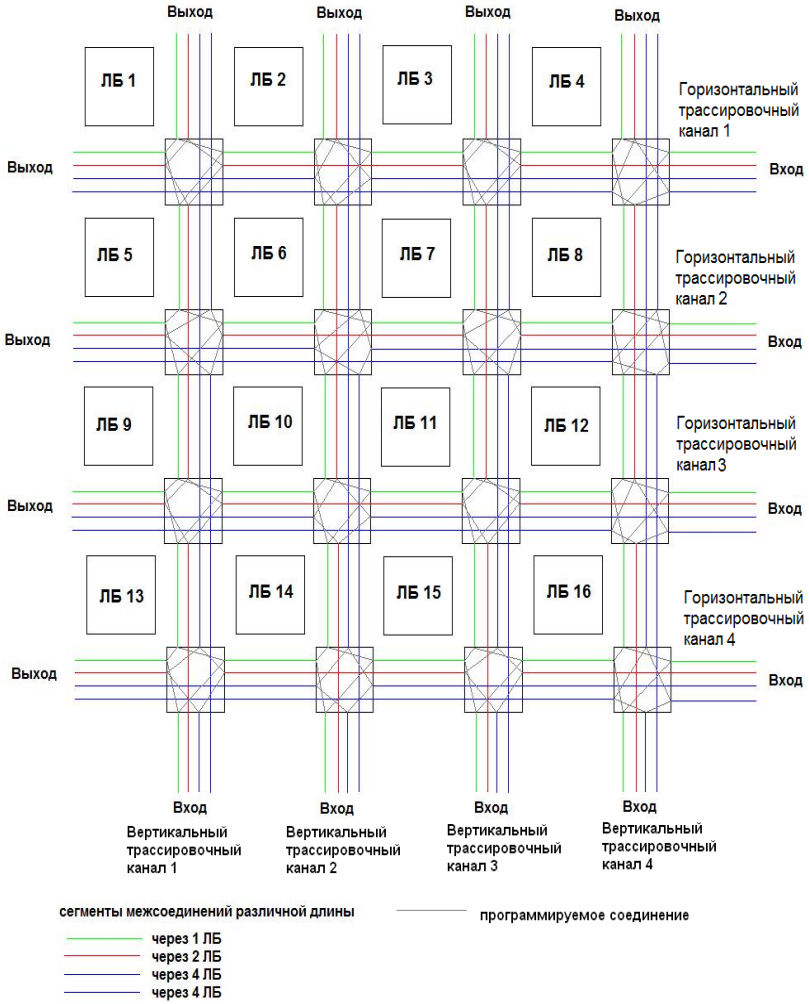


Рис.3.10. Программируемая структура межсоединений 16 плиток

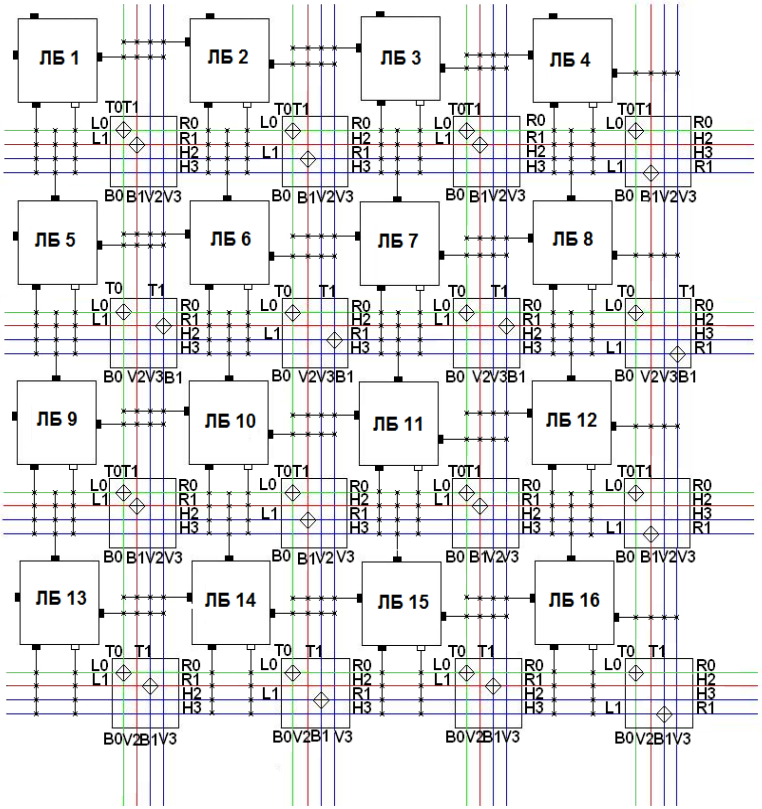


Рис.3.11. Детальная структура межсоединений 16-ти плиток

В табл.3.1 более подробно поясняется конфигурационный файл плитки 1. Плитка 1 согласно структуре межсоединений имеет два внешних входа in0 и in1 LUT-таблицы (входы t1_in_lut_0 и t1_in_lut_1 доступны с контактных площадок). А к входам in2 и in3 можно “добраться” только через маршрутизаторы со входов вертикального (соединительный блок С1) и горизонтального (соединительный блок С2) трассировочных каналов. Согласно рис.3.10 и рис.3.11 эти входы канала 1 находятся с права и с

низу, а выходы с лева и с верху, согласно схеме распространения сигнала в S-блоке (рис.3.8).

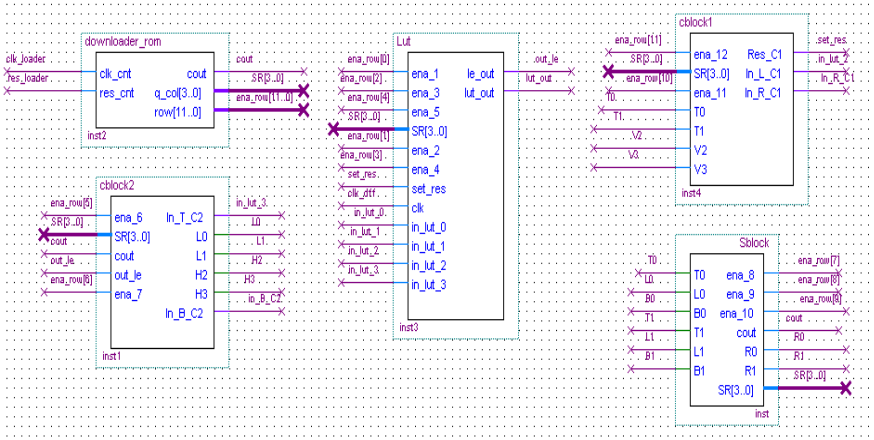


Рис.3.12. Электрическая схема одной плитки в САПР ПЛИС Quartus II

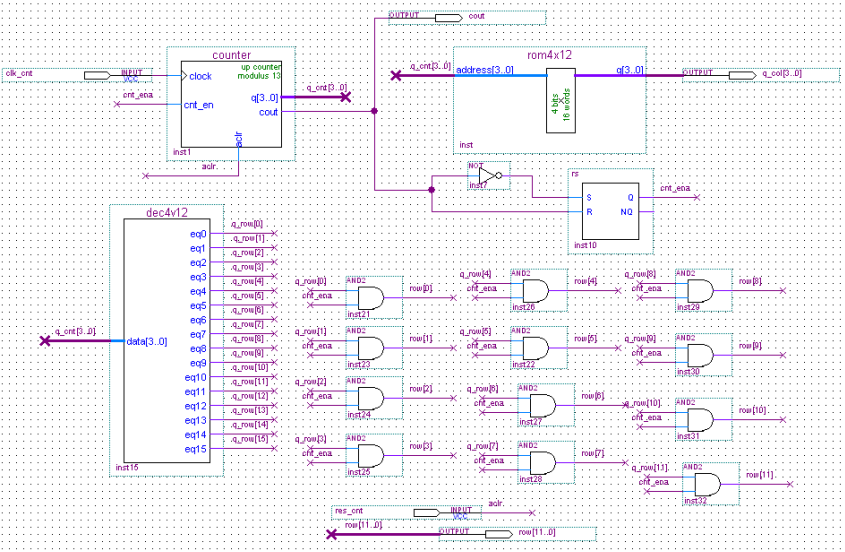


Рис.3.13. Электрическая схема “загрузчика” (блок downloader_blok) ячеек памяти в плитке

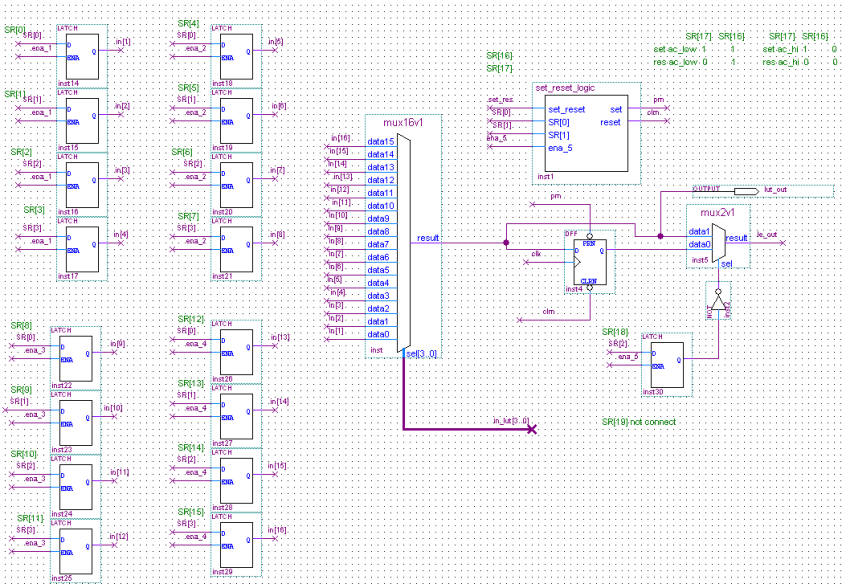


Рис.3.14. Электрическая схема логического блока

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	0110	1001	1001	0110	0101	0001	0001	0000
8	1110	0000	0010	0000				

а)

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	6	9	9	6	5	1	1	0
8	14	0	2	0				

б)

Рис.3.15. Файл конфигурации ПЗУ (адреса строк начинаются с нуля): а) содержимое представлено в двоичной системе; б) в десятичной

Используем нумерацию ячеек памяти табл.3.1, которая начинается с нуля. Под термином “трек” будем подразумевать короткое межсоединение. Предположим, что мы хотим считать содержимое LUT таблицы при комбинации на адресных входах 1100 (12D) (in3=1, in2=1, in1=0, in0=0) и вывести результат в горизонтальный трассировочный канал 1 на трек 0 (рис.3.10). Нумерация треков в каналах показана на

рис.3.8. Согласно табл.3.1 с выхода LUT-таблицы (lut_out) должны считать лог. 0.

Конфигурационный файл настроен так (табл.3.1), что выход логического блока (le_out) поступает на трек 0 соединительного блока С2, для этого в ячейку памяти с номером 20 должна быть записана лог.1, которая откроет буфер с третьим состоянием в выходном демультиплексоре типа track-to-pin блока С2. При этом когда, выход логического блока передается в горизонтальный канал на трек 0 находящийся справа, буфер с третьим состоянием управляемый ячейкой 32 в шеститранзисторном ключе должен быть закрыт, т.е. вход R0 должен быть отключен.

На трек 1 блока С2 со входа R1 шеститранзисторного ключа подается лог.1, что обеспечивает подачу сигнала лог 1 на вход in3 при условии что на адресных входах верхнего мультиплексора соединительного блока С2 в ячейках 25 и 24 записаны лог. 0 и лог.1. Для этого необходимо в ячейку памяти с номером 34 записать 1.

Для подачи лог.1 на вход in2 из вертикального трассировочного канала со входа B0 шеститранзисторного ключа необходимо в ячейку с номером 33 записать лог.1. Сигнал Set/Reset подается со входа B1 на трек 1 верхнего мультиплексора блока С1. Для этого в ячейку с номером 35 необходимо записать лог.1. На адресные входы мультиплексора подается комбинация SR41 лог. 1, SR40 лог.0. Если на вход B1 подать сигнал лог.0, то D-триггер логического блока будет сброшен в лог.0. Рис.3.16 демонстрирует временные диаграммы работы плитки 1.

Так как все плитки конфигурируются одновременно одной и той же информацией, то сигналы со входов вертикального (col1_in[3..0]) и горизонтального трассировочных (row1_in[3..0]) каналов под номерами 1 должны беспрепятственно быть поданы на входы R0, R1, H2, H3 и B1, B2, V2, V3 плитки 1. На рис.3.17 показан фрагмент

схемы “супер плитка” (плитки 1, 2, 5 и 6) для демонстрации организации электрических связей между плитками. На рис.3.18 показаны временные диаграммы работы “супер плитки”. Тестируется 12 ячейка памяти LUT-таблицы, для этого на адресные входы мультиплексора 16 в 1 логического блока подается комбинация 1100 (12D). На выходах горизонтального трассировочного канала 1 (row1_out[3..0]) должна появиться информация 0100: лог. 0 на треке 0 (L0, см. рис.10); лог. 1 на треке 1 (L1); лог. 0 на треке 2 (H2) и лог. 0 на треке 3 (H3).

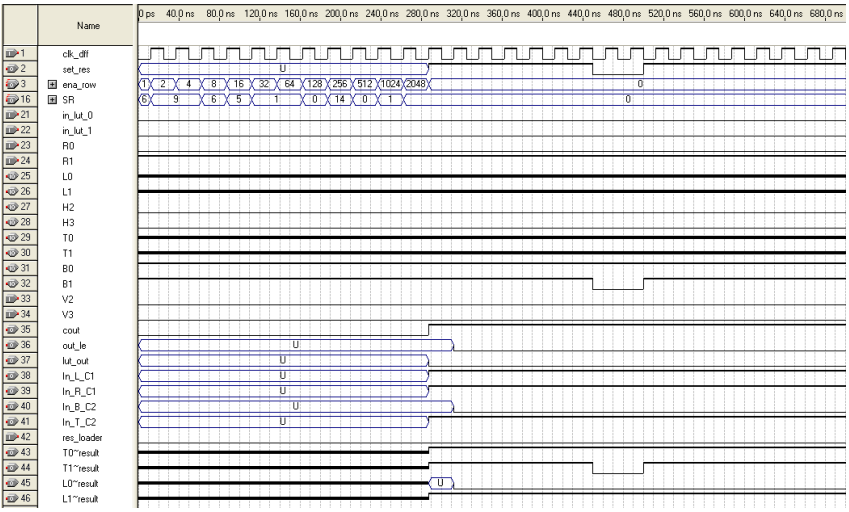


Рис.3.16. На адресные входы LUT-таблицы через S-блок плитки 1 подается комбинация 1100 (12D), а с выхода логического блока (t1_out_le) считывается содержимое 12-той ячейки памяти - лог. 0 (согласно табл.1)

Изменим содержимое 12 ячейки на лог.1, при той же комбинации на адресных входах 1100, на выходе логического блока (t1_out_le) должен появиться сигнал лог.1. На вход B1, после того как на выходе t1_out_le появится сигнал лог.1

подадим сигнал лог.0, что должно привести к сбросу триггера логического блока. Рис.3.19 демонстрирует данную ситуацию.

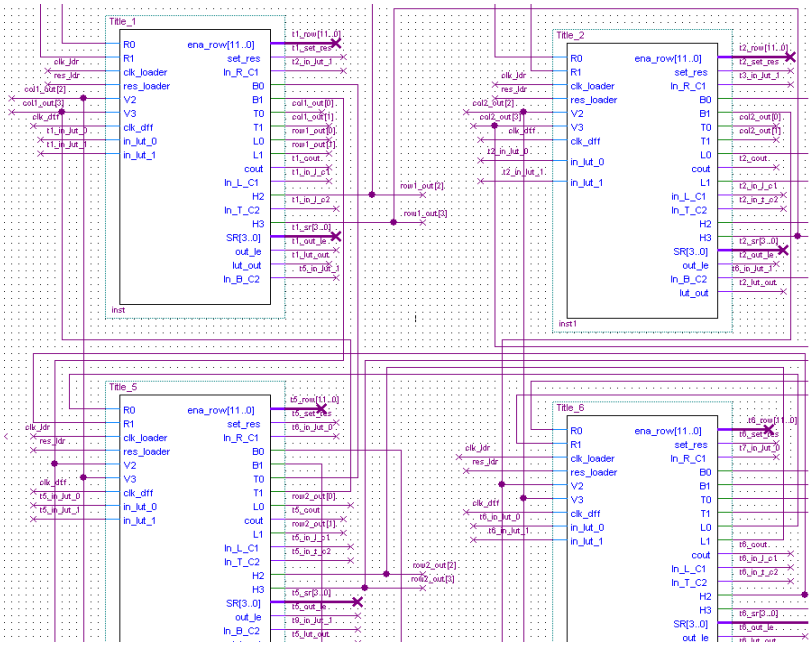


Рис.3.17. Фрагмент схемы “супер плитка” (плитки 1, 2, 5 и 6)

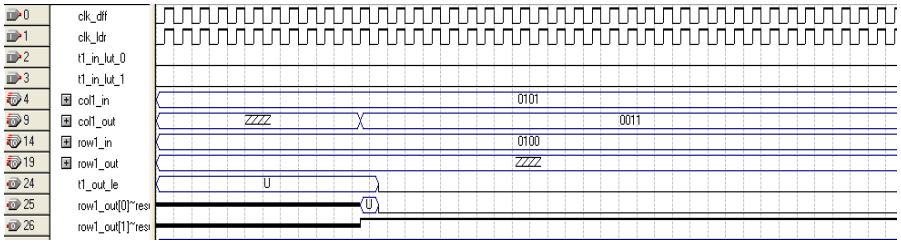


Рис.3.18. На адресные входы LUT-таблицы плитки 1 через S-блоки горизонтального и вертикального трассировочных каналов 1 подается комбинация 1100 (12D) с выхода логического блока (t1_out_le) считывается лог. 0, этот же сигнал поступает на трек 0 (row1_out[0]) горизонтального трассировочного канала 1

Таблица 3.1

Файл конфигурации одной плитки (младшие разряды с лева, SR3, SR2, SR1, SR0) на выполнение булевой функции $f = x3 \oplus x2 \oplus x1 \oplus x0$ от четырех входных переменных

N строки	Строка 0				Строка 1				Строка 2				Строка 3			
N ячейки	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
Содержимое BIN	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
DEC	6				9				9				6			

Строка 4				Строка 5				Строка 6				Строка 7				Строка 8			
19	18	17	16	23	22	21	20	27	26	25	24	31	30	29	28	35	34	33	32
0	1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1	1	0
5				1				1				0				14			

Строка 9				Строка 10				Строка 11											
39	38	37	36	43	42	41	40	47	46	45	44								
0	0	0	0	0	0	0	1	0	0	0	0								
0				1				0											

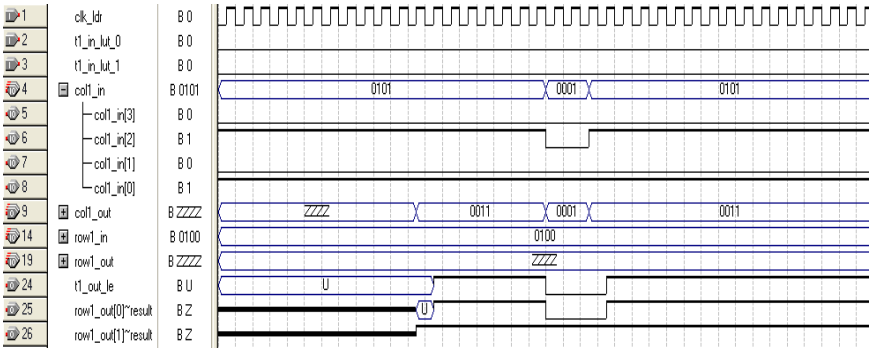


Рис.3.19. На адресные входы LUT-таблицы плитки 1 через S-блоки горизонтального и вертикального трассировочных каналов 1 подается комбинация 1100 (12D) с выхода логического блока (t1_out_le) считывается лог. 1, сигнал лог.0 на входе B1 приводит к сбросу триггера ЛБ

В данном разделе разработана упрощенная модель ПЛИС типа ППВМ с одноуровневой структурой межсоединений с функциональной емкостью 16 ЛБ в САПР Quartus II. В трассировочном канале используются однонаправленные сегментированные межсоединения различной длины, 1, 2, 4 и 4, проходящие непрерывно через 1, 2, 4 и 4 логических блока по вертикали и горизонтали. В качестве однонаправленного ключа используется буфер с третьим состоянием. Для реализации проекта требуются ПЛИС со встроенными блоками памяти, необходимые для хранения файлов конфигурации. Для конфигурирования “супер плитки” требуется 768 бит памяти и 32 шеститранзисторных ключа, для реализации которых необходимо 192 буфера с третьем состоянием.

3.2. Разработка модели ПЛИС типа ППВМ с одноуровневой структурой межсоединений с использованием технологии соединений single-driver в системе визуально-имитационного моделирования Matlab/Simulink

Успехи в области создания академических ПЛИС и программных инструментов, оказали существенное влияние на развитие архитектур индустриальных ПЛИС.

В академических ПЛИС для обеспечения программируемой коммутации существует две технологии соединений: multi-driver и single-driver. В настоящее время преимущество отдается технологии single-driver, т.к. ее использование позволяет получать существенный выигрыш по сравнению с технологией multi-driver как по быстродействию, так и по площади кристалла.

На рис.3.20 показаны основные функциональные блоки (логический блок (ЛБ), два соединительных блока С1 и С2, маршрутизатор (S)) ПЛИС с одноуровневой системой межсоединений и принцип коммутации пар разнонаправленных межсоединений по технологии single-driver. Соединительные блоки подключают входы/выходы логического блока к горизонтальному и вертикальному трассировочному каналу. Маршрутизатор осуществляет коммутацию сигналов (межсоединений) в трассировочных каналах. Так же показан наиболее распространенный способ коммутации сигналов с помощью мультиплексоров и принцип сегментации межсоединений (подобный принцип сегментации single, Double, HEX-3, HEX-6 используется в ПЛИС серии Virtex II компании Xilinx, которые классифицируются как ПЛИС с одноуровневой структурой межсоединений). Межсоединение по которому сигнал распространяется только в одну сторону получило название – однонаправленное.

В качестве учебных целей разработаем модель ПЛИС типа ППВМ с одноуровневой системой межсоединений и с использованием технологии соединений single-driver в системе Matlab/Simulink и покажем, как можно в автоматическом режиме получить код высокоуровневого языка описания аппаратных средств (VHDL). Полученный код может быть использован для разработки функциональной модели проектируемой ПЛИС, например в САПР Quartus II.

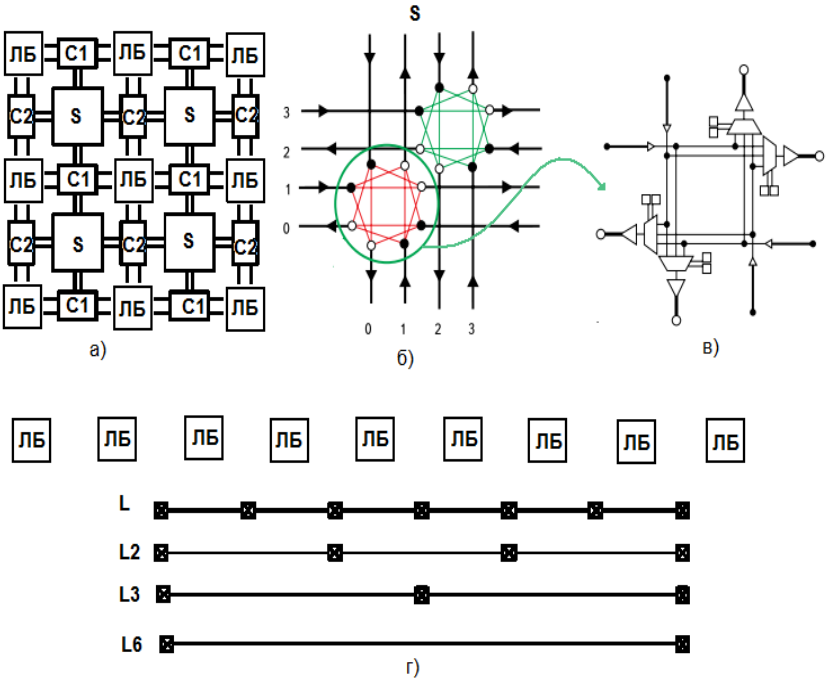


Рис.3.20. Основные функциональные блоки ПЛИС с одноуровневой системой межсоединений (а); б) – коммутация двух пар разнонаправленных межсоединений в горизонтальном и вертикальном направлениях; в) – принцип коммутации разнонаправленных межсоединений с использованием мультиплексоров; г) принцип сегментации межсоединений (через один ЛБ, через два ЛБ, через три ЛБ, через шесть ЛБ)

Модель ПЛИС реализована в формате с фиксированной запятой (точкой), с использованием fi-объектов системы Matlab/Simulink. Simulink – графическая среда визуально-имитационного моделирования аналоговых и дискретных систем. Предоставляет пользователю графический интерфейс для конструирования моделей из стандартных функциональных блоков. Simulink работает с линейными, нелинейными, непрерывными, дискретными и многомерными системами. Система Matlab/Simulink содержит встроенный генератор кода языка описания аппаратных средств HDL (Simulink HDL Coder). Simulink HDL Coder – программный продукт для генерации VHDL-кода без привязки к конкретной архитектуре ПЛИС и платформе по Simulink-моделям. Используем следующий формат, для представления десятичных чисел:

a = fi(v, s, w, f),

где v – десятичное число, s – знак (0 (false)– для чисел без знака и 1 (true) – для чисел со знаком), w - размер слова в битах (целая часть числа), f – дробная часть числа в битах.

Формат a = fi(v, s, w, f, fimath), позволяет задать режим округления (Roundmode) – ‘floor’ – округление вниз; реакцию на переполнение (OverflowMode) – ‘wrap’ – перенос, при выходе значения v из допустимого диапазона, “лишние” старшие разряды игнорируются. При выполнении операций умножения (‘ProductMode’) и сложения (‘SumMode’), для повышения точности вычислений (precision) используется машинное слово шириной в 32 бита.

```
% HDL specific fimath
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);
```

Массив ПЛИС разобъем на “плитки”. “Плитка” – минимальная структурная единица. В плитку (верхняя левая в массиве плиток) включают ЛБ, два соединительных блока, маршрутизатор (рис.3.21). На рис.3.22 показан принцип коммутации межсоединений в разрабатываемой модели. L2-маршрутизатор обеспечивает длину сегмента межсоединения в два ЛБ.

По четырем сторонам маршрутизатора располагаются многовходовые мультиплексоры, в которых сегментируется только лишь одна из двух пар разнонаправленных межсоединений в горизонтальных и вертикальных направлениях. Не сегментируемая пара разнонаправленных межсоединений перекручивается “косичкой” с сегментируемой парой за пределами плитки (рис.3.22).

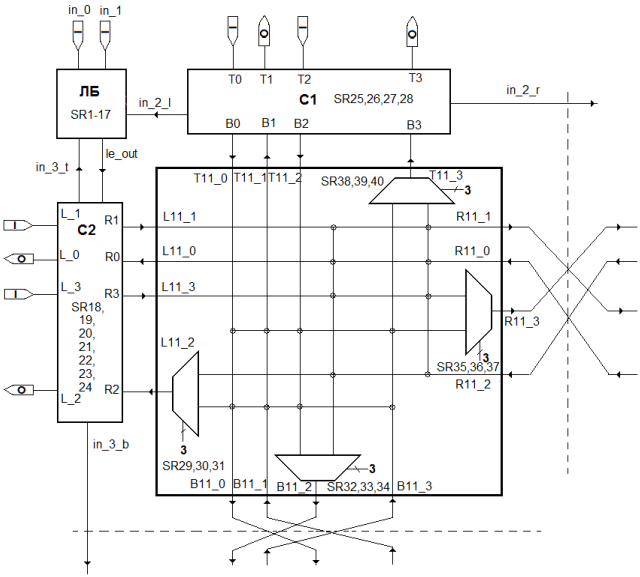


Рис.3.21. Плитка ПЛИС типа ППВМ с одноуровневой системой межсоединений

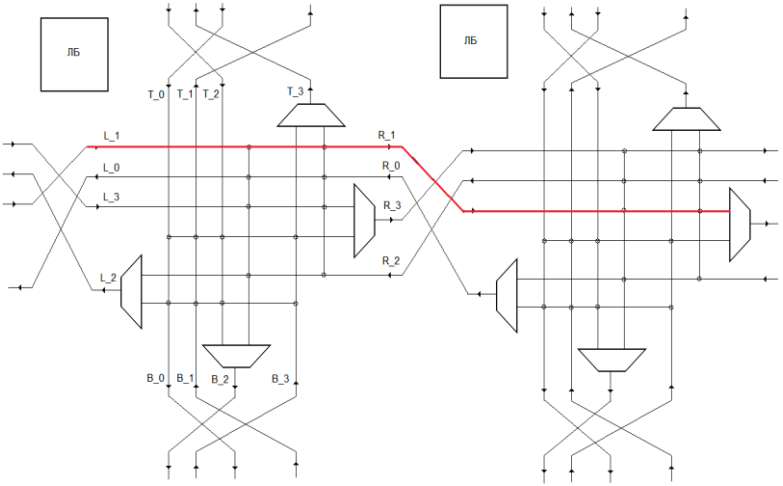


Рис.3.22. L2-маршрутизатор. Коммутирует две пары разнонаправленных межсоединений в горизонтальном и вертикальном направлениях, обеспечивая длину сегмента в два логических блока (длинная линия L=2)

Тестирование плитки на выполнение булевой функции 4И-НЕ в системе Matlab/Simulink показано на рис.3.23. На рис.3.24 показан нижний уровень иерархии. Для конфигурирования плитки необходимо 40 бит памяти. Конфигурационная карта памяти одной плитки ПЛИС показана в табл.3.2. В табл.3.3 приведен тест на выполнение булевой функции 4 И-НЕ. На адресные входы мультиплексора 16 в 1 (LUT-таблица) подается комбинация из четырех логических единиц 1111, а на информационные входы подключается 16 ячеек памяти с содержимым, отражающим принцип работы логического элемента 4И-НЕ (таблица истинности) при этом, на выходе логического блока ожидаем логический 0. Остальные 24 ячейки памяти необходимы для программируемой коммутации.

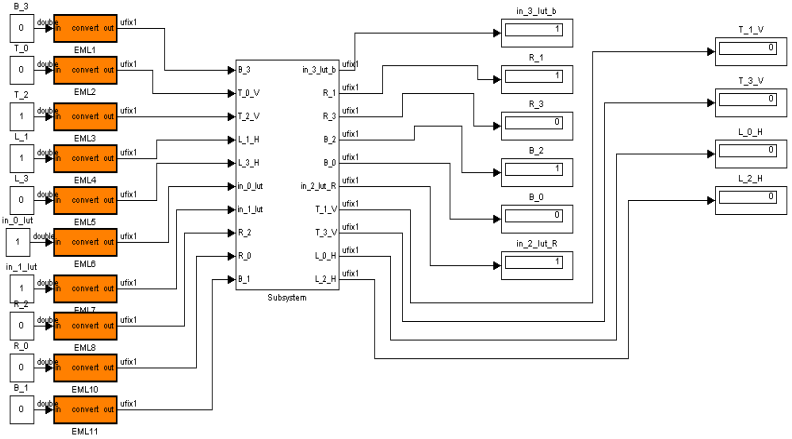


Рис.3.23. Плитка в системе Matlab/Simulink. Верхний уровень иерархии

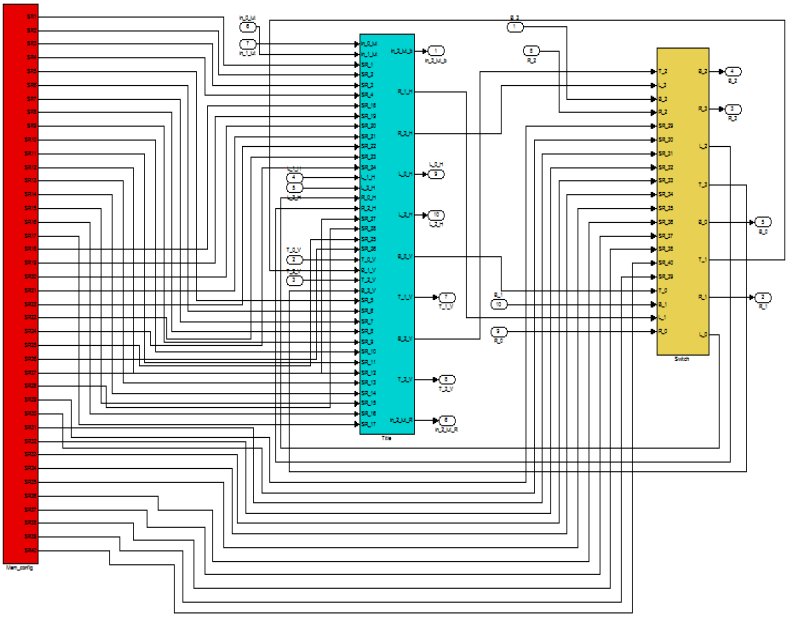


Рис.3.24. Нижний уровень иерархии плитки. Массив конфигурационной памяти (40 бит), логический блок с соединительными блоками, маршрутизатор

Таблица 3.2

Конфигурационная карта памяти одной плитки ПЛИС

Функциональный блок	Конфигурационные биты (ячейки памяти)	Примечание
1	2	3
Логический блок		
LUT-таблица	SR1-SR4; SR5-SR8; SR9-SR12; SR13-SR16	In_0, In_1, In_2_l, In_3_t
Выходной мультиплексор 2 в 1	SR17	SR17=1 – регистерный выход SR17=0 – комбинаторный выход
Соединительный блок С2		
Crossbar (подключает выход ЛБ и МС L_1, R_0, L_3, R_2 к трассировочному каналу или ко входам In_3_lut_t, In_3_lut_b ЛБ)		
Демультиплексор 1 в 4 (SR_demux)	SR18-19	Выход ЛБ подключается к одному из МС R_1, L_0, R_3, L_2
SR_En_demux	SR20	Разрешение подключения выхода ЛБ к одному из МС R_1, L_0, R_3, L_2
Мультиплексоры		
Мультиплексор 4 в 1 (верхний)	SR21-22 (SR_mux_t)	Подключает одно из МС L_1, R_0, L_3, R_2 ко входу In_3_t ЛБ
Мультиплексор 4 в 1 (нижний)	SR23-24 (SR_mux_b)	Подключает одно из МС L_1, R_0, L_3, R_2 ко входу In_3_b ЛБ

Продолжение табл. 3.2

1	2	3
Маршрутизатор, S-блок		
Мультиплексор 5 в 1 (левый)	SR 29,30,31	Подключает одно из МС R_2, B_0, B_1, T_2, B_3 к МС L_2
Мультиплексор 5 в 1 (нижний)	SR 32,33,34	Подключает одно из МС T_2, L_1, L_0, L_3 к МС B_2
Мультиплексор 5 в 1 (правый)	SR 35,36,37	Подключает одно из МС L_3, B_0, B_1, T_2, B_3 к МС R_3
Мультиплексор 5 в 1 (верхний)	SR 38,39,40	Подключает одно из МС B_3, L_1, L_0, L_3, R_2 к МС T_3
Соединительный блок C1		
Мультиплексор 4 в 1 (левый)	SR25-26	Подключает одно из МС T_0, B_1, T_2, B_3 ко входу in_2_lut_L
Мультиплексор 4 в 1 (правый)	SR27-28	Подключает одно из МС T_0, B_1, T_2, B_3 ко входу in_2_lut_R

* МС -межсоединение

Таблица 3.3

Тест на выполнение булевой функции 4 И-НЕ

Функциональный блок	Конфигурационные биты (ячейки памяти)	Примечание
1	2	3
Логический блок		
LUT-таблица	SR1-SR4 =1; SR5-SR8=1; SR9-SR12=1; SR13-SR15 =1; SR16=0	При In_0=1, In_1=1, In_2_l=1, In_3_t=1 на выходе le_out ожидаем лог.0 Доступ к In_2_l обеспечивается со входа T_2 Доступ к In_3_t обеспечивается со входа L_1
Выходной мультиплексор 2 в 1	SR17	SR17=1 – регистерный выход SR17=0 – комбинаторный выход
Соединительный блок C2		
Демультиплексор 1 в 4 (SR_demux)	SR18-SR19	Выход ЛБ подключаем к межсоединению R_3 при этом L_3=0 SR19=1 SR18=0
SR_En_demux	SR20	Разрешение подключения выхода ЛБ к МС R_1, L_0, R_3, L_2 SR20=1

Продолжение табл. 3.3

1	2	3
Мультиплексор 4 в 1 (верхний)	SR21-22 (SR_mux_t)	Подключает МС L_1 ко входу In_3_t ЛБ SR22=0 и SR21=0
Мультиплексор 4 в 1 (нижний)	SR23-24 (SR_mux_b)	Подключает МС L_1 ко входу In_3_b ЛБ SR24=0 и SR23=0
Маршрутизатор, S-блок		
Мультиплексор 5 в 1 (левый)	SR 29,30,31	Подключает МС R_2 к МС L_2 SR 29,30,31=0
Мультиплексор 5 в 1 (нижний)	SR 32,33,34	Подключает МС T_2 к МС B_2 SR 32,33,34=0
Мультиплексор 5 в 1 (правый)	SR 35,36,37	Подключает МС L_3 к МС R_3 SR 35,36,37=0
Мультиплексор 5 в 1 (верхний)	SR 38,39,40	Подключает МС B_3 к МС T_3 SR 38,39,40=0
Соединительный блок C1		
Мультиплексор 4 в 1 (левый)	SR25-26	Подключает МС T_2 ко входу in_2_lut_L SR26=1 и SR25=0
Мультиплексор 4 в 1 (правый)	SR27-28	Подключает МС T_2 ко входу in_2_lut_R SR28=1 и SR27=0

На рис.3.25 показан фрагмент массива конфигурационной памяти плитки. Конфигурационные биты задаются как константы типа double и сохраняются в блоках памяти. Для автоматической генерации кода языка VHDL необходимо осуществить конвертацию типа double в тип ufix (пример 1). На рис.3.26 показан логический блок и два соединительных блока C1 и C2. Адресные шины

мультиплексоров соединительных блоков и LUT-таблицы организуются с помощью функции объединения битов bitconcat (пример 2).

На рис.3.27 показан логический блок плитки, который состоит из LUT-таблицы (мультиплексора 16 в 1), элемента памяти (триггера), мультиплексора 2 в 1. На рис.3.28 показан соединительный блок С2, который осуществляет коммутацию выхода ЛБ к одному из межсоединений трассировочного канала и коммутацию одного из межсоединения трассировочного канала к третьему входу логического блока LUT-таблицы (in_3_lut_t или in_3_lut_b). С помощью верхнего и нижнего мультиплексора выход логического блока может быть подключен обратно на третий вход LUT-таблицы верхнего или нижнего логического блока.

Описание блока Crossbar на языке М-файлов демонстрирует пример 3. Блок Crossbar выполняет функцию демультиплексора. Выходы которого объединяются с межсоединениями трассировочного канала с использованием функции bitor (ИЛИ) с разрешением по выходу (En). На выходах демультиплексора предварительно устанавливаются нули, сигнал En=1 разрешает подключение выхода логического блока к трассировочному каналу. Пример 4 демонстрирует автоматически сгенерированный код языка VHDL блока Crossbar полученный с помощью Simulink HDL Coder.

На рис.3.29 показан соединительный блок С1, который осуществляет подключение межсоединения из вертикального трассировочного канала ко второму входу LUT-таблицы, левому или к правому соседнему (in_2_lut_l или in_2_lut_r). Пример 5 демонстрирует М-файл мультиплексора 4 в 1 входящего в состав соединительного блока С1. Подключение сигналов ко второму in_2_lut_l и третьему in_3_lut_t входу LUT-таблицы осуществляется с помощью элемента задержки Unit Delay. На рис.3.30 показан маршрутизатор (S-блок) и код

языка VHDL мультиплексора 5 в 1 используемого в маршрутизаторе.

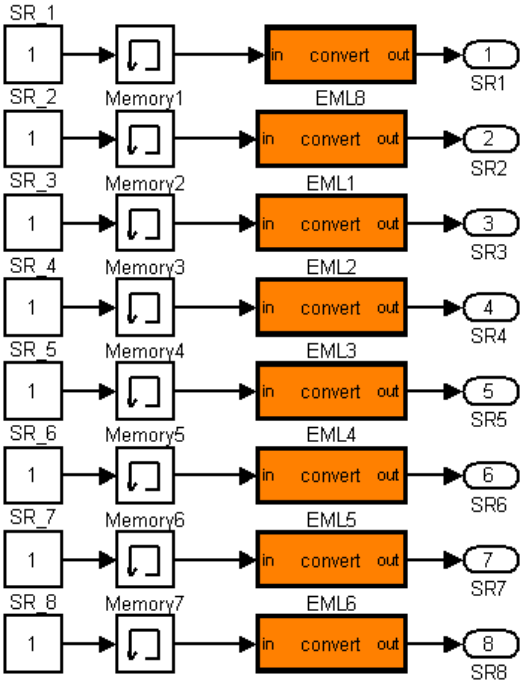


Рис.3.25. Фрагмент массива конфигурационной памяти

Пример 1. Конвертация типа double в тип ufix для генерации кода языка VHDL

```
function out  = convert(in)
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);
out = fi(in, 0, 1, 0, hdl_fm)
end
```

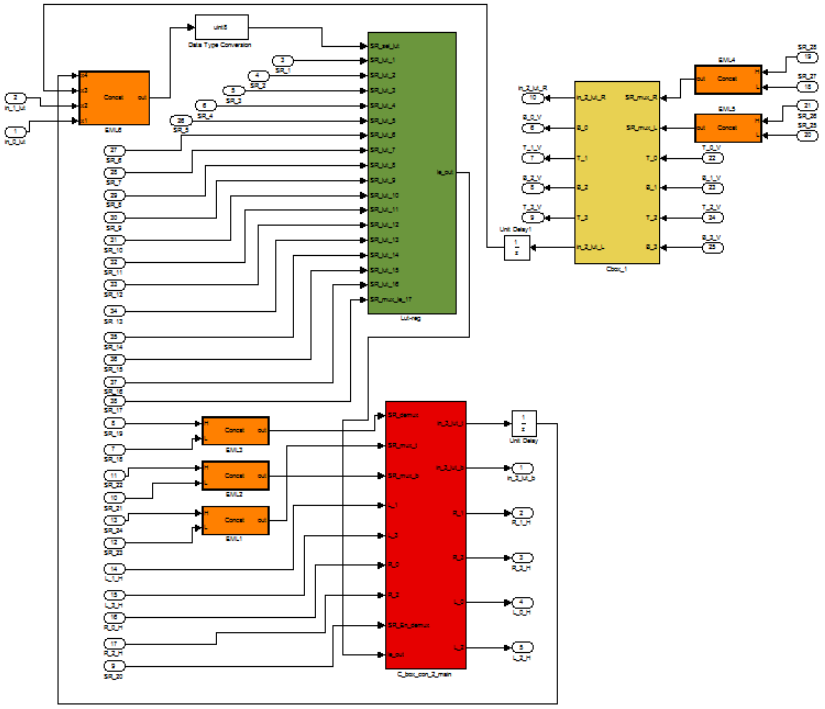


Рис.3.26. Логический блок и два соединительных блока C1 и C2

Пример 2. Создание двух разрядной шины

```
function out  = Concat(H, L)
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);
out = fi(bitconcat(fi(H, 0, 1, 0, hdl_fm), fi(L, 0, 1, 0, hdl_fm)));
end
```

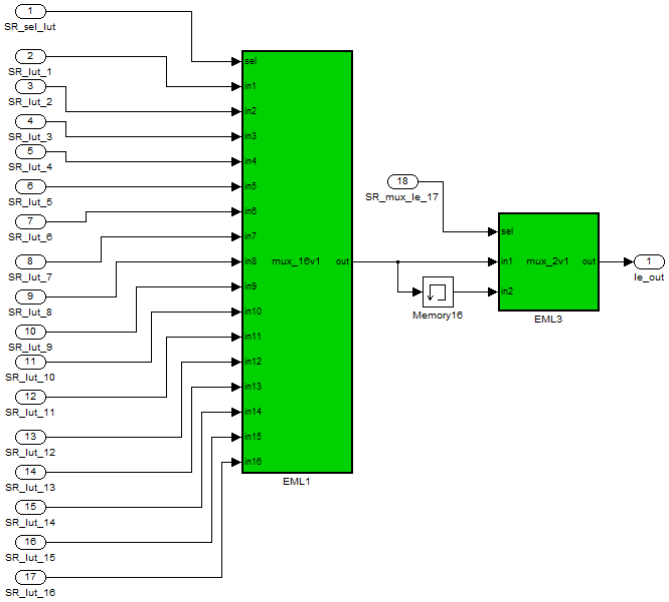


Рис.3.27. Логический блок плитки (мультиплексор 16 в 1, элемент памяти, мультиплексор)

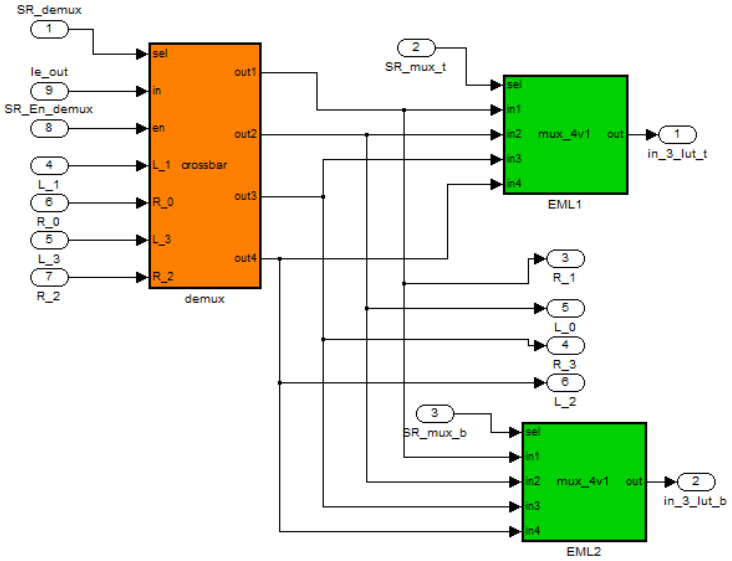


Рис.3.28. Соединительный блок C2

Пример 3. М-файл блока Crossbar входящего в состав соединительного блока C2

```
function [out1, out2, out3, out4] = crossbar(sel, in, en, L_1, R_0, L_3,
R_2)
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);

out1_de = fi(0, 0, 1, 0, hdl_fm);
out2_de = fi(0, 0, 1, 0, hdl_fm);
out3_de = fi(0, 0, 1, 0, hdl_fm);
out4_de = fi(0, 0, 1, 0, hdl_fm);

if  (logical(en))
switch (uint8(sel))
            case 0,
                out1_de = fi(in, 0, 1, 0, hdl_fm);
            case 1,
                out2_de = fi(in, 0, 1, 0, hdl_fm);
            case 2,
                out3_de = fi(in, 0, 1, 0, hdl_fm);
            case 3,
                out4_de = fi(in, 0, 1, 0, hdl_fm);

end
end
                out1 = fi(bitor(out1_de,L_1), 0, 1, 0, hdl_fm);
                out2 = fi(bitor(out2_de,R_0), 0, 1, 0, hdl_fm);
                out3 = fi(bitor(out3_de,L_3), 0, 1, 0, hdl_fm);
                out4 = fi(bitor(out4_de,R_2), 0, 1, 0, hdl_fm);
end
```

Пример 4. Код языка VHDL блока Crossbar входящего в состав соединительного блока C2

```
ENTITY demux IS
   PORT (
      sel : IN std_logic_vector(1 DOWNTO 0);
      in_rsvd : IN std_logic;
      en : IN std_logic;
      L_1 : IN std_logic;
      R_0 : IN std_logic;
      L_3 : IN std_logic;
      R_2 : IN std_logic;
      out1 : OUT std_logic;
      out2 : OUT std_logic;
      out3 : OUT std_logic;
      out4 : OUT std_logic);
END demux;
ARCHITECTURE rtl OF demux IS
BEGIN
   demux : PROCESS (sel, in_rsvd, en, L_1, R_0, L_3, R_2)
      VARIABLE out1_de : std_logic;
      VARIABLE out2_de : std_logic;
      VARIABLE out3_de : std_logic;
      VARIABLE out4_de : std_logic;
      VARIABLE c_uint : std_logic;
      VARIABLE b_c_uint : std_logic;
      VARIABLE c_c_uint : std_logic;
      VARIABLE d_c_uint : std_logic;
   BEGIN
      out1_de := '0';
      out2_de := '0';
      out3_de := '0';
      out4_de := '0';

      IF en /= '0' THEN

         CASE sel IS
            WHEN "00" =>
```

```
                out1_de := in_rsvd;
            WHEN "01" =>
                out2_de := in_rsvd;
            WHEN "10" =>
                out3_de := in_rsvd;
            WHEN "11" =>
                out4_de := in_rsvd;
            WHEN OTHERS =>
                NULL;
        END CASE;
    END IF;
    c_uint := out1_de OR L_1;
    out1 <= c_uint;
    b_c_uint := out2_de OR R_0;
    out2 <= b_c_uint;
    c_c_uint := out3_de OR L_3;
    out3 <= c_c_uint;
    d_c_uint := out4_de OR R_2;
    out4 <= d_c_uint;
  END PROCESS demux;
END rtl;
```

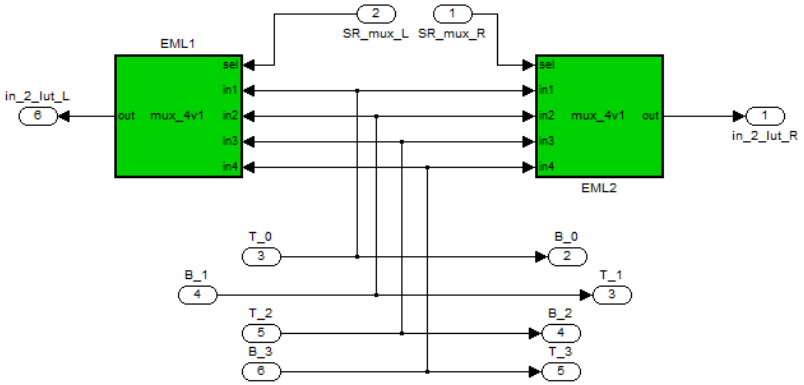


Рис.3.29. Соединительный блок C1. Подключение МС из вертикального трассировочного канала ко второму входу LUT-таблицы (in_2_lut_l или in_2_lut_r)

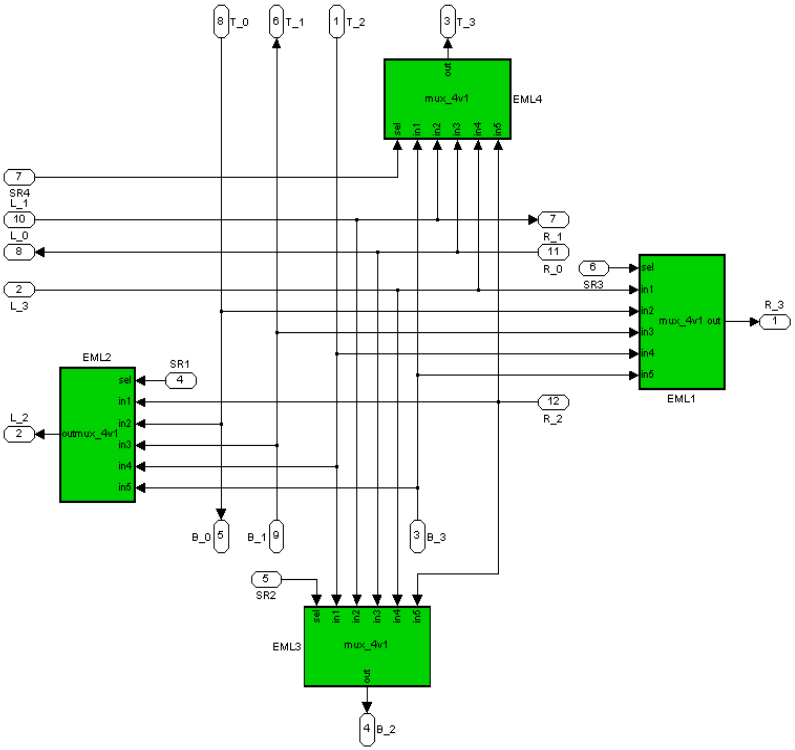


Рис.3.30. Маршрутизатор трассировочных ресурсов

Пример 5. М-файл мультиплексора 4 в 1 входящего в состав соединительного блока C1

```
function out  = mux_4v1(sel, in1, in2, in3, in4)
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);

out = fi(0, 0, 1, 0, hdl_fm);
switch (uint8(sel))
        case 0,
```

```
            out = fi(in1, 0, 1, 0, hdl_fm);
        case 1,
            out = fi(in2, 0, 1, 0, hdl_fm);
        case 2,
            out = fi(in3, 0, 1, 0, hdl_fm);
        case 3,
            out = fi(in4, 0, 1, 0, hdl_fm);
end
```

Пример.6. Код языка VHDL дешифратора 5 в 1 используемого в маршрутизаторе

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY EML1_block IS
   PORT (
      sel : IN std_logic_vector(2 DOWNTO 0);
      in1 : IN std_logic;
      in2 : IN std_logic;
      in3 : IN std_logic;
      in4 : IN std_logic;
      in5 : IN std_logic;
      out_rsvd : OUT std_logic);
END EML1_block;

ARCHITECTURE rtl OF EML1_block IS

BEGIN
   EML1_block : PROCESS (sel, in1, in2, in3, in4, in5)
   BEGIN
      out_rsvd <= '0';
      CASE sel IS
         WHEN "000" =>
            out_rsvd <= in1;
         WHEN "001" =>
            out_rsvd <= in2;
         WHEN "010" =>
```

```
            out_rsvd <= in3;
          WHEN "011" =>
            out_rsvd <= in4;
          WHEN "100" =>
            out_rsvd <= in5;
          WHEN OTHERS =>
            NULL;
        END CASE;
    END PROCESS EML1_block;
END rtl;
```

Продемонстрирована возможность использования системы визуально-иммитационного моделирования Matlab/Simulink с приложением Simulink HDL Coder для разработки архитектуры академической ПЛИС типа ППВМ с одноуровневой структурой межсоединений и технологией соединений single-driver на уровне системы. Разработанная архитектура ПЛИС обеспечивает однотипность трассировочность ресурсов по всей площади кристалла.

4. ПРОЕКТИРОВАНИЕ МИКРОПРОЦЕССОРНЫХ ЯДЕР ДЛЯ РЕАЛИЗАЦИИ В БАЗИСЕ ПЛИС

4.1. Проектирование учебного процессора для реализации в базисе ПЛИС с помощью конечного автомата

Микропроцессорные ядра представляют важный класс вычислительных заготовок, так как от их качеств, в основном, зависят основные технические и потребительские свойства систем на кристалле. Эти заготовки различаются по степени гибкости настройки под условия потребителя как программные (“мягкие” описанные на языке HDL), жесткие (логическая схема) и аппаратные (“твердые” маски под определенную технологию). Программные заготовки можно легко подстраивать к условиям нового проекта, обладают высоким быстродействием и они независимы от технологии. Их реализация в ПЛИС (например, 8-разрядное микропроцессорное ядро PicoBlaze для реализации в базисе ПЛИС семейств Spartan и Virtex) позволяет ускорить процесс разработки микропроцессорных систем. Наиболее важными потребительскими свойствами вычислительных заготовок процессоров являются: повторяемость, быстродействие, аппаратурные затраты.

Путем несложной перенастройки “мягкой” заготовки можно получить ряд модификаций микроконтроллера с различным сочетанием объема памяти, периферийных устройств, источников прерывания и т.п. Такой процессор можно реализовать в ПЛИС различных фирм. Описание модели на VHDL позволяет не только сделать ее перенастраиваемой и независимой от технологии, но и выполнять ее моделирование и синтез на симуляторах и средствах синтеза различных фирм. На рис.4.1 показано отображение процессора в ПЛИС.

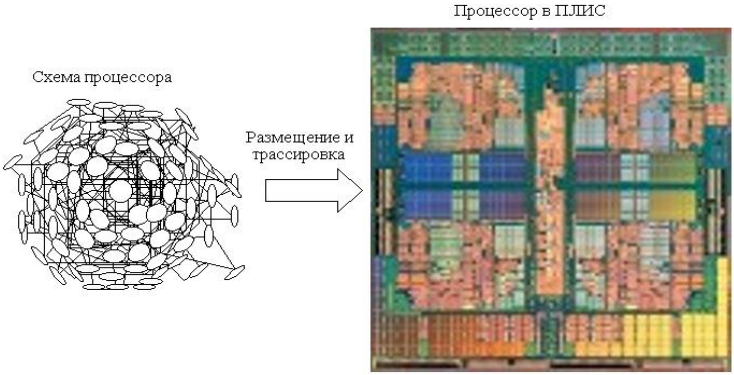


Рис.4.1. Отображение схемы процессора в базисе ПЛИС

Рассмотрим систему команд синхронного процессора, реализованного с помощью конечного автомата, с циклом работы в два такта. При разработке системы команд процессора, используется слабое кодирование. В табл.4.1 представлена система команд процессора с синхронной архитектурой.

Процессор способен работать с синхронным ОЗУ. Это обеспечивается использованием оператора case, который используется в ветви оператора if при детектировании атрибута переднего фронта синхроимпульса clk и позволяет организовать цикл работы в два такта.

Реализуем процессор в ПЛИС фирмы Altera APEX20KE как с асинхронным ПЗУ (мегафункция LPM_ROM). На рис.4.2 показана тестовая схема управляющего автомата процессора в графическом редакторе САПР ПЛИС Quartus II версии 2.0. На рис.4.3 показано содержимое конфигурационного файла ПЗУ.

В описание процессора на языке VHDL добавлен асинхронный сброс регистров A, B и счетчика команд на регистре ip (врезка 1). Декодирование переменной-селектора cmd осуществляется с помощью оператора case. Оператор преобразования типов conv_integer(cmd) переводит вектор в десятичное число отдельных кодов.

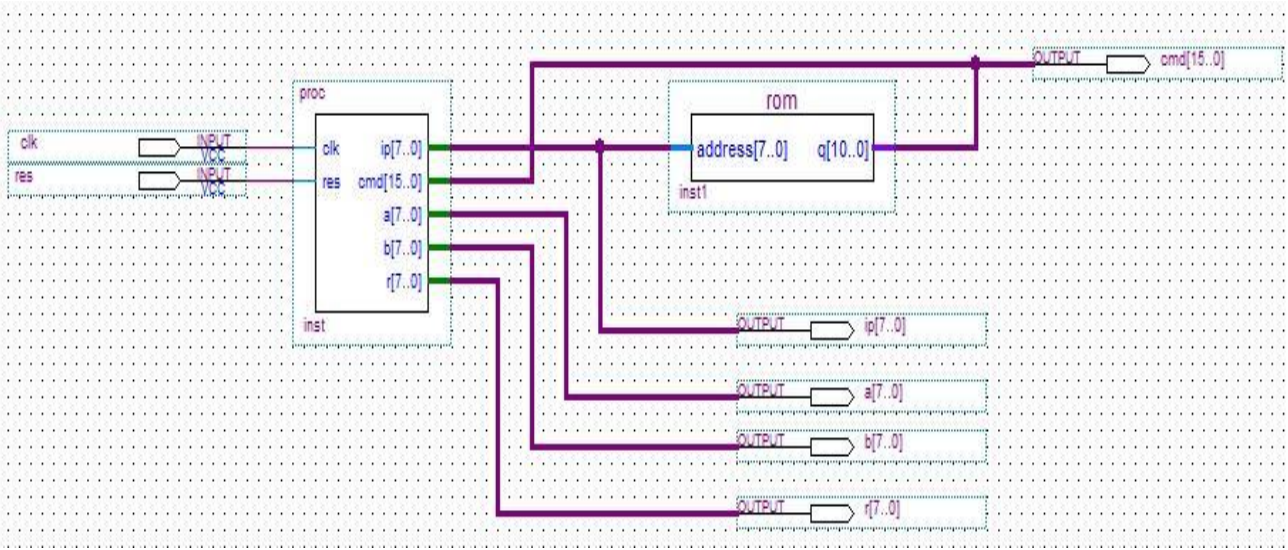


Рис.4.2. Тестовая схема микропроцессорного ядра в графическом редакторе САПР ПЛИС Quartus II версии 2.0 с использованием управляющего автомата с циклом работы в два такта на языке VHDL и асинхронного ПЗУ (мегафункция LPM_ROM)

Таблица 4.1

Система команд процессора с синхронной архитектурой

Код операции	Мнемоника	Описание
0	NOP	Нет операции
01xxH	JMP	Безусловный переход по адресу, заданному младшим байтом команды
02xxH	JMPZ	Переход по адресу, заданному младшим байтом команды, если содержимое регистра A равно нулю
03xxH	CALL	Вызов подпрограммы по адресу, заданному младшим байтом команды
04xxH	MOV A,xx	Непосредственная загрузка в регистр A значения, заданного младшим байтом команды
05xxH	MOV B,xx	Непосредственная загрузка в регистр B значения, заданного младшим байтом команды
0600H	RET	Возврат из подпрограммы
0601H	MOV A,B	Загрузка в регистр A значения, содержащегося в регистре B
0602H	MOV B,A	Загрузка в регистр B значения, содержащегося в регистре A
0603H	XCHG A,B	Обмен местами значений в регистрах A и B
0604H	ADD A,B	Сложение значений в регистрах A и B, результат помещается в A
0605H	SUB A,B	Вычитание значений в регистрах A и B, результат помещается в A
0606H	AND A,B	Побитное логическое И значений в регистрах A и B, результат помещается в A
0607H	OR A,B	Побитное логическое ИЛИ значений в регистрах A и B, результат помещается в A
0608H	XOR A,B	Побитное логическое ИСКЛЮЧАЮЩЕЕ ИЛИ значений в регистрах A и B, результат помещается в A

При каждом допустимом значении кода команды, происходят различные действия, которые состоят в назначении регистрам новых значений в соответствии с описанием команд. Счетчик команд (регистр) не обновляется автоматически, поэтому в каждом варианте кода команды, присваивание счетчику нового значения, указывается явно. Процессор ограничивается двумя регистрами общего назначения (A и B). Процессор имеет указатель инструкций ip и регистр r (стек), для хранения адреса, с которого произошел вызов подпрограммы, поддерживает минимальный набор команд: команда пересылки “регистр - регистр”; команды непосредственной загрузки; команда безусловного перехода к новому адресу; команды перехода по условию; набор арифметико-логических операций. Пример 1 показывает управляющий автомат на языке VHDL.

Наиболее сложными являются команды передачи управления JMP и JMPZ, и команда обращения к подпрограммам CALL и команда возврата из подпрограммы RET. Временные диаграммы на рис.4.4 демонстрируют принцип работы управляющего автомата с асинхронным ПЗУ при отработке команд CALL (0305H) и RET (0600H). При нормальной последовательности работы процессора отрабатываются регистровые команды. Последовательно загружаются регистры A и B. В регистр A загружается число 1D (1H), а в регистр B, число 17D (11H). По команде 0305H происходит запись содержимого счетчика команд ip в регистр r (2D) и загрузка в счетчик команд числа 5D. Таким образом, процессор начнет выполнять подпрограмму хранящуюся в ПЗУ с адреса 5H. По указанному адресу извлекается регистровая команда 0403H. Происходит загрузка в регистр A числа 3H, командой 0404P загрузка числа 4H. Следющей командой с кодом 0604H произойдет сложение содержимых регистров с сохранением результата в регистре A (число 21D). Далее будут отработаны команды 0406H и 0407H. По команде

возврата из подпрограммы 600H произойдет изменение содержимого счетчика с 10D на 2D+1D, т.е. на 3D.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity proc is
port (ip: inout std_logic_vector(7 downto 0);
        cmd: inout std_logic_vector(15 downto 0);
        clk,res: in std_logic;
        a: inout std_logic_vector(7 downto 0);
        b,r: inout std_logic_vector(7 downto 0));
end proc;
architecture a of proc is
signal stage: std_logic;
begin
process(clk)
begin
if (res = '1') then
                a <="00000000";
                b <="00000000";
                ip <="00000000";
elsif clk'event and clk='1' then
case stage is
when '0'=> stage<='1';
when others=> stage<='0';
case conv_integer(cmd) is
when 0=> ip <= ip+1;
when 256 to 511 =>ip<=cmd(7 downto 0);
when 512 to 767 =>if conv_integer(a)=0
                                then ip<=cmd(7 downto 0);
                                else ip<=ip+1;
                                end if;
when 768 to 1023 =>r<=ip; ip<=cmd(7 downto 0);
when 1024 to 1279 => a<=cmd(7 downto 0); ip<=ip+1;
when 1280 to 1535 => b<=cmd(7 downto 0); ip<=ip+1;
```

```
when 1536 =>ip<=r+1;
when 1537=>a<=b; ip<=ip+1;
when 1538=>b<=a; ip<=ip+1;
when 1539=>a<=b;b<=a; ip<=ip+1;
when 1540=>a<=a+b; ip<=ip+1;
when 1541=>a<=a-b; ip<=ip+1;
when 1542=>a<=a and b; ip<=ip+1;
when 1543=>a<=a or b; ip<=ip+1;
when 1544=>a<=a xor b; ip<=ip+1;
when 1545=>a<=a-1; ip<=ip+1;
when others=>ip<=ip+1;
end case;
end case;
end if;
end process;
end a;
```

Пример 1. Управляющий автомат на языке VHDL

Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	0401	0511	0305	0512	0402	0403	0404	0604
08	0406	0407	0600	0000	0000	0000	0000	0000

Рис.4.3. Файл конфигурации ПЗУ для тестирования команды обращения к подпрограммам CALL и возврата RET

Рассмотрим вариант процессора без использования управляющего автомата. С этой целью регистровые команды 04xxH, 05xxH, 0601H-0609H предлагается реализовать на тактируемом дешифраторе (пример 2), выполняющим функцию арифметически-логического устройства (АЛУ), а команды передачи управления JMP, JMPZ и обращения к подпрограммам с кодами 01xxH-03xxH, 0600H на 8-ми разрядном суммирующем счетчике адресов памяти команд (пример 3), тактируемым фронтом синхросигнала (рис.4.5).

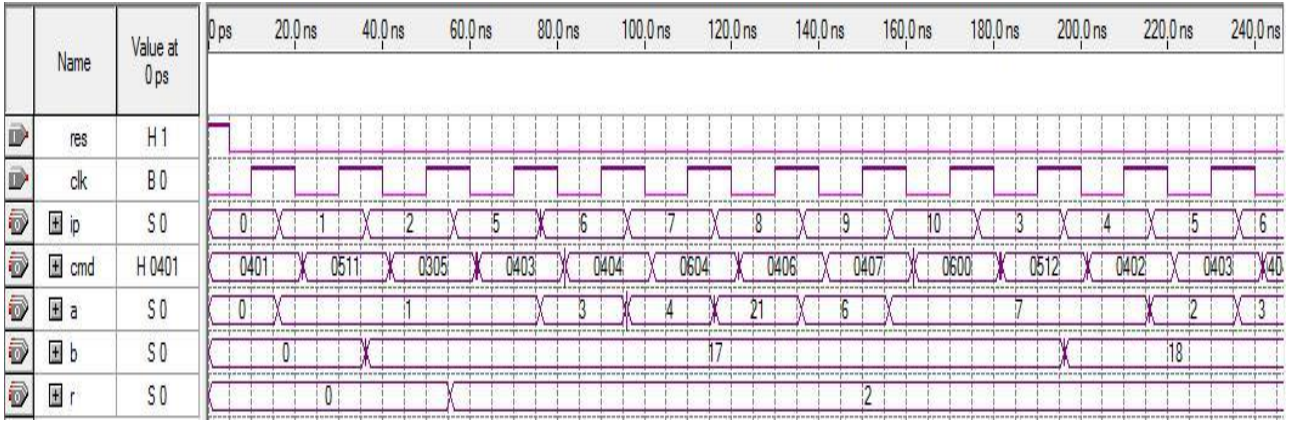


Рис.4.4. Временная диаграмма работы процессора, с использованием управляющего автомата с циклом работы в два такта и мегафункции асинхронного ПЗУ. Отрабатываются регистровые команды и команда вызова подпрограммы с кодом 0305H (CALL) и команда возврата из подпрограммы 0600H (RET)

Счетчик содержит асинхронный сброс Reset. Активным является сигнал высокого уровня. Во вложенных ветвях оператора if происходит проверка условий и синхронная загрузка счетчика команд. Счетчик команд ip и регистр r при инициализации системы по сигналу Reset устанавливаются в состояние 0, после чего производит счет адресов памяти программ хранимых в ПЗУ. Регистр r выполняет функцию стека, в который заносится прежнее состояние счетчика команд. Из шины cmd[10..0], для счетчика команд выделяется поле cop[10..8] и поле data[7..0]. Поле cop означает код операции, который используется для идентификации команд JMP, JMPZ и CALL. Для команды RET поле cop не формируется, а задается полный адрес на шине cmd "11000000000", это связано с тем, что 8 младших бит для команд JMP, JMPZ и CALL могут принимать любые значения, а для команды RET только указанный. Поле data содержит 8-разрядный операнд, который загружается в регистр команд.

ПЗУ реализовано с использованием мегафункции LPM_ROM. В табл.4.2 представлены сведения по общему числу задействованный ресурсов ПЛИС. В обоих случаях проект отображается в ПЛИС APEX20KE (EP20K30ETC144). На рис.4.6 показано тестирование процессора.

Таблица 4.2

Общие сведения по числу задействованных ресурсов ПЛИС APEX20KE

Номер проекта	Общее число логических элементов	Общее число используемых ESB-бит памяти	D-триггеров
С использованием управляющего автомата	198/1200 (16 %)	2816/24576 (11 %)	32
Вариант с асинхронным ПЗУ	164/1200 (13 %)	2816/24576 (11 %)	32

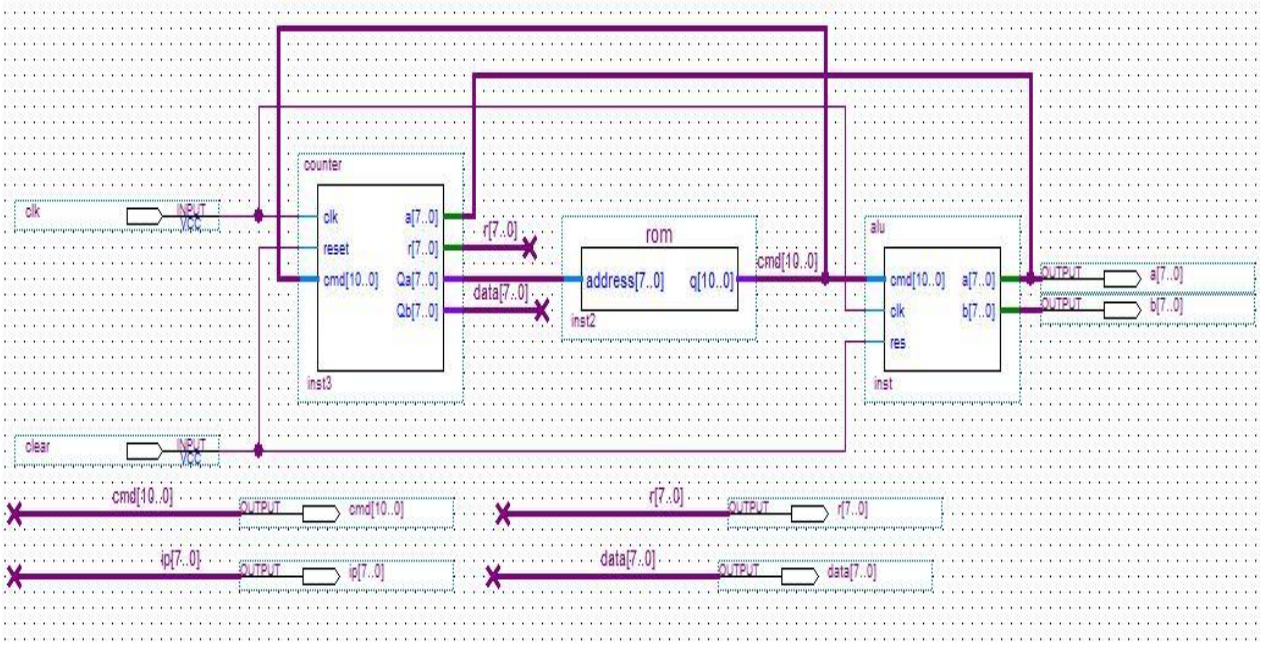


Рис.4.5. Тестовая схема микропроцессорного ядра без использования управляющего автомата с асинхронным ПЗУ (мегафункция LPM_ROM) в графическом редакторе САПР ПЛИС Quartus II версии 2.0

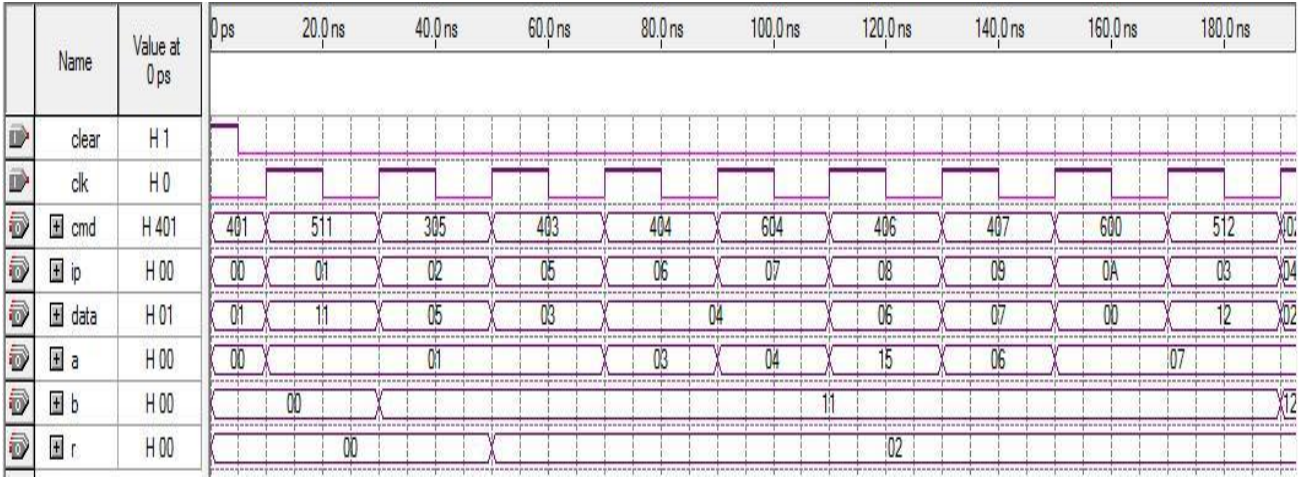


Рис.4.6. Временные диаграммы работы микропроцессорного ядра без использования управляющего автомата с мегафункцией асинхронного ПЗУ

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

ENTITY alu IS
PORT
(cmd              : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
clk,res           : IN STD_LOGIC;
a,b               : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END alu;
ARCHITECTURE a OF alu IS
signal regA,regB: std_logic_vector(7 downto 0);
BEGIN
PROCESS (clk,res)
BEGIN
        if (RES = '1') then
                        regA <="00000000";
                        regB <="00000000";
        elsif (clk'event and clk='1') then
        case conv_integer(cmd) is
        when 1024 to 1279 => regA<=cmd(7 downto 0);
        when 1280 to 1535 => regB<=cmd(7 downto 0);
        when 1537=>regA<=regB;
        when 1538=>regB<=regA;
        when 1539=>regA<=regB; regB<=regA;
        when 1540=>regA<=regA+regB;
        when 1541=>regA<=regA-regB;
        when 1542=>regA<=regA and regB;
        when 1543=>regA<=regA or regB;
        when 1544=>regA<=regA xor regB;
        when 1545=>regA<=regA-1;
        when others=> a<=regA; b<=regB;
        end case;
end if;
a<=regA;
b<=regB;
```

```
END PROCESS;
END a;
```

Пример 2. Арифметически-логическое устройство процессора

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

ENTITY counter IS
PORT(
clk       : IN  STD_LOGIC;
reset     : IN       STD_LOGIC;
cmd       : IN  STD_LOGIC_VECTOR(10 downto 0);
a          : INOUT  STD_LOGIC_VECTOR(7 downto 0);
r          : INOUT STD_LOGIC_VECTOR(7 downto 0);
Qa,Qb   : OUT STD_LOGIC_VECTOR(7 downto 0));
END counter;
ARCHITECTURE a OF counter IS
SIGNAL pci,pc,pcplus,data,regA: STD_LOGIC_VECTOR(7 downto 0);
SIGNAL cop: STD_LOGIC_VECTOR(2 downto 0);
BEGIN
regA<=a;
cop<=cmd(10 downto 8);
data<=cmd(7 downto 0);
process(clk,reset,cop,data,a)
begin
          if         (reset = '1') then
                                        pci <=(others=>'0');
                                        r <=(others=>'0');
          elsif (clk'event and clk='1') then
          if
          cmd="001"  then pci<=data; --JMP H1
          else if
(cop="010" and conv_integer(regA)=0) then pci<=data; --JMPZ H2
          else if
                    cop="011"  then r<=pci; pci<=data;--CALL H3
```

```
else if
        cmd="11000000000" then pci<=r+1; --RET H6
else pci<=pci+1;
end if; end if; end if; end if; end if;
END PROCESS;
Qa <= pci;
Qb <=data;
a<=regA;
END a;
```

Пример 3. Счетчик адресов памяти команд процессора с асинхронным сбросом

Рассмотрим вариант реализации проектируемого процессора с использованием асинхронного ОЗУ (рис.4.7). Для этого воспользуемся мегафункцией LPM_RAM_IO. Для того, что бы ОЗУ выполняло функцию ПЗУ, необходимо сигнал разрешения записи we “посадить” на землю, т.к. активным является сигнал высокого уровня, а сигнал разрешения вывода outenab подключить к питанию. На рис.4.8 показан файл конфигурации ОЗУ для тестирования команды JMPZ с кодом 0205H. По команде 0205H осуществляется переход по адресу, заданному младшим байтом команды (на адрес в ОЗУ под номером 5, где хранится команда 0403H), если содержимое регистра A равно нулю. Чтобы содержимое регистра A оказалось равным нулю, необходимо воспользоваться регистровыми командами 0405H и 0505H, для загрузки в регистры A и B числа 5, а затем с помощью команды 0605H (SUB A,B) осуществить операцию A-B (рис.4.9).

Данный вариант процессора, реализованный в базисе ПЛИС, можно отнести к классу RISC-процессоров, у которых все команды выполняются за один такт синхрочастоты.

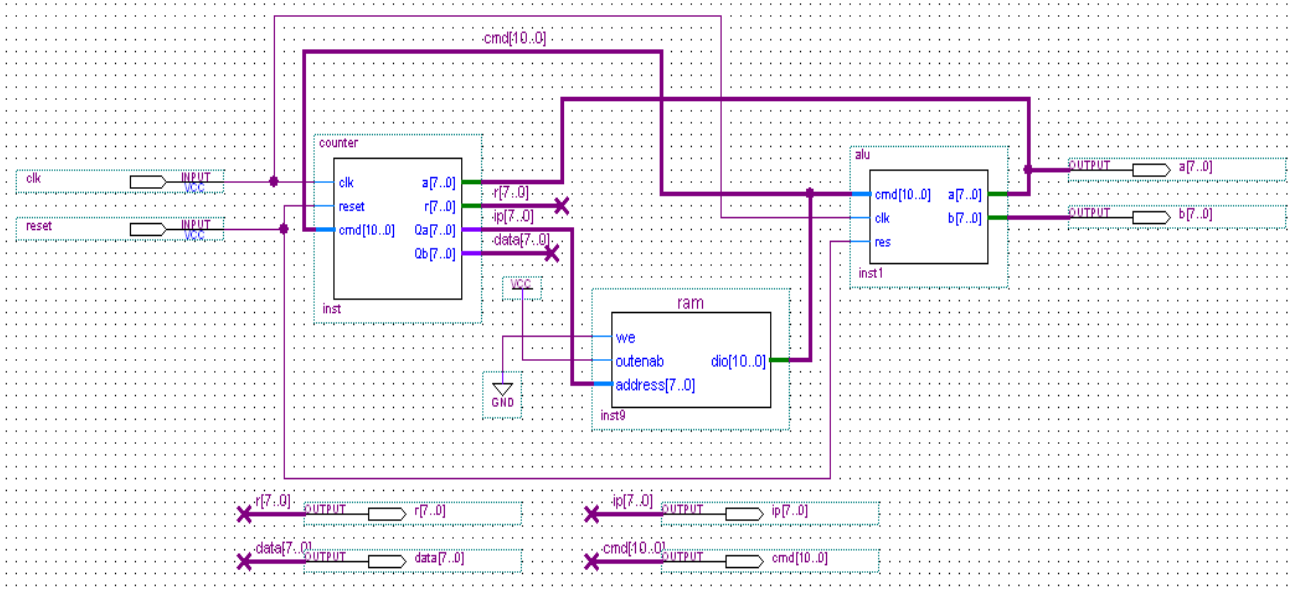


Рис.4.7. Тестовая схема процессора без использования управляющего автомата с асинхронным ОЗУ (мегафункция LPM_RAM_IO) в графическом редакторе САПР ПЛИС Quartus II версии 2.0

Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	0405	0505	0605	0205	0402	0403	0000	0000

Рис.4.8. Файл конфигурации ОЗУ для тестирования команды JMPZ

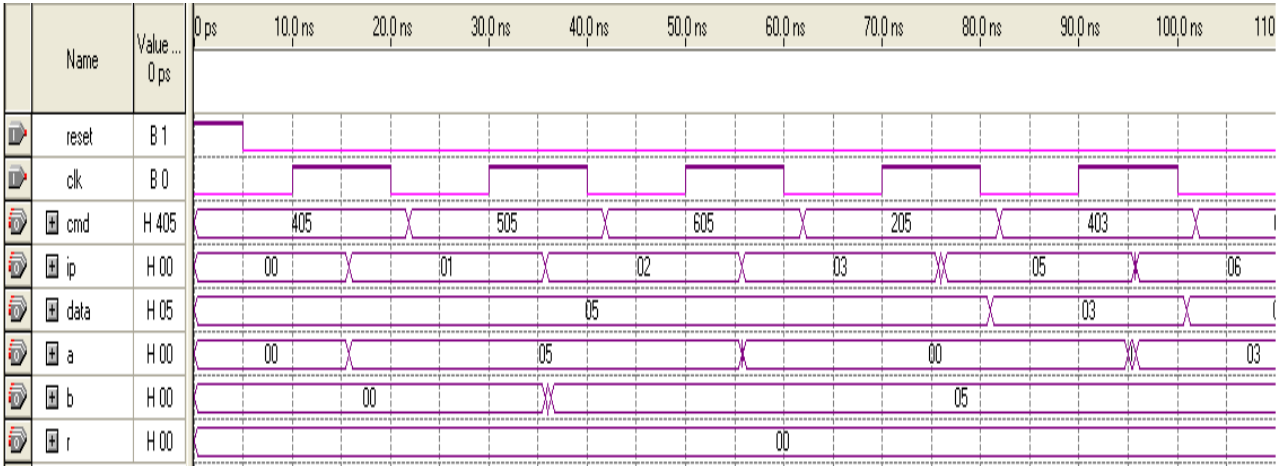


Рис.4.9. Временные диаграммы работы микропроцессорного ядра с мегафункцией асинхронного ОЗУ. Тестирование команды условного перехода JMPZ

Процессор может быть модифицирован путем наращивания блока регистров, добавлением блока управления прерываний, добавления блока управления ввода/вывода и других функциональных блоков. Вариант с асинхронным ПЗУ реализованный в ПЛИС APEX20KE EP20K30ETC144-1 занимает всего лишь 13 % от общего числа логических элементов и способен работать на частоте 60 МГЦ.

4.2. Использование различных типов памяти при проектировании учебного микропроцессорного ядра для реализации в базисе ПЛИС

Предлагается повторно переработать проект микропроцессорного ядра из раздела 4.1 в базисе ПЛИС APEX20KE и Stratix III компании Altera с использованием САПР ПЛИС Quartus II версии 8.1, с целью изучения особенностей использования различных видов памяти. В качестве микропроцессорного ядра используется автомат с циклом работы в два такта.

Особенности современной цифровой микроэлектроники обуславливают преимущественное использование синхронных интерфейсов устройств разного типа, в том числе и памяти. Попытки реализовывать асинхронный интерфейс могут в ряде случаев получать схемы с различными задержками распространения сигналов («гонки фронтов», «перекосы»), которые не выявляются средствами моделирования, но оказывают существенное негативное влияние на характеристики проекта, загружаемого в ПЛИС, вплоть до неработоспособности схемы или перемежающихся неисправностей.

Семейство ПЛИС Stratix III позволяет реализовать на одном кристалле булевы логические функций с помощью адаптивных логических блоков, блоки памяти, при этом

память может быть реализована без затрат основной логики и блоки цифровой обработки сигналов. ПЛИС APEX20KE содержит лишь встроенные блоки памяти (встроенное ОЗУ/ПЗУ).

ПЛИС Stratix имеют структуру памяти TriMatrix, составленную из встроенных блоков памяти RAM трех видов: блоки MLAB (Memory LAB, емкость блока 320 бит, блок может быть представлен как простое двух портовое ОЗУ); блоки M9K (емкость блока 9216 бит); блоки M144K (емкость 147456 бит). Так же в ПЛИС Stratix III встроена дополнительная собственная память (встроенное реконфигурируемое ОЗУ). Блоки MLAB используются для реализации сдвиговых регистров, буферов FIFO, линий задержек для цифровых фильтров и др. Блоки M9K используются как блоки памяти общего назначения. Блоки M144K используются для хранения исполняемого кода синтезируемых процессорных ядер, для реализации буферов большого объема в задах видеообработки сигналов. Все блоки памяти TriMatrix поддерживают синхронный режим работы. Блоки M9K и M144K в ПЛИС Stratix не поддерживают асинхронную память, а блок MLAB поддерживает только асинхронный режим чтения данных. Каждый из блоков памяти с помощью мегафункции altsyncram может быть сконфигурирован как: RAM: 1 PORT, RAM: 2 PORT, RAM: 3 PORT, ROM: 1 PORT, ROM: 2 PORT, shift register (RAM-based), FIFO.

Рассмотрим два варианта построения микропроцессорного ядра: с асинхронным ПЗУ (рис.4.10, а) с использованием устаревших серий ПЛИС, например, ПЛИС APEX20KE и синхронным ПЗУ (рис.4.10, б) с использованием ПЛИС Stratix III. Вариант микропроцессорного ядра с мегафункцией асинхронного ПЗУ (LPM_ROM) рассмотрен в разделе 4.1 с использованием САПР ПЛИС Quartus II версии 2.0. В обоих случаях емкость ПЗУ 256 слов на 16 бит, т.е.

входная адресная шина 8-ми (address[7..0]), а выходная шина данных (q[15..0]) 16-ти разрядная. Файл прошивки ПЗУ приведен ниже (mif – файл, пример 1).

```
-- Quartus II generated Memory Initialization File (.mif)
WIDTH=16;
DEPTH=256;
ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;
CONTENT BEGIN
        000 :  0401;
        001 :  0511;
        002 :  0305;
        003 :  0512;
        004 :  0402;
        005 :  0403;
        006 :  0404;
        007 :  0604;
        008 :  0406;
        009 :  0407;
        00A :  0600;
        [00B..0FF] :  0000;
END;
```

Пример 1. Файл прошивки ПЗУ

Мегафункция LPM_ROM для ПЛИС Stratix III работает в режиме совместимости. Поэтому в случае повторной переработки проектов выполненных на устаревших сериях ПЛИС, например на APEX20KE, при смене серии ПЛИС на Stratix III (рис.4.11, а, галочка Match project/default снята) возникает предупреждение, что мегафункция LPM_ROM будет сконфигурирована на синхронный режим работы. Возникнет предупреждение о рекомендации к использованию мегафункции altsyncram (ROM: 1 PORT). Фактически произойдет замещение мегафункции LPM_ROM на мегафункцию ROM: 1 PORT.

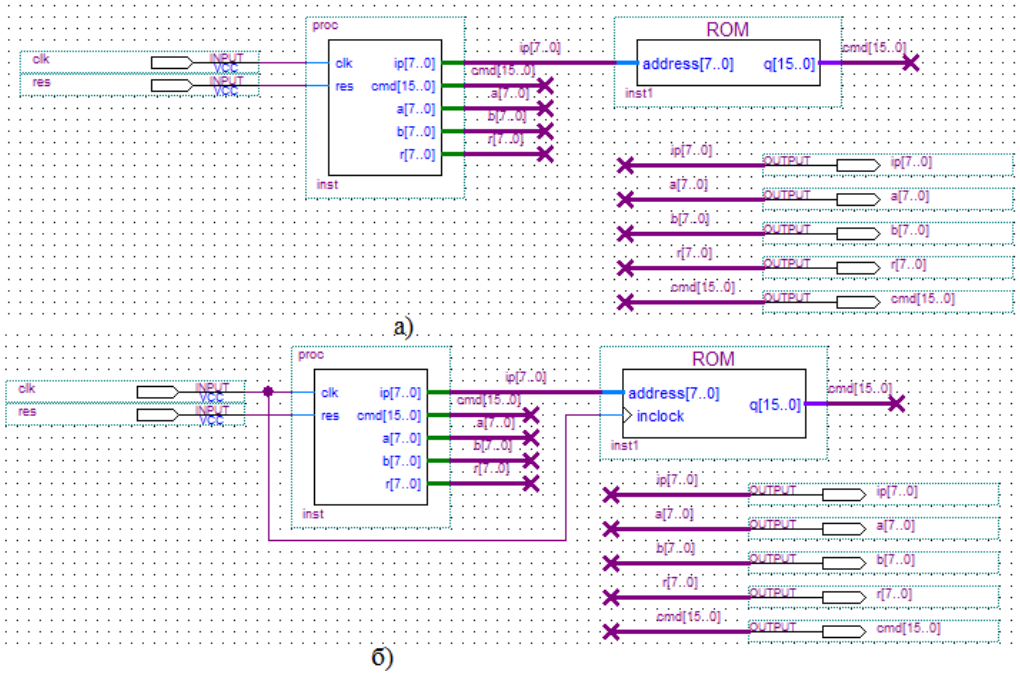
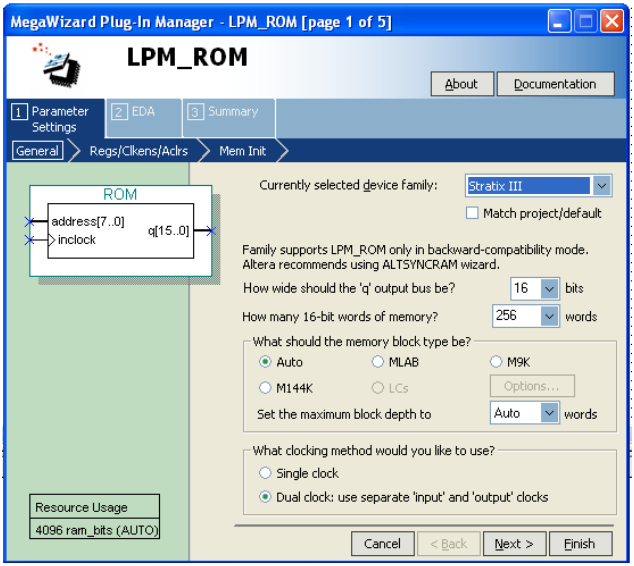
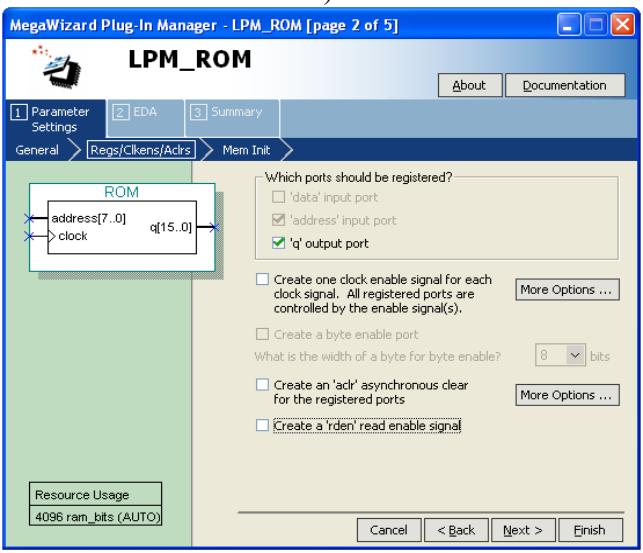


Рис.4.10. Микропроцессорное ядро с асинхронным ПЗУ в базисе ПЛИС APEX20KE (а) и синхронным (б) ПЗУ в базисе ПЛИС Stratix III (САПР ПЛИС Quartus II версии 8.1)



а)



б)

Рис.4.11. Менеджер по работе с мегафункцией ПЗУ LPM_ROM ПЛИС Quartus II версии 8.1 для ПЛИС Stratix III в режиме совместимости: а) шаг 1; б) шаг 2

При настройке мегафункции для ПЛИС на Stratix III пользователь должен самостоятельно выбрать один из типов блоков памяти или выбрать тип Auto. Для ПЛИС APEX20KE доступен только тип Auto (мегафункция LPM_ROM), что объясняется архитектурными особенностями данной ПЛИС, а именно встроенными блоками памяти. Если выбрать тип Auto для ПЛИС Stratix III, то по умолчанию доступны 4096 бит памяти (зависит от разрядности адресной и выходной шины данных).

Галочку с ‘address’ input port снять нельзя, т.е. адресный порт по умолчанию настраивается как регистерный (на условном обозначении появляется указатель динамического входа – треугольник, т.е. адресация к словам ПЗУ осуществляется по переднему фронту синхроимпульса inclock, а выходной порт q – асинхронный) (рис.4.11, б, мегафункции LPM_ROM для ПЛИС Stratix III, шаг 2). Это говорит о том что ПЗУ будет сконфигурированна на синхронный режим работы. По желанию, работу выходного порта q можно также синхронизовать сигналом outclock, таким образом можно реализовать режим двойного тактирования.

Использование внутренних триггеров адаптивных логических модулей в качестве памяти ПЗУ в мегафункции для ПЛИС на Stratix III недопустимо, поэтому опция LC (логические элементы, задействуется основная логика: таблицы перекодировок и внутренние триггеры логических элементов) не доступна. Опция LC доступна при разработке ОЗУ без предварительной конфигурации. В случае выбора опции MLAB и режима Auto доступны в качестве памяти 20 таблиц перекодировок (LUT-таблица, таблица перекодировок или логическая таблица, служит для реализации комбинационных функций и может быть упрощенно представлена как столбец ОЗУ с мультиплексором) + 7 MLAB + 24 триггера. При использовании одного из трех типов блоков памяти MLAB, M9K и M144K, дополнительный режим Auto

назначает максимально доступный к использованию объем памяти. Число используемых слов для каждого блока памяти может быть ограничено пользователем.

Что бы избежать данного предупреждения и повторно переработать проект с использованием мегафункции однопортового ПЗУ (ROM: 1 PORT) для ПЛИС Stratix III, необходимо удалить из проекта блок памяти ПЗУ созданный с помощью мегафункции LPM_ROM и заново с помощью мастера MegaWizard Plugin сгенерировать блок памяти (рис.4.12). Сменив предварительно в Assignments/Device серию ПЛИС APEX20KE на Stratix III.

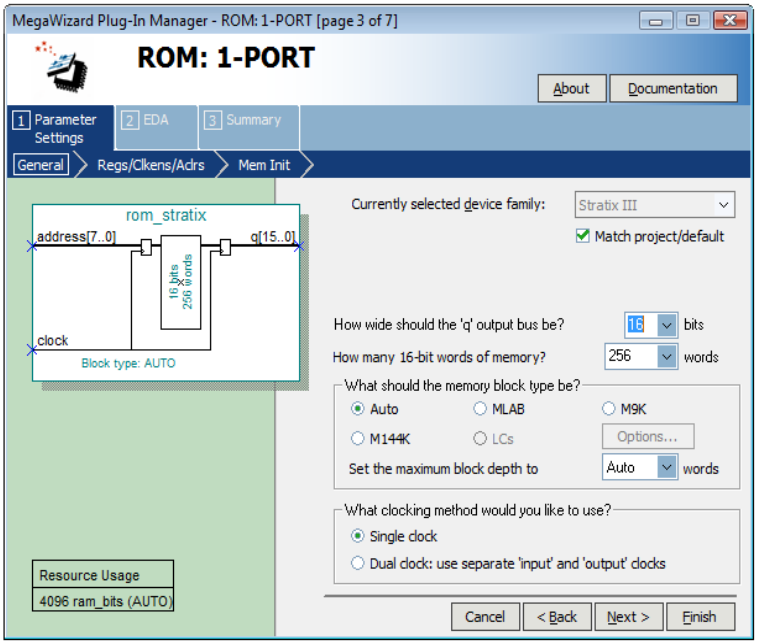


Рис.4.12. Менеджер по работе с мегафункцией однопортовой памяти ПЗУ LPM_ROM ПЛИС Quartus II версии 8.1 для ПЛИС Stratix III (ROM: 1 PORT)

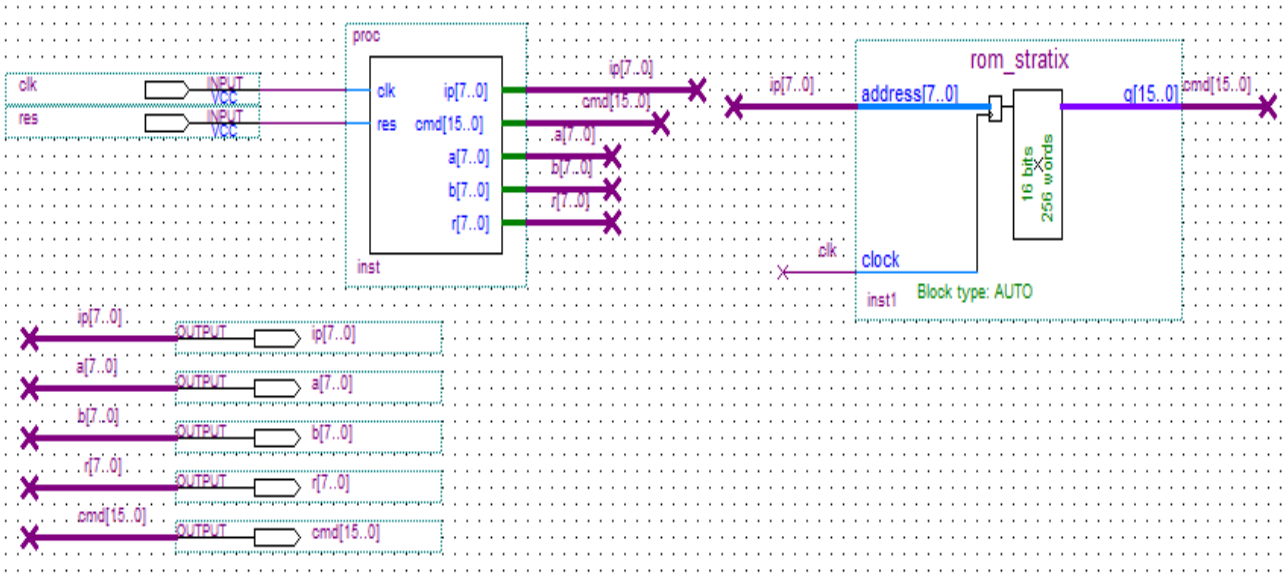


Рис.4.13. Микропроцессорное ядро с синхронным ПЗУ (синхронный режим адресации и асинхронный режим чтения) в базисе ПЛИС Stratix III созданное с помощью мегафункции ROM: 1 PORT САПР ПЛИС Quartus II версии 8.1

На рис.4.13 показано микропроцессорное ядро с синхронным ПЗУ в базисе ПЛИС Stratix III созданное с помощью мегафункции ROM: 1 PORT. Сравнивая рис.4.11 и рис.4.12, видим, что условное обозначение мегафункции ROM: 1 PORT несколько отличается от обозначения мегафункции LPM_ROM в режиме совместимости. Это означает, что все входы в память и выходы из нее защелкиваются в регистрах.

На рис.4.14 и рис.4.15 показано функциональное и временное моделирование работы микропроцессорного ядра с асинхронным ПЗУ в базисе ПЛИС APEX20KE (а) и синхронным (б) ПЗУ в базисе ПЛИС Stratix III с использованием мегафункции ПЗУ LPM_ROM САПР Quartus II версии 8.1.

Для того чтобы обеспечить переносимость проекта с одной серии ПЛИС на другую, без применения мегафункций, рассмотрим использование языка VHDL для проектирования ПЗУ. Пример 2 демонстрирует проектирование асинхронного ПЗУ на языке VHDL с использованием элементов поведенческого описания (используются абстрактные логические структуры, такие как циклы и процессы). На рис.4.16 показаны результаты функционального моделирования в базисе ПЛИС APEX20KE.

Пример 3 демонстрирует проектирование синхронного ПЗУ на языке VHDL, а на рис.4.17 показаны результаты функционального моделирования в базисе ПЛИС Stratix III. Проект микропроцессора с синхронным ПЗУ на языке VHDL, демонстрирует работоспособность, как на старых, так и на новых сериях ПЛИС фирмы Altera. Пример 4 показывает описание асинхронного ПЗУ с использованием элементов потокового описания (представление на уровне регистровых передач). Оператор case, обеспечивает параллельную обработку и используется для выбора одного варианта из нескольких в зависимости от условий.

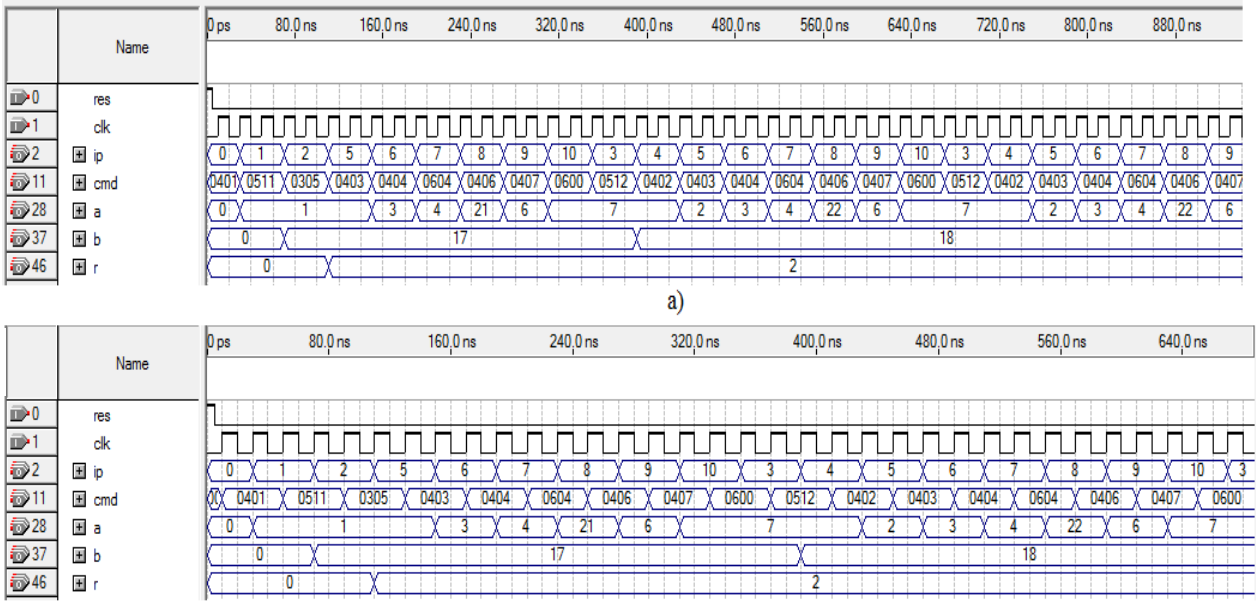


Рис.4.14. Функциональное моделирование работы микропроцессорного ядра с асинхронным ПЗУ в базисе ПЛИС APEX20KE (а) и синхронным (б) ПЗУ в базисе ПЛИС Stratix III с использованием мегафункции ПЗУ LPM_ROM

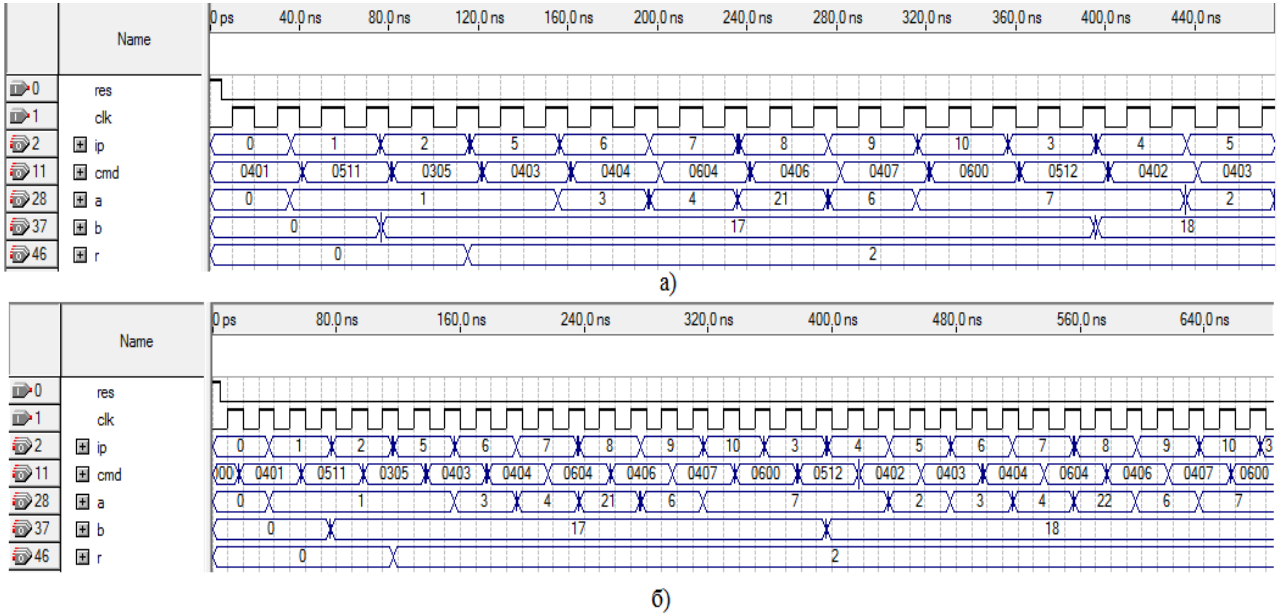


Рис.4.15. Временное моделирование работы микропроцессорного ядра с асинхронным ПЗУ в базисе ПЛИС APEX20KE (а) и синхронным (б) ПЗУ в базисе ПЛИС Stratix III с использованием мегафункции ПЗУ LPM_ROM

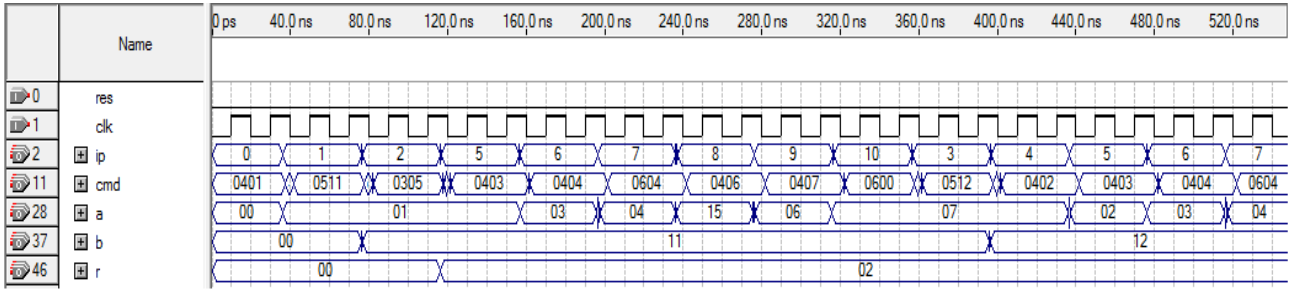


Рис.4.16. Функциональное моделирование работы микропроцессорного ядра с асинхронным ПЗУ на языке VHDL (ПЛИС APEX 20KE)

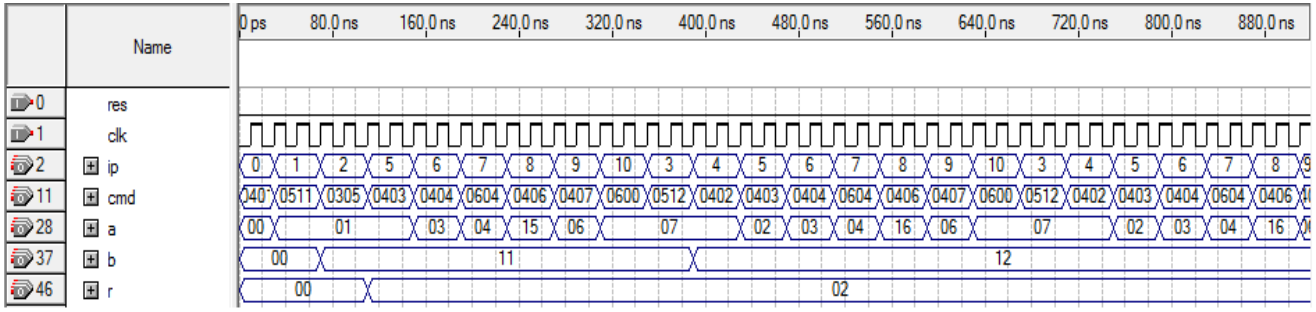


Рис.4.17. Функциональное моделирование работы микропроцессорного ядра с синхронным ПЗУ на языке VHDL (ПЛИС Stratix III)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY rom_syn IS
   PORT (
      clk : IN std_logic;
      addr : IN std_logic_vector(7 DOWNTO 0);
      rom_out : OUT std_logic_vector(15 DOWNTO 0));
END rom_syn;
ARCHITECTURE a OF rom_syn IS
   TYPE T_UFIX_16_256 IS ARRAY (255 DOWNTO 0) of
unsigned(15 DOWNTO 0);
BEGIN
  PROCESS (addr)
      -- local variables
      VARIABLE data_temp : T_UFIX_16_256;
   BEGIN
      FOR b IN 0 TO 255 LOOP
         data_temp(b) := to_unsigned(0, 16);
      END LOOP;
      data_temp(0) := to_unsigned(1025, 16);
      data_temp(1) := to_unsigned(1297, 16);
      data_temp(2) := to_unsigned(773, 16);
      data_temp(3) := to_unsigned(1298, 16);
      data_temp(4) := to_unsigned(1026, 16);
      data_temp(5) := to_unsigned(1027, 16);
      data_temp(6) := to_unsigned(1028, 16);
      data_temp(7) := to_unsigned(1540, 16);
      data_temp(8) := to_unsigned(1030, 16);
      data_temp(9) := to_unsigned(1031, 16);
      data_temp(10) := to_unsigned(1536, 16);
   rom_out <= std_logic_vector(data_temp(to_integer(unsigned(addr))));
   END PROCESS;
END a;
```

Пример 2. Описание асинхронного ПЗУ на языке VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY ozu_sin IS
   PORT (
      clk : IN std_logic;
      reset : IN std_logic;
      addr : IN std_logic_vector(7 DOWNTO 0);
      rom_out : OUT std_logic_vector(15 DOWNTO 0));
END ozu_sin;
ARCHITECTURE a OF ozu_sin IS
TYPE T_UFIX_16_256 IS ARRAY (255 DOWNTO 0) of unsigned(15
DOWNTO 0);
SIGNAL data, data_next: T_UFIX_16_256;
BEGIN
  PROCESS (reset, clk)
      VARIABLE b : INTEGER;
   BEGIN
      IF reset = '1' THEN
         FOR b IN 0 TO 255 LOOP
            data(b) <= to_unsigned(0, 16);
         END LOOP;
      ELSIF clk'EVENT AND clk= '1' THEN
            FOR b IN 0 TO 255 LOOP
               data(b) <= data_next(b);
            END LOOP;
      END IF;
   END PROCESS;
   PROCESS (addr)
      -- local variables
          VARIABLE data_temp : T_UFIX_16_256;
   BEGIN
      FOR b IN 0 TO 255 LOOP
         data_temp(b) := to_unsigned(0, 16);
      END LOOP;
      data_temp(0) := to_unsigned(1025, 16);
      data_temp(1) := to_unsigned(1297, 16);
```

```
      data_temp(2) := to_unsigned(773, 16);
      data_temp(3) := to_unsigned(1298, 16);
      data_temp(4) := to_unsigned(1026, 16);
      data_temp(5) := to_unsigned(1027, 16);
      data_temp(6) := to_unsigned(1028, 16);
      data_temp(7) := to_unsigned(1540, 16);
      data_temp(8) := to_unsigned(1030, 16);
      data_temp(9) := to_unsigned(1031, 16);
      data_temp(10) := to_unsigned(1536, 16);
      rom_out <=
std_logic_vector(data_temp(to_integer(unsigned(addr))));
      FOR c IN 0 TO 255 LOOP
         data_next(c) <= data_temp(c);
      END LOOP;
   END PROCESS;
END a;
```

Пример 3. Описание синхронного ПЗУ на языке VHDL (поведенческий уровень)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY ROM IS
PORT( clk                    :  IN   std_logic;
      reset                  :  IN   std_logic;
      addr                   :  IN   std_logic_vector(7 DOWNTO 0);
      cmd                    :  OUT  std_logic_vector(15 DOWNTO 0)
      );
END ROM;
ARCHITECTURE rtl OF ROM IS
  -- Signals
  SIGNAL s                        : unsigned(7 DOWNTO 0);
  SIGNAL Lookup_Table_out1        : unsigned(15 DOWNTO 0);
BEGIN
  s <= unsigned(addr);
  PROCESS(s)
```

```
  BEGIN
    CASE s IS
WHEN "00000000" => Lookup_Table_out1 <= "0000010000000001";
WHEN "00000001" => Lookup_Table_out1 <= "0000010100010001";
WHEN "00000010" => Lookup_Table_out1 <= "0000001100000101";
WHEN "00000011" => Lookup_Table_out1 <= "0000010100010010";
WHEN "00000100" => Lookup_Table_out1 <= "0000010000000010";
WHEN "00000101" => Lookup_Table_out1 <= "0000010000000011";
WHEN "00000110" => Lookup_Table_out1 <= "0000010000000100";
WHEN "00000111" => Lookup_Table_out1 <= "0000011000000100";
WHEN "00001000" => Lookup_Table_out1 <= "0000010000000110";
WHEN "00001001" => Lookup_Table_out1 <= "0000010000000111";
WHEN "00001010" => Lookup_Table_out1 <= "0000011000000000";
WHEN OTHERS => Lookup_Table_out1 <= "0000000000000000";
END CASE;
END PROCESS;
cmd <= std_logic_vector(Lookup_Table_out1);
END rtl;
```

Пример 4. Описание асинхронного ПЗУ на языке VHDL (уровень регистровых передач)

Отредактируем исходный код языка VHDL управляющего автомата микропроцессорного ядра приведенного в разделе 4.1 с использованием перечислимых типов. Применение перечислимых типов преследует две цели: улучшение смысловой читаемости программы; более четкий и простой визуальный контроль значений. Наиболее часто перечислимый тип используется для обозначения состояний конечных автоматов.

Перечислимый тип объявляется путем перечисления названий элементов-значений. Объекты, тип которых объявлен как перечислимый, могут содержать только те значения, которые указаны при перечислении, следовательно, количество всех возможных значений конечно. Объявление перечислимого типа имеет вид:

TYPE *имя_типа* **IS (** *название_элемента* [**,** *название_элемента*] **);**

```
Type State_type IS (stateA, stateB, stateC);
VARIABLE State : State_type;
.
State := stateB;
```

В данном примере объявляется переменная State, допустимыми значениями которой являются stateA, stateB, stateC. Пример 4 демонструет использование перечислимого типа для проектирования управляющего автомата.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity proc is
port (ip: inout std_logic_vector(7 downto 0);
        cmd: inout std_logic_vector(15 downto 0);
        clk,res: in std_logic;
        a: inout std_logic_vector(7 downto 0);
        b,r: inout std_logic_vector(7 downto 0));
end proc;
architecture a of proc is

TYPE state_values IS (st0, st1);
signal state, next_state: state_values;

begin
process(clk)
begin
if (res = '1') then
                a <="00000000";
                b <="00000000";
                ip <="00000000";
                r <="00000000";
elsif clk'event and clk='1' then
```

```
case state is
when st0=> next_state <=st1;
case conv_integer(cmd) is
when 0=> ip <= ip+1;
when 256 to 511 =>ip<=cmd(7 downto 0);
when 512 to 767 =>if conv_integer(a)=0
                                    then ip<=cmd(7 downto 0);
                                    else ip<=ip+1;
                                    end if;
when 768 to 1023 =>r<=ip; ip<=cmd(7 downto 0);
when 1024 to 1279 => a<=cmd(7 downto 0); ip<=ip+1;
when 1280 to 1535 => b<=cmd(7 downto 0); ip<=ip+1;
when 1536 =>ip<=r+1;
when 1537=>a<=b; ip<=ip+1;
when 1538=>b<=a; ip<=ip+1;
when 1539=>a<=b;b<=a; ip<=ip+1;
when 1540=>a<=a+b; ip<=ip+1;
when 1541=>a<=a-b; ip<=ip+1;
when 1542=>a<=a and b; ip<=ip+1;
when 1543=>a<=a or b; ip<=ip+1;
when 1544=>a<=a xor b; ip<=ip+1;
when 1545=>a<=a-1; ip<=ip+1;
when others=>ip<=ip+1;
end case;
when st1=> next_state<=st0;
end case;
end if;
end process;
end a;
```

Пример 5. Описание управляющего автомата микропроцессорного ядра с использованием перечислимого типа на языке VHDL с циклом работы в два такта

Изучая рис.4.14-4.17, видим, что функциональное и временное моделирование подтверждают правильность работы микропроцессорного ядра реализованного в базисах ПЛИС APEX20KE и Stratix III САПР ПЛИС Quartus II версии 8.1.

Независимо от типа используемой памяти (синхронный или асинхронный режим работы) процессор работает в два такта. По первому такту синхроимпульса происходит выборка и дешифрирование команды, а по второму такту происходит непосредственная отработка команды, например, выдача результатов в регистры общего назначения или загрузка новых команд, как в случае с “прыжковыми” командами, например, команды CALL или RET.

Таблица 4.3

Используемые ресурсы ПЛИС Stratix III EP3SL50F484C2

Проект	Logic utilization		Total block memory bits
	Combinational ALUTs	Dedicated logic registers	
Синхронное ПЗУ, ПЛИС Stratix III, мегафункция LPM_ROM, тип Auto	109	33	4096
*Синхронное ПЗУ, ПЛИС APEX20KE EP20K30ETC144, мегафункция LPM_ROM, тип Auto	188 LE		4096
Синхронное ПЗУ, ПЛИС Stratix III, мегафункция ROM: 1 PORT, тип Auto	109	33	4096
Синхронное ПЗУ, ПЛИС Stratix III, мегафункция ROM: 1 PORT, тип Auto (управляющий автомат, пример 5)	94	32	4096
Асинхронное ПЗУ на языке VHDL, пример 4	111	33	-
Синхронное ПЗУ на языке VHDL, управляющий автомат, пример 5	113	33	-

* приводится для сравнения

Для разработки микропроцессорных ядер в базисе ПЛИС Stratix III в качестве синхронного ПЗУ необходимо использовать универсальную мегафункцию altsyncram, которая может быть сконфигурирована как ROM: 1 PORT (мегафункция altsyncram может быть так же использоваться и для конфигурирования ОЗУ), что позволяет снизить нагрузку по числу используемых логических элементов (табл.4.3) и обеспечить синхронный режим работы ПЗУ. Если управляющий автомат и ПЗУ будут спроектированны с использованием языка VHDL, то в этом случае будет задействованна основная логика: таблицы перекодировок (ALUT) и внутренние триггеры логических элементов (logic registers). Из анализа представленной таблицы, можно сделать вывод, что в простейшем случае, наиболее оптимальным оказывается использование языка VHDL а блоки памяти используются не эффективно.

4.3. Проектирование учебного процессора для реализации в базисе ПЛИС с использованием системы Matlab/Simulink

Целью данного раздела является демонстрация возможностей системы визуально-иммитационного моделирования Matlab/Simulink по проектированию микропроцессорных ядер для реализации в базисе ПЛИС фирмы Altera.

При количестве логических вентилей в проекте свыше 5000, визуализировать выполняемые функции устройства, крайне сложно. Гарантировать, что весь проект можно осознать, только взглянув на его схему, практически невозможно.

Разработчики, использующие в своих проектах методологию системного уровня проектирования (Electronic System Level (ESL) Design - проектирование на системном

уровне или проектирование “сверху вниз”), получат очевидные преимущества. Во-первых, любой отдельно взятый разработчик может обеспечить решение задачи повышенной (и постоянно увеличивающейся) сложности, обращаясь к более высоким уровням абстракции и передавая реализацию мелких деталей проекта автоматизированному процессу. Проще и быстрее разрабатывать модели сложно-функциональных устройств на более высоких уровнях абстракции, чем с использованием уровня регистровых передач (RTL-уровень, register transfer level. Самый простой способ понять концепцию RTL-описания – представить сложно-функциональное устройство в виде совокупности регистров, связанных между собой элементами комбинационной логики и управляемых с помощью общего синхросигнала). Симуляция в этом случае выполняется на порядки быстрее, поскольку не симулируются несущественные для данного уровня абстракции детали.

Во-вторых, разработчики могут значительно сократить цикл производства и улучшить качество изделий, благодаря проверке функциональных возможностей, ещё на этапе проектирования, когда внесение изменений в системы легко и относительно дешево.

Первым этапом в потоке ESL-проектирования является определение требований к проекту. Требования устанавливаются конкретным заказчиком или определяются в результате соответствующих маркетинговых исследований.

Далее проводится системное проектирование и верификация алгоритмов, как правило, с помощью языка программирования $C/C^{++}/SystemC$ или с помощью специальных средств визуально-иммитационного моделирования типа MathLab/Simulink компании The MathWorks или SPW2000 (signal processing worksystem - рабочая среда обработки сигналов) компании Cadence. На этом этапе могут быть использованы и ESL-продукты компании Summit Design, такие как Visual Elite, FastC, System Architect и

Virtual CPU. С помощью текстовых или графических средств создаются исполняемые функциональные спецификации, которые описывают поведение проекта в рамках заданных ограничений. Функциональная верификация технологически независима. Такое описание лишено деталей реализации.

Simulink – графическая среда визуально-имитационного моделирования аналоговых и дискретных систем. Предоставляет пользователю графический интерфейс для конструирования моделей из стандартных функциональных блоков. Simulink работает с линейными, нелинейными, непрерывными, дискретными и многомерными системами. Система Matlab/Simulink содержит встроенный генератор кода языка описания аппаратных средств HDL (Simulink HDL Coder) и ориентирована на поддержку симулятора VHDL ModelSim. Simulink HDL Coder – программный продукт для генерации VHDL-кода без привязки к конкретной архитектуре ПЛИС и платформе по Simulink-моделям. Справедливости ради, следует заметить, что стиль кодирования может повлиять на производительность проектируемого устройства, особенно на ПЛИС. Логические эквивалентные, но разные RTL-операторы могут выдавать различные результаты.

Покажем возможности системы Matlab/Simulink на этапе ESL-проектирования по созданию микропроцессорных ядер. Воспользуемся системой команд синхронного процессора, реализованного с помощью управляющего автомата, с циклом работы в два такта.

Для учебных целей, разработаем два проекта с использованием асинхронного и синхронного ПЗУ на языке VHDL. Для ускорения процесса проектирования предлагается использовать возможности Simulink HDL Coder.

Запуск процессора в Simulink следует производить с использованием отладчика (Simulink Debugger). Перед отладкой необходимо зайти в меню Simulation/Configuration Parameters и выбрать диалог Solver (“решатели”, методы

численного решения дифференциальных и дифференциально-алгебраических уравнений). В Solver options выбрать **Type**: Fixed-step; **Solver**: discrete (no continuous state); **Fixed step size (fundamental sample time)** – 1.0.

На рис.4.18 показана отладка процессора. Процессор состоит из следующих блоков: ROM - ПЗУ процессора; COP – блок выделения полей команды; ALU – АЛУ процессора; RON – регистры общего назначения, регистры A и B; RSN – регистры специального назначения, регистр R для обеспечения команд обращения к подпрограммам (CALL) и возврата (RET) и регистр-счетчик Ip. На рис.4.19 показан файл прошивки ПЗУ. Для построения ПЗУ используется функциональный блок Lookup Table, который формирует таблицу, в строках которой находятся порядковые номера (адреса) команд. Например, строке 1 соответствует команда 1036D (мнемонический код MOV A,12). Адреса команд (входной порт addr), вещественные числа (Real World Values) 0,1,2.. 23, представляются в формате uint8 (Unsigned integer fixed-point data type, целые числа без знака в формате с фиксированной запятой, с 8-ми битной точностью), а команды (выходной порт) представляются в формате uint16, с 16-ти битной точностью. Все сигналы в процессоре представлены в формате uint8 кроме сигналов cmd и cmdData, они представлены в формате uint16.

В формате с фиксированной запятой без знака, вещественное число V можно считать обозначением полинома:

$$V = S * \left[\sum_{i=0}^{ws-1} b_i 2^i \right],$$

где b_i – двоичное число, $b_i = 1,0$; ws – размер слова, ответственно за точность представления десятичных чисел с запятой, максимально 128 бит; S – масштаб, $S = 2^E$, E – число битов левее младшего значащего бита (LSB) в слове, обозначающие месторасположение позиционной точки (разделительной точки или разделительной запятой).

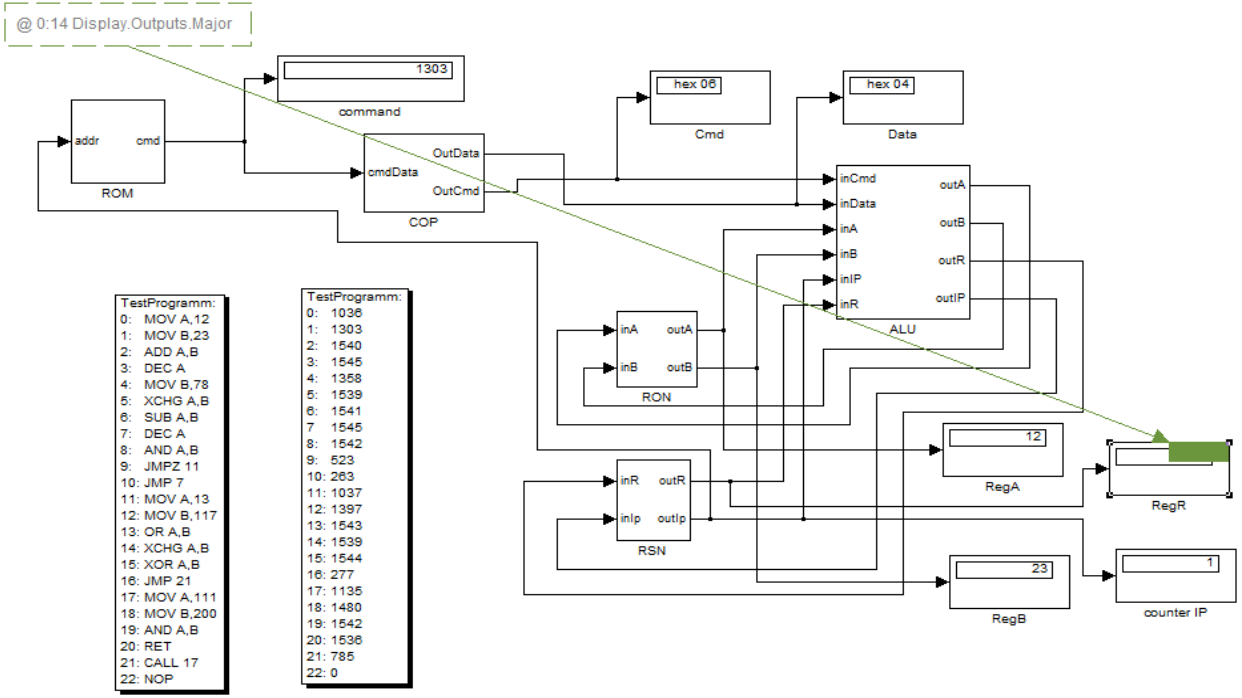


Рис.4.18. Отладка модели микропроцессорного ядра в системе Matlab/Simulink и тестовая программа (файл прошивки ПЗУ, содержимое блока ROM)

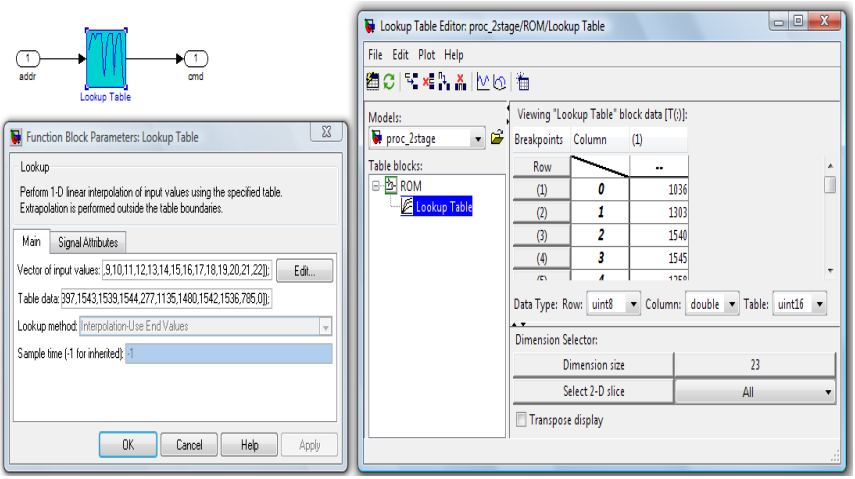


Рис.4.19. Файл прошивки ПЗУ (содержимое блока ROM) в системе Matlab/Simulink с использованием функционального блока Lookup Table

Для целых чисел $S=1$. Например, число 0011.0101 при длине слова $ws=8$:

$$3.3125=2^{-4}\left(\begin{array}{l}0*2^7+0*2^6+1*2^5+\\+1*2^4+0*2^3+1*2^2+0*2^1+1*2^0\end{array}\right).$$

Выделение полей команды осуществляется в блоке COP (рис.4.20). Сигнал OutCmd (uint8) – 8 старших бит 16-ти разрядной команды CmdData (uint16) извлекаемой из ПЗУ. Сигнал OutData (uint8) – 8 младших бит, содержащие данные. Функциональные блоки Shift Arithmetic (арифметический сдвиг) и Data Type Conversion (преобразование типа данных) используются для формирования сигнала OutCmd. Они осуществляют логический сдвиг вправо сигнала CmdData, младшими битами вперед. Для формирования сигнала OutData осуществляется логический сдвиг влево сигнала CmdData, старшими битами вперед, а затем сдвиг вправо, на 8 бит вперед.

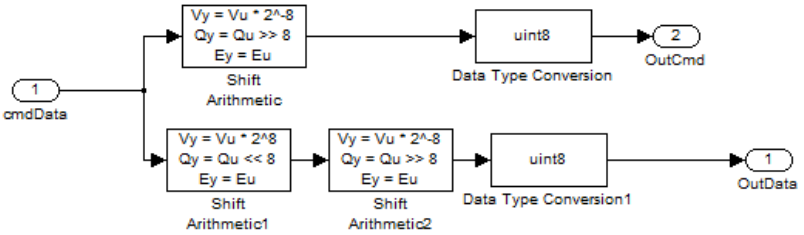


Рис.4.20. Выделение полей команды с помощью операций арифметического сдвига

В блоках Display (command, Cmd, Data, RegA, RegB, RegR, counter IP) отображается содержимое внутренних узлов процессора. Например, по команде MOV A,12 в регистр A загружается число 12, по команде MOV B,23 в регистр B загружается число 23. A по команде 1540 должно произойти сложение содержимых регистров, с сохранением результата в регистре A.

На рис.4.21 показан блок АЛУ. Пример 1 показывает M-файл функции АЛУ в системе Matlab/Simulink. Текст функции начиается с заголовка function, имеющего следующий вид:

function [y1, y2, ...]=fname(x1,x2, ...),

где fname – имя фунции; x1, x2 и т.д. – входные параметры; y1, y2 и т.д. – выходные параметры. В условном операторе switch выбирается одна из возможных альтернатив. Запись оператора выбора производится с помощью ключевых слов switch, case, otherwise. При выполнении оператора switch значение inCmd поочередно сравнивается с 1,2,3,… При обнаружении первого совпадения выполняются операторы соответствующей ветви, после чего производится выход из оператора выбора.

На рис.4.22 показано построение регистров общего назначения с использованием функциональных блоков

Memory. Аналогично формируются и блоки регистров специального назначения.

Далее с помощью Simulink HDL Coder сгенерируем код языка VHDL функциональных блоков проектируемого процессора. При этом необходимо помнить, что САПР ПЛИС Quartus II относится к компиляторам-синтезаторам и имеет привязку к конкретной архитектуре, а сгенерированный код языка VHDL с помощью Simulink HDL Coder в первую очередь предназначен для ModelSim SE (Mentor Graphics HDL simulator). Последняя версия САПР ПЛИС Quartus II (версия 8.1) содержит адаптированный симулятор ModelSim-Altera.

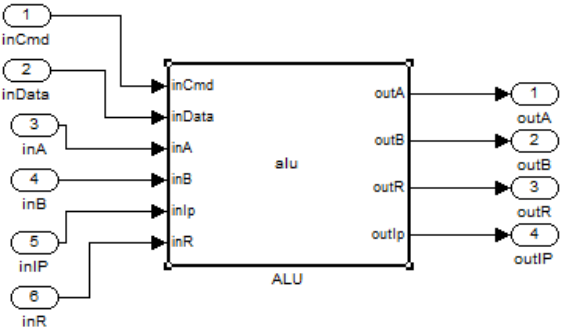


Рис.4.21. Блок АЛУ процессора

```
function [outA, outB, outR, outIp] = alu(inCmd,inData,inA,inB,inIp,inR)
outA = inA;
outB = inB;
outR = inR;
outIp = inIp+1;
switch inCmd
   case 0 %NOP
   case 1 %JMP
      outIp = inData;
   case 2 %JMPZ
      if inA == 0
         outIp = inData;
```

```
    end
  case 3 % CALL
    outR = inIp+1;
    outIp = inData;
  case 4 %MOV A,xx
    outA = inData;
  case 5 %MOV B,xx
    outB = inData;
  case 6
      switch inData
        case 0 %RET
          outIp = inR;
        case 1 %MOV A,B
           outA = inB;
        case 2 %MOV B,A
           outB = inA;
        case 3 %XCHG A,B
          X = inB;
          outB = inA;
          outA = X;
        case 4 %ADD A,B
          outA = inA+inB;
        case 5 %SUB A,B
          outA = inA-inB;
        case 6 %AND A,B
          outA = bitand(inA,inB);
        case 7 %OR A,B
          outA = bitor(inA,inB);
        case 8 %XOR A,B
          outA = bitxor(inA,inB);
        case 9 %DEC A
          outA = inA-1;
      end
end
```

Пример 1. М-файл функции АЛУ в системе Matlab/Simulink

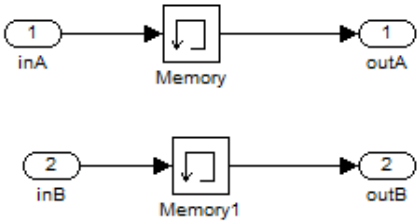


Рис.4.22. Регистры общего назначения процессора

Пример 2 показывает сгенерированное описание прошивки ПЗУ на языке VHDL, пример 3 выделение полей команды, пример 4 поведенческое описание АЛУ, пример 5 – регистры общего назначения. С целью сокращения избыточности кода, все блоки процессора подверглись ручному редактированию. При генерации кода языка VHDL функциональных блоков процессора, Simulink HDL Coder использует только два пакета std_logic_1164 и numeric_std. Так как заранее предполагается, что процессор работает только с целыми положительными двоичными числами (тип unsigned). Добавим в каждый блок следующие пакеты std_logic_arith и std_logic_unsigned. Компилятор-синтезатор наиболее эффективно отображает в ресурсы ПЛИС функции (например, арифметические) которые используются из пакетов std_logic_1164, numeric_std, std_logic_arith и std_logic_unsigned.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
ENTITY ROM IS
PORT(
        addr:   IN    std_logic_vector(7 DOWNTO 0);
        cmd:    OUT   std_logic_vector(15 DOWNTO 0));
```

```
END ROM;
ARCHITECTURE rtl OF ROM IS
SIGNAL Lookup_Table_out1: std_logic_vector(15 DOWNTO 0);

BEGIN
PROCESS(addr)
BEGIN
CASE addr IS
WHEN "00000000" => Lookup_Table_out1 <= "0000010000001100";
WHEN "00000001" => Lookup_Table_out1 <= "0000010100010111";
WHEN "00000010" => Lookup_Table_out1 <= "0000011000000100";
WHEN "00000011" => Lookup_Table_out1 <= "0000011000001001";
WHEN "00000100" => Lookup_Table_out1 <= "0000010101001110";
WHEN "00000101" => Lookup_Table_out1 <= "0000011000000011";
WHEN "00000110" => Lookup_Table_out1 <= "0000011000000101";
WHEN "00000111" => Lookup_Table_out1 <= "0000011000001001";
WHEN "00001000" => Lookup_Table_out1 <= "0000011000000110";
WHEN "00001001" => Lookup_Table_out1 <= "0000001000001011";
WHEN "00001010" => Lookup_Table_out1 <= "0000000100000111";
WHEN "00001011" => Lookup_Table_out1 <= "0000010000001101";
WHEN "00001100" => Lookup_Table_out1 <= "0000010101110101";
WHEN "00001101" => Lookup_Table_out1 <= "0000011000000111";
WHEN "00001110" => Lookup_Table_out1 <= "0000011000000011";
WHEN "00001111" => Lookup_Table_out1 <= "0000011000001000";
WHEN "00010000" => Lookup_Table_out1 <= "0000000100010101";
WHEN "00010001" => Lookup_Table_out1 <= "0000010001101111";
WHEN "00010010" => Lookup_Table_out1 <= "0000010111001000";
WHEN "00010011" => Lookup_Table_out1 <= "0000011000000110";
WHEN "00010100" => Lookup_Table_out1 <= "0000011000000000";
WHEN "00010101" => Lookup_Table_out1 <= "0000001100010001";
WHEN "00010110" => Lookup_Table_out1 <= "0000000000000000";
WHEN OTHERS => Lookup_Table_out1 <= "0000000000000000";
END CASE;
END PROCESS;
cmd <=Lookup_Table_out1;
END rtl;
```

Пример 2. Файл прошивки асинхронного ПЗУ на языке VHDL

```
library ieee;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
ENTITY COP IS
PORT(
cmdData :   IN    std_logic_vector(15 DOWNTO 0);
OutData :   OUT   std_logic_vector(7 DOWNTO 0);
OutCmd  :   OUT   std_logic_vector(7 DOWNTO 0));
END COP;
ARCHITECTURE rtl OF COP IS
BEGIN
OutCmd<=cmdData(15 downto 8);
OutData<=cmdData(7 downto 0);
END rtl;
```

Пример 3. Выделение полей команды на языке VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
ENTITY ALU_entity IS
PORT (
      inCmd : IN std_logic_vector(7 DOWNTO 0);
      inData : IN std_logic_vector(7 DOWNTO 0);
      inA : IN std_logic_vector(7 DOWNTO 0);
      inB : IN std_logic_vector(7 DOWNTO 0);
      inIp : IN std_logic_vector(7 DOWNTO 0);
      inR : IN std_logic_vector(7 DOWNTO 0);
      outA : OUT std_logic_vector(7 DOWNTO 0);
      outB : OUT std_logic_vector(7 DOWNTO 0);
      outR : OUT std_logic_vector(7 DOWNTO 0);
      outIp : OUT std_logic_vector(7 DOWNTO 0));
END ALU_entity;
ARCHITECTURE fsm_SFHDL OF ALU_entity IS
BEGIN
PROCESS (inCmd, inData, inA, inB, inIp, inR)
```

```
BEGIN
   outA <= inA;
   outB <= inB;
   outR <= inR;
   outIp <= inIp+1;
   CASE inCmd IS
      WHEN "00000000" =>
         --NOP
         NULL;
      WHEN "00000001" =>
         --JMP
         outIp <= inData;
      WHEN "00000010" =>
         IF inA = 0 THEN
            --JMPZ
            outIp <= inData;
         END IF;
      WHEN "00000011" =>
         -- CALL
        outR <= inIp+1;
         outIp <= inData;
      WHEN "00000100" =>
         --MOV A,xx
         outA <= inData;
      WHEN "00000101" =>
         --MOV B,xx
         outB <= inData;
      WHEN "00000110" =>
         CASE inData IS
            WHEN "00000000" =>
               --RET
               outIp <= inR;
            WHEN "00000001" =>
               --MOV A,B
               outA <= inB;
            WHEN "00000010" =>
               --MOV B,A
               outB <= inA;
            WHEN "00000011" =>
```

```
                --XCHG A,B
                outB <= inA;
                outA <= inB;
              WHEN "00000100" =>
                --ADD A,B
                outA <= inA + inB;
              WHEN "00000101" =>
                --SUB A,B
                outA <= inA - inB;
              WHEN "00000110" =>
                --AND A,B
                outA <= inA AND inB;
              WHEN "00000111" =>
                --OR A,B
                outA <= inA OR inB;
              WHEN "00001000" =>
                --XOR A,B
                outA <= inA XOR inB;
              WHEN "00001001" =>
                --DEC A
                outA <= inA - 1;
              WHEN OTHERS =>
                NULL;
            END CASE;
          WHEN OTHERS =>
            NULL;
        END CASE;
    END PROCESS;
END fsm_SFHDL;
```

Пример 4. Асинхронное АЛУ на языке VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
ENTITY RON IS
PORT( clk :   IN    std_logic;
      reset :   IN    std_logic;
```

```
      enb :   IN    std_logic;
      inA :   IN    std_logic_vector(7 DOWNTO 0);
      inB :   IN    std_logic_vector(7 DOWNTO 0);
      outA :   OUT   std_logic_vector(7 DOWNTO 0);
      outB :   OUT   std_logic_vector(7 DOWNTO 0));
END RON;
ARCHITECTURE rtl OF RON IS
BEGIN
  PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      outA <= (OTHERS => '0');
    ELSIF clk'event AND clk = '1' THEN
      IF enb = '1' THEN
       outA <= inA;
      END IF;
    END IF;
  END PROCESS;
  PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      outB <= (OTHERS => '0');
    ELSIF clk'event AND clk = '1' THEN
      IF enb = '1' THEN
        outB <= inB;
      END IF;
    END IF;  END PROCESS; END rtl;
```

Пример 5. Регистры общего назначения (РОН)

На рис.4.23 показана тестовая схема микропроцессорного ядра в графическом редакторе САПР ПЛИС Quartus II, построенная по модели разработанной в системе Matlab/Simulink и временные диаграммы работы (рис.4.24). Анализируя полученные результаты, приходим к выводу, что ПЗУ и АЛУ являются асинхронными блоками. Регистры общего и специального назначения представляют собой 8-ми разрядные регистры тактируемые фронтом синхросигнала, с асинхронным сбросом reset и синхронным

сигналом ena. Недостатком сгенерированного кода языка VHDL, является нетактируемое АЛУ и ПЗУ.

Сгенерируем файл прошивки ПЗУ, используя М-файл и fi-объекты системы Matlab. Fi-объекты позволяют представлять числа в формате с фиксированной запятой. Например, по команде a=fi(1536) целое положительное десятичное число 1536 будет представлено в формате M.N, где M - общее число двоичных разрядов, N - число разрядов дробной части. Наиболее распространенный формат 16.15. Пятнадцать разрядов после запятой обеспечивают дискретность представления равную $2^{-15} \approx 3*10^{-5}$. В нашем случае используем следующий формат: a = fi(v,s,w,f), где v - объект со значением, s – знак (0 (false) – для чисел без знака и 1 (true) – для чисел со знаком), w - размер слова в битах (целая часть числа), f – дробная часть числа в битах. Например, по команде a=fi(1536, 0, 16, 0) целое положительное десятичное число 1536 будет представлено в формате 16.0. По команде disp(bin(a)) можно посмотреть десятичное число в двоичной форме: 0000011000000000. Пример 6 показывает М-файл прошивки ПЗУ в системе Matlab/Simulink. А пример 7 отредактированный вариант сгенерированного файла программой Simulink HDL Coder в САПР Quartus II.

Анализируя код (пример 7), можно сделать вывод, что сгенерированно синхронное ПЗУ с асинхронным сбросом, синхронным сигналом разрешения тактирования clk_enable. Для организации массива памяти используется последовательный оператор **for ... loop**, выполняющий повторяющиеся операторы. Оператор **for ... loop** имеет целую схему итерации, при которой количество повторов определяется целым диапазоном. Цикл повторяется один раз для каждого значения диапазона. После того, как будет достигнуто последнее значение диапазона итерации, цикл пропускается, и выполнение программы продолжается, начиная с оператора, стоящего вслед за циклом.

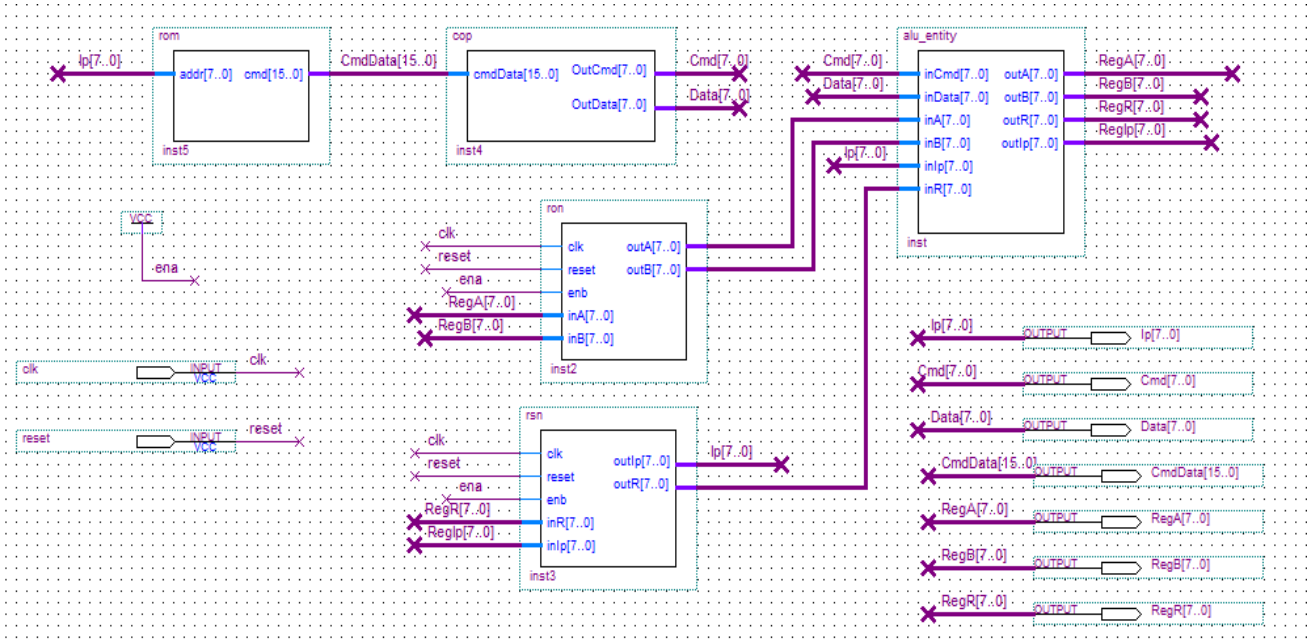


Рис.4.23. Тестовая схема микропроцессорного ядра в графическом редакторе САПР ПЛИС Quartus II версии 8.1, построенная по модели. Код языка VHDL функциональных блоков модели сгенерирован с помощью приложения Simulink HDL Coder системы Matlab/Simulink

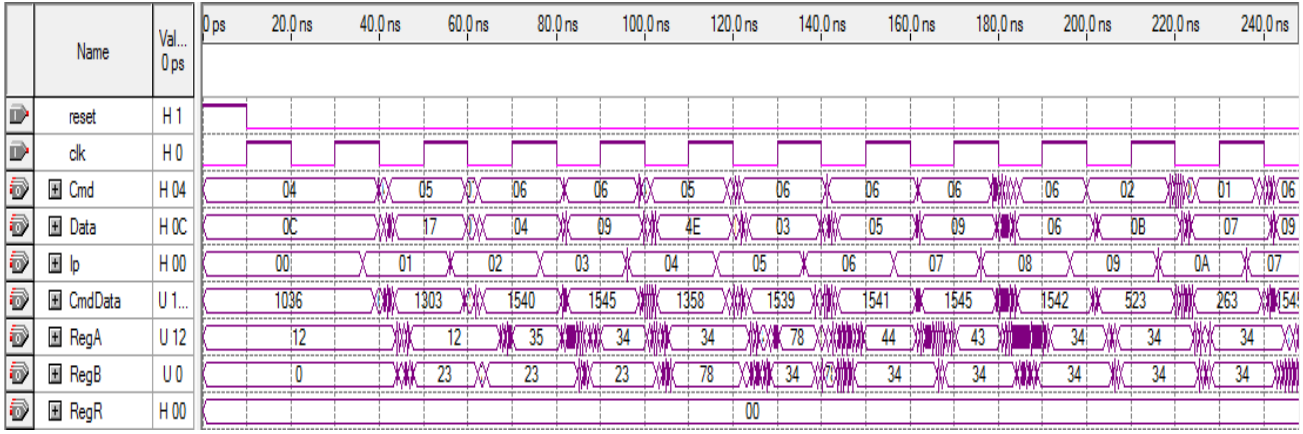


Рис.4.24. Временная диаграмма работы микропроцессорного ядра, код языка VHDL функциональных блоков которого сгенерирован с помощью приложения Simulink HDL Coder системы Matlab/Simulink

```
function rom_out = Memory(addr)
persistent data;
data = fi(zeros(1, 256), 0, 16, 0);
data(1) = fi(1036, 0, 16, 0);
data(2) = fi(1303, 0, 16, 0);
data(3) = fi(1540, 0, 16, 0);
data(4) = fi(1545, 0, 16, 0);
data(5) = fi(1358, 0, 16, 0);
data(6) = fi(1539, 0, 16, 0);
data(7) = fi(1541, 0, 16, 0);
data(8) = fi(1545, 0, 16, 0);
data(9) = fi(1542, 0, 16, 0);
data(10) = fi(523, 0, 16, 0);
data(11) = fi(263, 0, 16, 0);
data(12) = fi(1037, 0, 16, 0);
data(13) = fi(1397, 0, 16, 0);
data(14) = fi(1543, 0, 16, 0);
data(15) = fi(1539, 0, 16, 0);
data(16) = fi(1544, 0, 16, 0);
data(17) = fi(277, 0, 16, 0);
data(18) = fi(1135, 0, 16, 0);
data(19) = fi(1480, 0, 16, 0);
data(20) = fi(1542, 0, 16, 0);
data(21) = fi(1536, 0, 16, 0);
data(22) = fi(785, 0, 16, 0);
data(23) = fi(0, 0, 16, 0);
rom_out = data(addr+1);
```

Пример 6. М-файл прошивки ПЗУ в системе Matlab/Simulink

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY rom_syn IS
   PORT (
      clk : IN std_logic;
      clk_enable : IN std_logic;
      reset : IN std_logic;
```

```
      addr : IN std_logic_vector(7 DOWNTO 0);
      rom_out : OUT std_logic_vector(15 DOWNTO 0));
END rom_syn;
ARCHITECTURE a OF rom_syn IS
   TYPE T_UFIX_16_256 IS ARRAY (255 DOWNTO 0) of unsigned(15
DOWNTO 0);
   SIGNAL data : T_UFIX_16_256;
   SIGNAL data_next : T_UFIX_16_256;
BEGIN
   PROCESS (reset, clk)
      -- local variables
      VARIABLE b : INTEGER;
   BEGIN
      IF reset = '1' THEN
         NULL;
      ELSIF clk'EVENT AND clk= '1' THEN
         IF clk_enable= '1' THEN
            FOR b IN 0 TO 255 LOOP
               data(b) <= data_next(b);
            END LOOP;
         END IF;
      END IF;
   END PROCESS;
  PROCESS (addr)
      -- local variables
      VARIABLE data_temp : T_UFIX_16_256;
   BEGIN
      FOR b IN 0 TO 255 LOOP
         data_temp(b) := to_unsigned(0, 16);
      END LOOP;
      data_temp(0) := to_unsigned(1036, 16);
      data_temp(1) := to_unsigned(1303, 16);
      data_temp(2) := to_unsigned(1540, 16);
      data_temp(3) := to_unsigned(1545, 16);
      data_temp(4) := to_unsigned(1358, 16);
      data_temp(5) := to_unsigned(1539, 16);
      data_temp(6) := to_unsigned(1541, 16);
      data_temp(7) := to_unsigned(1545, 16);
      data_temp(8) := to_unsigned(1542, 16);
```

```
    data_temp(9) := to_unsigned(523, 16);
    data_temp(10) := to_unsigned(263, 16);
    data_temp(11) := to_unsigned(1037, 16);
    data_temp(12) := to_unsigned(1397, 16);
    data_temp(13) := to_unsigned(1543, 16);
    data_temp(14) := to_unsigned(1539, 16);
    data_temp(15) := to_unsigned(1544, 16);
    data_temp(16) := to_unsigned(277, 16);
    data_temp(17) := to_unsigned(1135, 16);
    data_temp(18) := to_unsigned(1480, 16);
    data_temp(19) := to_unsigned(1542, 16);
    data_temp(20) := to_unsigned(1536, 16);
    data_temp(21) := to_unsigned(785, 16);
    data_temp(22) := to_unsigned(0, 16);
  rom_out <= std_logic_vector(data_temp(to_integer(unsigned(addr))));
  END PROCESS;
END a;
```

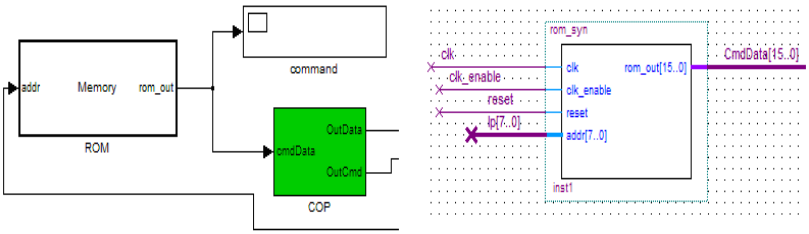
Пример 7. Файл прошивки синхронного ПЗУ на языке VHDL

Следует упомянуть про функции преобразования типов to_integer и to_unsigned из пакета Numeric_std, которые используются при проектировании синхронного ПЗУ. Функция to_integer преобразует тип unsigned в подтип natural (встроенный подтип natural используют для объектов, которые, не должны принимать отрицательные значения), а функция to_unsigned преобразует подтип natural в тип unsigned, при этом необходимо указывать размер желаемого слова.

На рис.4.25 показаны изменения, которые необходимо внести в проект для ПЗУ с использованием М-функции в системе Matlab/Simulink (рис.4.25, а) и для синхронного ПЗУ в САПР ПЛИС Quartus II (рис.4.25, б). На рис.4.26 представлена временная диаграмма работы процессора с синхронным ПЗУ.

Проект микропроцессора с синхронным ПЗУ, код языка VHDL которого был получен с использованием Simulink HDL Coder, продемонстрировал работоспособность, как на старых, так и на новых сериях ПЛИС фирмы Altera, что не удавалось

осуществить с использованием встроенных мегафункций ОЗУ и ПЗУ.



а) б)

Рис.4.25. Фрагмент модели и схемы процессора: а – ПЗУ с использованием М-функции в системе Matlab/Simulink; б - символ синхронного ПЗУ в САПР Quartus II

На рис.4.27 показана тестовая схема процессора в графическом редакторе САПР ПЛИС Quartus II (версия 8) с управляющим автоматом с циклом работы в два такта (пример 1, раздел 4.1) и синхронным ПЗУ на языке VHDL. Сравнивая рис.4.26 и рис.4.28 видим, что два процессора логически работают одинаково.

Система Matlab/simulink с Simulink HDL Coder может быть эффективно использована для ускорения процесса разработки моделей микропроцессорных ядер. Проект микропроцессора с асинхронным ПЗУ на языке VHDL может быть успешно размещен в ПЛИС APEX20KE EP20K30ETC144-1, при этом общее число задействованных ресурсов составляет 22 %, с рабочей частотой до 60 МГц.

Недостатком процессора реализованного в системе Matlab/simulink и адаптированного в САПР Quartus II, является отсутствие управляющего автомата.

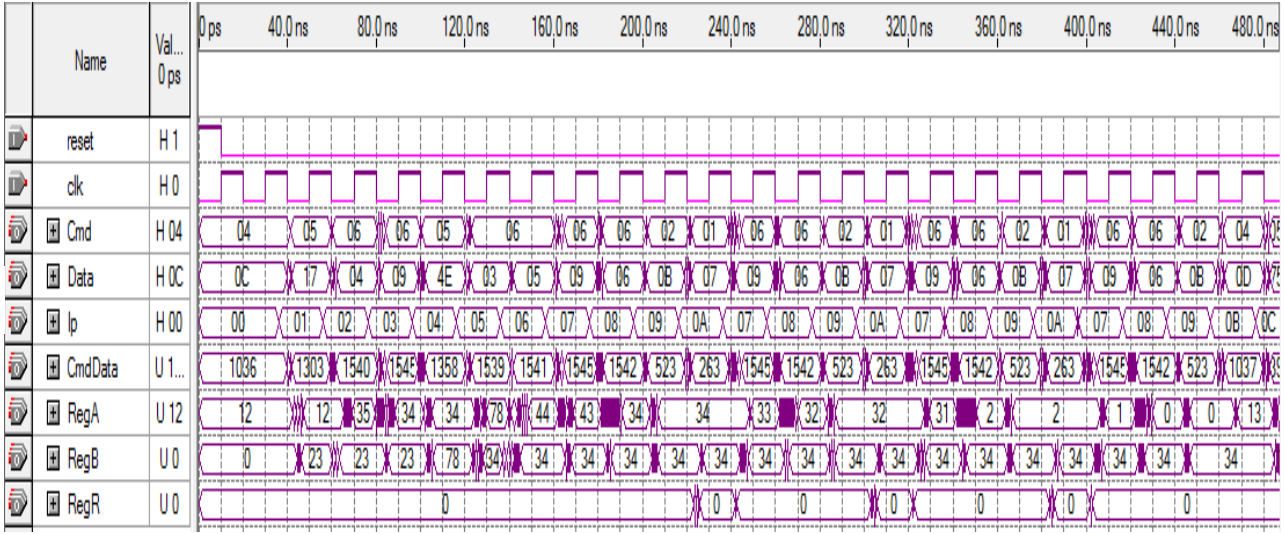


Рис.4.26. Временная диаграмма работы микропроцессорного ядра с синхронным ПЗУ на языке VHDL (код языка VHDL функциональных блоков сгенерирован с помощью приложения Simulink HDL Coder системы Matlab/Simulink)

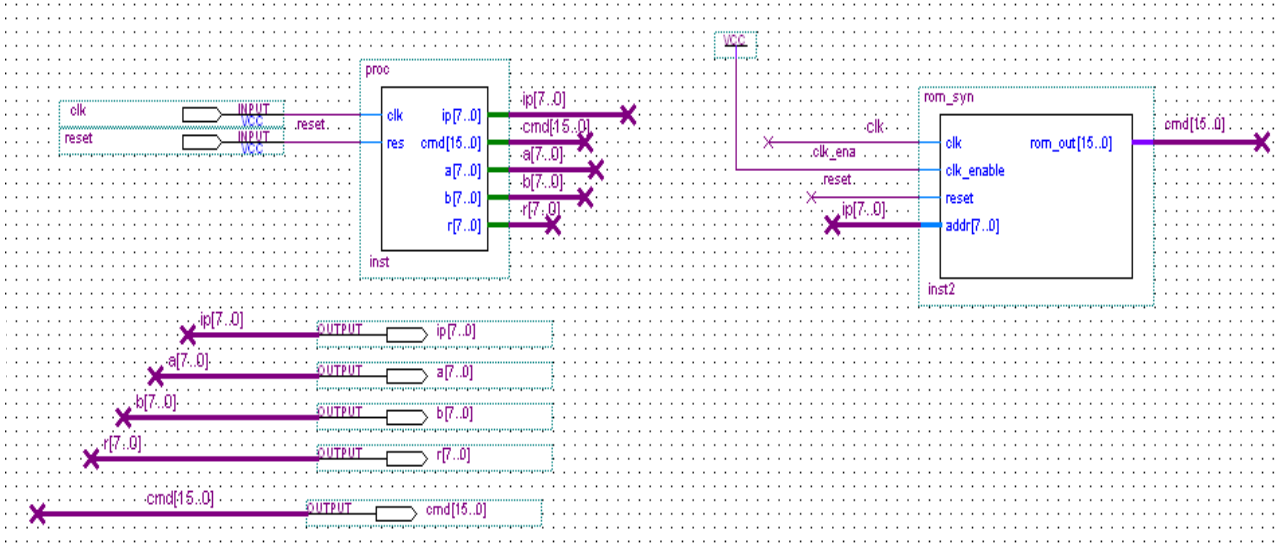


Рис.4.27. Тестовая схема микропроцессорного ядра в графическом редакторе САПР ПЛИС Quartus II (версия 8) с управляющим автоматом с циклом работы в два такта (пример 1, раздел 4.1) и синхронным ПЗУ на языке VHDL, код языка которого сгенерирован с помощью приложения Simulink HDL Coder системы Matlab/Simulink

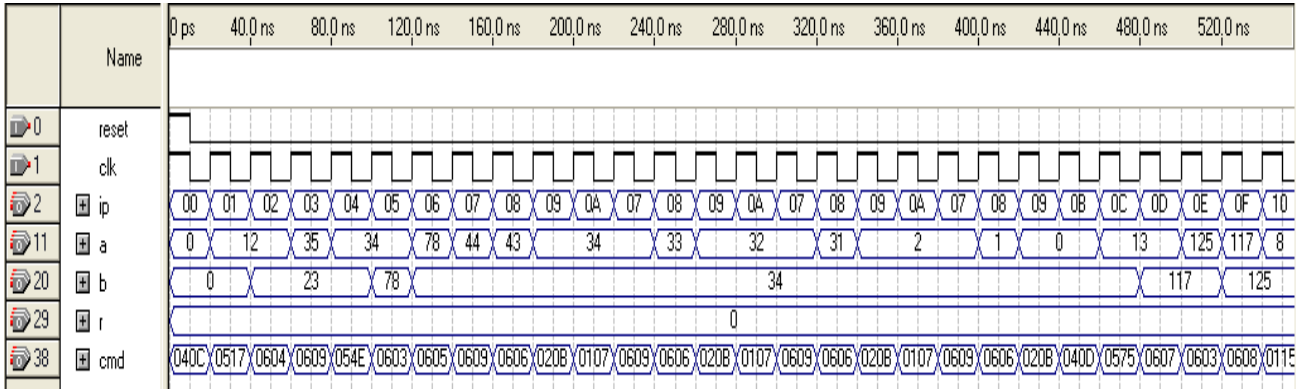


Рис.4.28. Временная диаграмма работы микропроцессорного ядра с управляющим автоматом с циклом работы в два такта (пример 1, раздел 4.1) и синхронным ПЗУ на языке VHDL

4.4. Проектирование учебного процессора с фиксированной запятой в системе Matlab/Simulink

В выше приведенных примерах показаны примеры проектирования микропроцессорных ядер для реализации в базисе ПЛИС фирмы Altera, как с использованием мегафункций асинхронного ОЗУ/ПЗУ САПР Quartus II, так и с использованием функциональных блоков на языке VHDL, сгенерированных с помощью Simulink HDL Coder системы Matlab/Simulink. Общим недостатком является отсутствие управляющего автомата. Предлагается спроектировать в системе Matlab/Simulink процессор с управляющим автоматом и позволяющим проводить вычисления с фиксированной зяпятой. Выполнение арифметических операций над операндами, представленными в формате с фиксированной запятой, позволяет получать высокую скорость вычислений, но возможно переполнение разрядной сетки либо значительной погрешности из-за округления.

На рис.4.29 показан процессор с управляющим автоматом на шесть состояний и его отладка в системе Matlab/Simulink с использованием отладчика (Simulink Debugger). Перед отладкой необходимо в меню Simulation/Configuration Parameters выбрать диалог Solver (“решатели”, методы численного решения дифференциальных и дифференциально-алгебраических уравнений). В Solver options выбрать **Type**: Fixed-step; **Solver**: discrete (no continuous state); **Fixed step size (fundamental sample time)** – 1.0. Осуществляется тестирование команд MOV A,12; MOV B,23; ADD A,B.

Проектируемый процессор состоит из следующих блоков: управляющий автомат (блок CPU_Controller, пример 1); память программ - ПЗУ процессора (блок Memory, пример 4); АЛУ процессора (блок alu, пример 7); двух регистров общего назначения (РОН, блоки RegisterA, пример 6 и

RegisterB); регистра специального назначения (PCH, блок PC_Inc, пример 2), необходимого для обеспечения “прыжковых” команд, таких как JMP, JMPZ, CALL и RET; счетчика команд (блок PC, пример 3); регистра инструкций (блок Instruction_Reg, пример 5).

Процессор реализован в формате с фиксированной запятой, с использованием fi-объектов системы Matlab. Будем используем следующий формат, для представления десятичных чисел:

a = fi(v, s, w, f),

где v – десятичное число, s – знак (0 (false)– для чисел без знака и 1 (true) – для чисел со знаком), w - размер слова в битах (целая часть числа), f – дробная часть числа в битах. Все используемые десятичные числа в процессоре беззнаковые (положительные) и целые. В системе Matlab пользователь имеет возможность определить беззнаковые (например, uint8, uint16) и знаковые целые числа (sint), с помощью внутренних форматов.

При проектировании процессоров с фиксированной запятой необходимо учитывать следующие факторы: диапазон для результатов вычислений; требуемую погрешность результата; ошибки, связанные с квантованием; алгоритм реализации вычислений и др.

Это связано с тем, что десятичное число v представляется с использованием формулы: $V \cong 2^{-m} \times Q$, где $m-$длина дробной части числа; для беззнаковых чисел $Q = \sum_{i=0}^{n-1} W_i \times 2^i$, $W_i -$весовые коэффициенты, $2^i -$веса двоичных разрядов машинного слова, $n -$длина двоичного слова в битах. Диапазон целого беззнакового числа определяется выражением: $0 \le V \le 2^n - 1$.

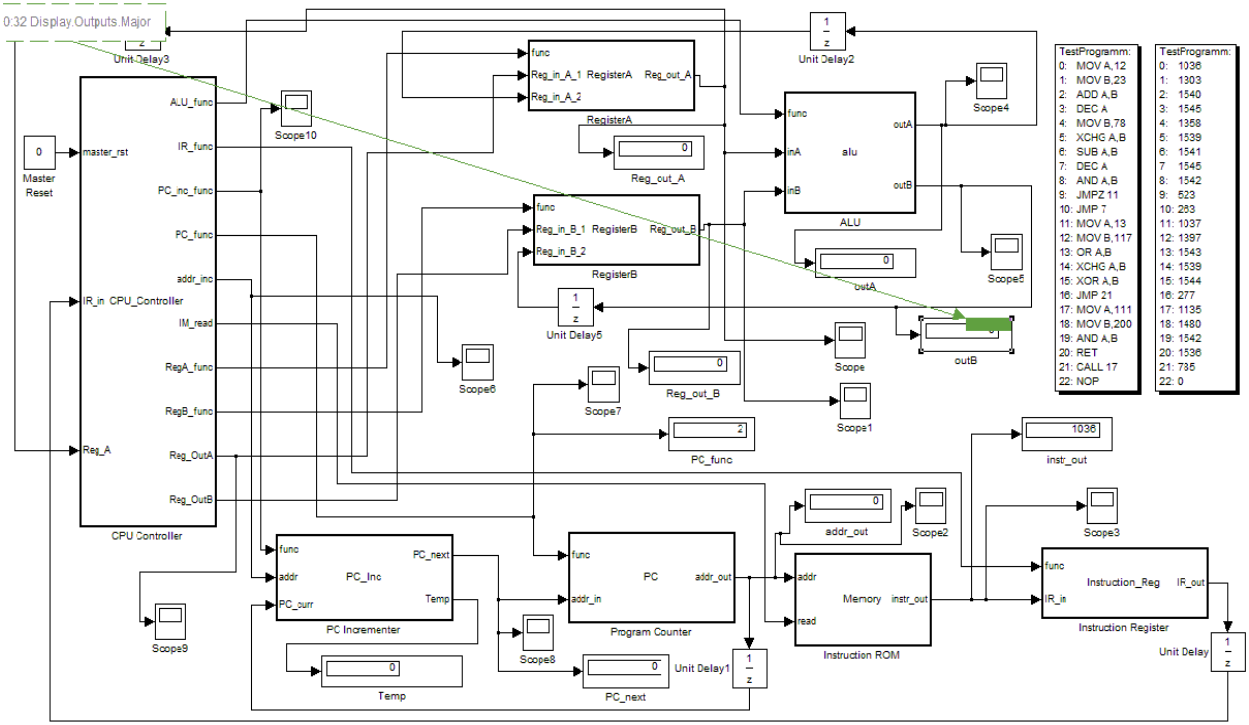


Рис.4.29. Модель процессора с управляющим автоматом в системе Matlab/Simulink. Тестирование команд MOV A,12; MOV B,23; ADD A,B

Это можно осуществить с использование следующего формата:

a = fi(v, s, w, f, fimath).

```
% HDL specific fimath
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);
```

Данные настройки вычислений в формате с фиксированной запятой приняты в системе Simulink по умолчанию. Можно задать режим округления (Roundmode) – ‘floor’ – округление вниз; реакцию на переполнение (OverflowMode) – ‘wrap’ – перенос, при выходе значения v из допустимого диапазона, “лишние” старшие разряды игнорируются. При выполнении операций умножения (‘ProductMode’) и сложения (SumMode), для повышения точности вычислений (precision) используется машинное слово шириной в 32 бита.

Для блоков РОН, в качестве примера, используем формат a = fi(v, s, w, f, fimath). Можно также добавить учет выше приведенных факторов и в другие М-файлы функций блоков процессора. Это позволит “управлять” встроенным генератором кода языка HDL (Simulink HDL Coder). Если этого не сделать, то необходимо с использованием проводника модели осуществить настройки блоков процессора для вычислений в формате с фиксированной запятой (рис.4.30).

Процессор имеет распределенное управление. В блоках alu, RegisterA, RegisterB, PC_Inc и PC, имеется свой локальный управляющий сигнал func, дешифрация которого внутри блоков будет приводить к выполнению некоторых операций, например, к изменению внутреннего содержимого блока или, наоборот, к его сохранению.

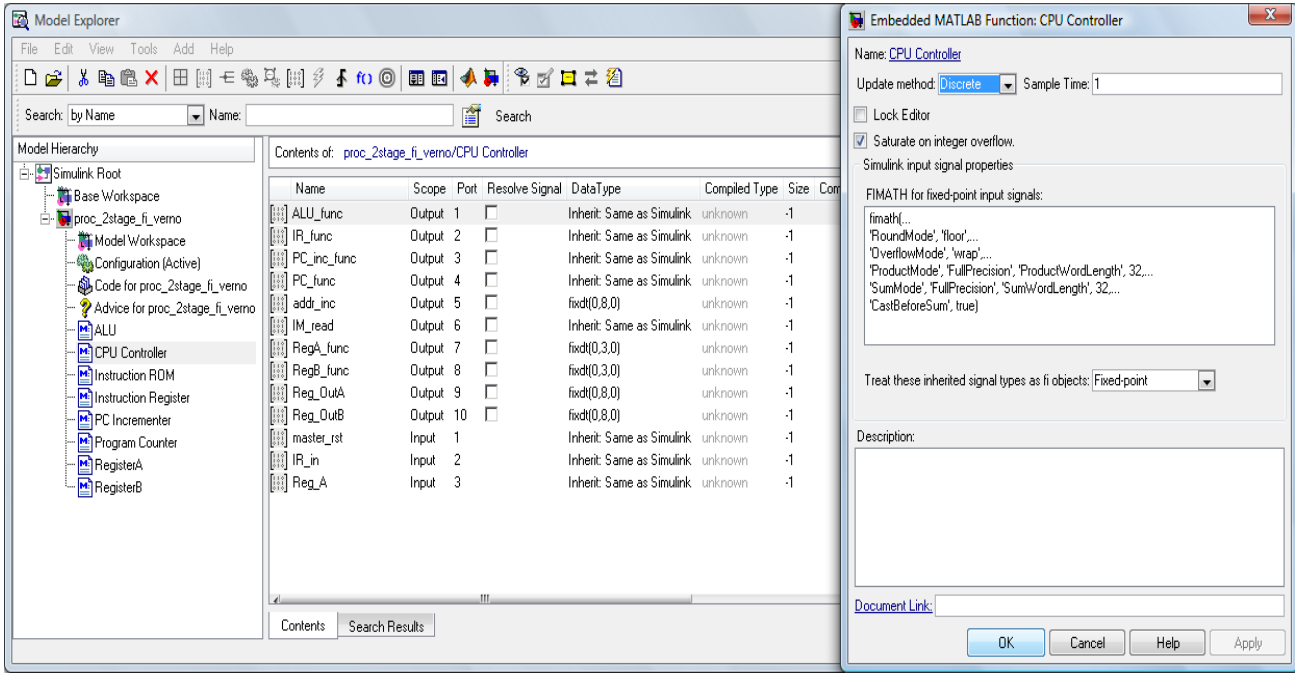


Рис.4.30. Настройка блоков модели процессора с помощью проводника модели для вычислений в формате с фиксированной запятой

Например, в блоке АЛУ локальный сигнал func 4 – х разрядный, десятичные числа с 0 по 8 кодируют логико-арифметические операции процессора, такие как ADD A,B; SUB A,B; AND A,B; OR A,B; XOR A,B и DEC и команды пересылки, такие как MOV A,B; MOV B,A; XCHG A,B. В блоках PC_Inc, PC и Instruction_Reg сигнал func 2 – х разрядный, а в блоках RegisterA и RegisterB 3 – х разрядный.

Пример 1 показывает М-файл функции управляющего автомата микропроцессора в системе Matlab/Simulink (блок CPU_Controller). Управляющий автомат может принимать 6 состояний. Состояния кодируются сигналом CPU_state в формате uint8 (целое десятичное число без знака с размером слова 8 бит). По сигналу master_rst (логическая 1), происходит установка автомата в нулевое состояние CPU_state = uint8(0). Далее происходит настройка блоков процессора с помощью локальных управляющих сигналов func.

```
PC_inc_func = fi(0, 0, 2, 0);
IR_func = fi(3, 0, 2, 0);
PC_func = fi(3, 0, 2, 0);
IM_read = fi(0, 0, 1, 0);
addr_inc = fi(0, 0, 8, 0);
Reg_OutA = fi(0, 0, 8, 0);
Reg_OutB = fi(0, 0, 8, 0);
RegA_func = fi(4, 0, 3, 0);
RegB_func = fi(4, 0, 3, 0);
ALU_func = fi(9, 0, 4, 0);
```

Управляющий автомат формирует на выходе PC_inc_func десятичный ноль, по которому внутреннее содержимое блока РСН будет сброшено (распознается блоком как сигнал сброса), на выходах PC_func и IR_func формируется десятичное число 3, по которому текущее содержимое счетчика команд и регистра инструкций остается неизменным.

На выходах RegA_func и RegB_func формируется десятичное число 4, по которому текущее содержимое регистров общего назначения РОН А и В, также остается

неизменным. На выходе ALU_func формируется десятичное число 9, по которому в блоке АЛУ произойдет обход логико-арифметических операций и команд пересылки, а значение сигналов на входах inA и inB будут переданы на выход outA и outB без изменений.

```
outA = fi(inA, 0, 8, 0);
outB = fi(inB, 0, 8, 0);
```

На выходах IM_read, addr_inc, Reg_OutA, Reg_OutB автомат формирует десятичные нули. Ноль на выходе IM_read запрещает чтение из ПЗУ программ. А десятичные нули на выходах addr_inc, Reg_OutA и Reg_OutB означают обнуление этих выходов.

В нулевом состоянии (case 0) осуществляется загрузка в РОН (блоки RegisterA, RegisterB), в РСН (блок PC_Inc) и в счетчик команд (блок PC) нуля (десятичный нуль преобразуется в формат с фиксированной запятой с размером слова 8 бит), а в регистр инструкций (Instruction_Reg) также загружается десятичный ноль, но он преобразуется в формат с фиксированной запятой с размером слова 16 бит. Эти операции осуществляются с помощью локальных сигналов управления PC_inc_func, PC_func, IR_func, RegA_func, RegB_func:

```
case 0,
    PC_inc_func = fi(0, 0, 2, 0);
    PC_func = fi(0, 0, 2, 0);
    IR_func = fi(0, 0, 2, 0);
    RegA_func = fi(0, 0, 3, 0);
    RegB_func = fi(0, 0, 3, 0);
    CPU_state = uint8(1);
```

Следующим состоянием автомата будет CPU_state = uint8(1). В этом состоянии и в двух последующих состояниях uint8(2) и uint8(3) происходит выделение полей команды. В состоянии 1 управляющий автомат формирует сигал

разрешения чтения команды из памяти IM_read = fi(1, 0, 1, 0). Поскольку порядковые номера строк в памяти программ начинаются с 1, например, data(1) = fi(1036, 0, 16, 0), то счетчик команд предварительно должен быть обнулен, т.е. нулевое значение счетчика указывает на строку в ПЗУ с порядковым номером 1.

Для того чтобы счетчик команд содержал адрес следующей команды, управляющий автомат должен сформировать локальный сигнал управления счетчиком PC_func = fi(2, 0, 2, 0), т.е. на выходе PC_func должно присутствовать десятичное число 2, по которому текущее значение счетчика увеличится на 1. Поэтому эта строка стоит второй в операторе case 1. Извлеченную команду (в первоначальный момент и в последующие, в регистре инструкций сохраняются текущие команды, а не следующие, загруженные в счетчик по команде PC_func = fi(2, 0, 2, 0)) из памяти программ в этом состоянии необходимо сохранить в регистре инструкций (16-битный регистр). Поэтому автомат сформирует локальный сигнал управления IR_func = fi(1, 0, 2, 0), разрешающий запись команды в регистр. Следующим состоянием, которое примет автомат, будет состояние CPU_state = uint8(2).

```
case 1,
    % Read from IM (ROM)
    IM_read = fi(1, 0, 1, 0);
    % PC increment PC+1
    PC_func = fi(2, 0, 2, 0);
    % store into IR
    IR_func = fi(1, 0, 2, 0);
    CPU_state = uint8(2);
```

Рассмотрим состояние 3 (case 3) управляющего автомата проектируемого процессора. Для того, что бы понять как работает формат с фиксированной запятой, необходимо последовательно копировать ниже приведенные строки

фрагмента М-файла и вставлять их в командную строку системы Matlab.

Например, рассмотрим, как обрабатывается команда 1536 (RET). Из регистра инструкций целое беззнаковое десятичное число 1536 (размер слова 16 бит) поступает на вход IR_in управляющего автомата CPU Controller и присваивается переменной main_opcode, которая представляет 16 – ти битную инструкцию. Из этой инструкции выделяется переменная major_opcode путем сдвига 16 – ти битного вектора вправо на 8 позиций, с размером слова в 4 бита, таким образом мы выделяем, биты с 9 по 12 из 16 – ти разрядной инструкции. В рассматриваемой системе команд разряды с 13 по 16 нулевые, поэтому выделение переменной minor_opcode путем побитного И переменной major_opcode (4 разряда) и маски (переменная mask4, 4 разряда) в принципе не обязательно, но не обходимо в случае последующей модификации системы команд процессора. Для выделения операнда (переменная address_data) из инструкции потребуется маска в 16 разрядов. Побитное И с переменной IR_in и с маской mask8 (0000000011111111) позволяет выделить переменную address_data с размером слова 8 бит. Для команды 1536 переменная address_data это 8 нулей. Следующим состоянием которое примет автомат будет состояние CPU_state = uint8(4).

```
IR_in=fi(1536,0,16,0);
main_opcode = fi(IR_in, 0, 16, 0);
disp(bin(main_opcode))
%0000011000000000
% Сдвиг вектора в право на 8 позиций
major_opcode= fi(bitsrl(main_opcode, 8), 0, 4, 0);
disp(bin(major_opcode))
%0110
mask4 = fi(15, 0, 4, 0);
disp(bin(mask4))
%1111
```

```
% Выделение команды, ширина поля 4 бита
minor_opcode = fi(bitand(major_opcode, mask4), 0, 4, 0);
disp(bin(minor_opcode))
%0110
% Выделение из команды операнда
mask8 = fi(255, 0, 16, 0);
disp(bin(mask8))
% 0000000011111111
address_data = fi(bitand(main_opcode, mask8), 0, 8, 0);
```

В состоянии 4 (case 4) происходит декодирование и выполнение инструкции (case 4). Декодирование происходит по сигналу minor_opcode (фактически 9, 10 и 11 биты сигнала IR_in, 12 бит не используется, т.к. он нулевой). Далее, декодируются 6 команд: NOP, JMP, JMPZ, CAL, MOV A,XX, MOV B,XX. Рассмотрим команду JMP. Выделенный операнд address_data из инструкции содержит адрес команды в ПЗУ на который необходимо перейти. Операнд присваевается переменной addr_inc. Автомат формирует локальные сигналы управления PCH - PC_inc_func (десятичное число 1) и счетчика команд - PC_func (десятичное число 1). Далее выделенный операнд (содержит адрес команды на который необходимо перейти) будет загружен в PCH и в счетчик команд. При загрузки операнда в PCH, содержимое счетчика команд при этом сохраняется во внутренней переменной PC_Temp данного регистра (пример 2).

```
case 4,
        switch(uint8(minor_opcode))
           case 0,
              % NOP
            CPU_state = uint8(1);
           case 1,
             % JMP
              temp_addr_data = fi(address_data, 0, 8, 0);
              addr_inc = fi(temp_addr_data, 0, 8, 0);
              PC_inc_func = fi(1, 0, 2, 0);
```

```
                PC_func = fi(1, 0, 2, 0);
                CPU_state = uint8(1);
……
……
            case 6,
                switch(uint8(address_data))
                    case 0,
                        %RET
                        PC_inc_func = fi(2, 0, 2, 0);
                        PC_func = fi(2, 0, 2, 0);
                        CPU_state = uint8(5);
```

Если ни одна их этих команд не выполняется, то далее дешифрируются и отрабатываются команда RET, логико-арифметические команды (ADD A,B, OR A,B, XOR A,B, DEC A) и команды пересылки (MOV A,B, MOV B,A, XCHG A,B,). Последним состоянием является состояние case 5. В этом состоянии обновляются регистры РОН A и B, затем будет осущетвлен переход в состояние 1. И весь описанный выше процесс обработки команды повторится вновь и то тех пор, пока не будет отработана последняя команда в программе.

```
function [ALU_func, IR_func, PC_inc_func, PC_func, ...
   addr_inc, IM_read, RegA_func, RegB_func, ...
   Reg_OutA, Reg_OutB] = CPU_Controller(master_rst, IR_in, Reg_A)

% CPU Controller
% 16-bit Instruction Encoding:
% ------------minor_opcode---------
%   NOP:            00000 000 <00000000>
%   JMP:            00000 001 <8-bit>
%   JMPZ:           00000 010 <8-bit>
%   CALL:           00000 011 <8-bit>
%   MOV A,xx:   00000 100 <8-bit>
%   MOV B,xx:   00000 101 <8-bit>
% --------------------------------
%   RET:            00000 110 <00000000>
%   MOV A,B:    00000 110 <00000001>
```

```
%   MOV B,A:    00000 110 <00000010>
%   XCHG A,B:  00000 110 <00000011>
%   ADD A,B:    00000 110 <00000100>
%   SUB A,B:    00000 110 <00000101>
%   AND A,B:    00000 110 <00000110>
%   OR A,B:     00000 110 <00000111>
%   XOR A,B:    00000 110 <00001000>
%   DEC A:      00000 110 <00001001>

persistent CPU_state;
if(isempty(CPU_state))
   CPU_state = uint8(0);
end

if(master_rst)
   CPU_state = uint8(0);
end

PC_inc_func = fi(0, 0, 2, 0);
IR_func = fi(3, 0, 2, 0); % NOP
PC_func = fi(3, 0, 2, 0); % NOP
IM_read = fi(0, 0, 1, 0);
addr_inc = fi(0, 0, 8, 0);
Reg_OutA = fi(0, 0, 8, 0);
Reg_OutB = fi(0, 0, 8, 0);
RegA_func = fi(4, 0, 3, 0); % NOP
RegB_func = fi(4, 0, 3, 0); % NOP
ALU_func = fi(9, 0, 4, 0); % NOP

% main_code: <16..1>
% major_opcode: <16..9>
% minor_opcode: <12..9>
% address_data: <8..1>

persistent main_opcode;
persistent major_opcode;
persistent minor_opcode;
persistent address_data;
```

```
if(isempty(major_opcode))
   main_opcode = fi(0, 0, 16, 0);
   major_opcode = fi(0, 0, 4, 0);
   minor_opcode = fi(0, 0, 4, 0);
   address_data = fi(0, 0, 8, 0);
end
switch(CPU_state)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %  RESETTING OUTPUTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   case 0,
      PC_inc_func = fi(0, 0, 2, 0);
      PC_func = fi(0, 0, 2, 0);
      IR_func = fi(0, 0, 2, 0);
      RegA_func = fi(0, 0, 3, 0);
      RegB_func = fi(0, 0, 3, 0);
      CPU_state = uint8(1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %  FETCH
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   case 1,
       % Read from IM (ROM)
      IM_read = fi(1, 0, 1, 0);
       % PC increment PC+1
      PC_func = fi(2, 0, 2, 0);
      % store into IR
      IR_func = fi(1, 0, 2, 0);
      CPU_state = uint8(2);
   case 2,
      % Read from IR
      IR_func = fi(2, 0, 2, 0);
      % Accommodating for the 'unit delay' from IR_out to IR_in
      CPU_state = uint8(3);
   case 3,
      % IR_in <16..1>
      main_opcode = fi(IR_in, 0, 16, 0);
      % IR_in <16..9>
      major_opcode = fi(bitsrl(main_opcode, 8), 0, 4, 0);
```

```
        % for instructions NOP,JMP,JMPZ,CALL,MOV A,xx MOV
B,xx,RET
        % IR_in <12..9>
        mask4 = fi(15, 0, 4, 0);
        minor_opcode = fi(bitand(major_opcode, mask4), 0, 4, 0);
        % IR_in <8..1>
        mask8 = fi(255, 0, 16, 0);
        address_data = fi(bitand(main_opcode, mask8), 0, 8, 0);
        % Go to the decode stage
        CPU_state = uint8(4);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %  DECODE AND EXECUTE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    case 4,
            switch(uint8(minor_opcode))
              case 0,
                  % NOP
                CPU_state = uint8(1);
              case 1,
                 % JMP
                  temp_addr_data = fi(address_data, 0, 8, 0);
                  addr_inc = fi(temp_addr_data, 0, 8, 0);
                  PC_inc_func = fi(1, 0, 2, 0);
                  PC_func = fi(1, 0, 2, 0);
                  CPU_state = uint8(1);
              case 2,
                  %JMPZ
                  temp_addr_data = fi(address_data, 0, 8, 0);
                  if fi(Reg_A,0,8,0) == fi(0,0,8,0)
                  addr_inc = fi(temp_addr_data, 0, 8, 0);
                  PC_inc_func = fi(1, 0, 2, 0);
                  PC_func = fi(1, 0, 2, 0);
                  end
                  CPU_state = uint8(1);
              case 3,
                % CALL
                  temp_addr_data = fi(address_data, 0, 8, 0);
                  addr_inc = fi(temp_addr_data, 0, 8, 0);
                  PC_inc_func = fi(1, 0, 2, 0);
```

```
    PC_func = fi(1, 0, 2, 0);
    CPU_state = uint8(1);
case 4,
  %MOV A,xx
    temp_addr_data = fi(address_data, 0, 8, 0);
    Reg_OutA = fi(temp_addr_data , 0, 8, 0);
    RegA_func = fi(1, 0, 3, 0);
    CPU_state = uint8(1);
case 5,
   %MOV B,xx
   temp_addr_data = fi(address_data, 0, 8, 0);
    Reg_OutB = fi(temp_addr_data , 0, 8, 0);
    RegB_func = fi(1, 0, 3, 0);
    CPU_state = uint8(1);
case 6,
     switch(uint8(address_data))
        case 0,
             %RET
              PC_inc_func = fi(2, 0, 2, 0);
              PC_func = fi(2, 0, 2, 0);
              CPU_state = uint8(5);
        case 1,
             %MOV A,B
             ALU_func = fi(0, 0, 4, 0);
             RegA_func = fi(2, 0, 3, 0);
             RegB_func = fi(2, 0, 3, 0);
             CPU_state = uint8(5);
        case 2,
              %MOV B,A
             ALU_func = fi(1, 0, 4, 0);
             RegA_func = fi(2, 0, 3, 0);
             RegB_func = fi(2, 0, 3, 0);
             CPU_state = uint8(5);
        case 3,
              %XCHG A,B
             ALU_func = fi(2, 0, 4, 0);
             RegA_func = fi(2, 0, 3, 0);
             RegB_func = fi(2, 0, 3, 0);
             CPU_state = uint8(5);
```

```
            case 4,
                 %ADD A,B
                 ALU_func = fi(3, 0, 4, 0);
                 RegA_func = fi(2, 0, 3, 0);
                 RegB_func = fi(2, 0, 3, 0);
                 CPU_state = uint8(5);
            case 5,
               %SUB A,B
                 ALU_func = fi(4, 0, 4, 0);
                 RegA_func = fi(2, 0, 3, 0);
                 RegB_func = fi(2, 0, 3, 0);
                 CPU_state = uint8(5);
            case 6,
               %AND A,B
                 ALU_func = fi(5, 0, 4, 0);
                 RegA_func = fi(2, 0, 3, 0);
                 RegB_func = fi(2, 0, 3, 0);
                 CPU_state = uint8(5);
            case 7,
               %OR A,B
                 ALU_func = fi(6, 0, 4, 0);
                 RegA_func = fi(2, 0, 3, 0);
                 RegB_func = fi(2, 0, 3, 0);
                 CPU_state = uint8(5);
            case 8,
               %XOR A,B
                 ALU_func = fi(7, 0, 4, 0);
                 RegA_func = fi(2, 0, 3, 0);
                 RegB_func = fi(2, 0, 3, 0);
                 CPU_state = uint8(5);
           case 9,
              %DEC A
                 ALU_func = fi(8, 0, 4, 0);
                 RegA_func = fi(2, 0, 3, 0);
                 CPU_state = uint8(5);
        end
    end
case 5,
    RegA_func = fi(2, 0, 3, 0);
```

```
            RegB_func = fi(2, 0, 3, 0);
            CPU_state = uint8(1);
end
```

Пример 1. М-файл функции управляющего автомата микропроцессора (CPU_Controller) в системе Matlab/Simulink

```
function [PC_next,Temp] = PC_Inc(func, addr, PC_curr)

% func = 0 => reset PC_Inc
% func = 1 => store into PC_Inc when JMP, JMPZ, CALL
% func = 2 => load from PC_Inc when RET

persistent PC_Temp;
if(isempty(PC_Temp))
  PC_Temp = fi(0, 0, 8, 0);
end
PC_next = fi(PC_curr, 0, 8, 0);
Temp = fi(0, 0, 8, 0);
switch(uint8(func))
   case 0,
      % reset PC_Inc
      PC_next = fi(0, 0, 8, 0);
   case 1,
      % store into PC_Inc when JMP, JMPZ, CALL
        PC_next = fi(addr, 0, 8, 0);
        PC_Temp = fi(PC_curr, 0, 8, 0);
        Temp = fi(PC_Temp, 0, 8, 0);
   case 2,
      % load from PC_Inc when RET
      PC_next = fi(PC_Temp, 0, 8, 0);
end
```

Пример 2. М-файл функции блока специального назначения (PC_Inc) в системе Matlab/Simulink

```
function addr_out = PC(func, addr_in)

% Program Counter
% func = 0 => reset PC
```

```
% func = 1 => load PC
% func = 2 => increment PC

persistent PC_value;
if(isempty(PC_value))
   PC_value = fi(0, 0, 8, 0);
end
addr_out = fi(PC_value, 0, 8, 0);
switch(uint8(func))
   case 0,
      % reset
      PC_value = fi(0, 0, 8, 0);
   case 1,
      % store into PC
      PC_value = fi(addr_in, 0, 8, 0);
   case 2,
      % increment PC
      PC_value = fi(PC_value + 1, 0, 8, 0);
end
```

Пример 3. М-файл функции блока счетчика команд (PC) в системе Matlab/Simulink

```
function instr_out = Memory(addr,read)
persistent data;
if(isempty(data))
data = fi(zeros(1, 256), 0, 16, 0);
end
data(1) = fi(1036, 0, 16, 0);
data(2) = fi(1303, 0, 16, 0);
data(3) = fi(1540, 0, 16, 0);
data(4) = fi(1545, 0, 16, 0);
data(5) = fi(1358, 0, 16, 0);
data(6) = fi(1539, 0, 16, 0);
data(7) = fi(1541, 0, 16, 0);
data(8) = fi(1545, 0, 16, 0);
data(9) = fi(1542, 0, 16, 0);
data(10) = fi(523, 0, 16, 0);
data(11) = fi(263, 0, 16, 0);
data(12) = fi(1037, 0, 16, 0);
```

```
data(13) = fi(1397, 0, 16, 0);
data(14) = fi(1543, 0, 16, 0);
data(15) = fi(1539, 0, 16, 0);
data(16) = fi(1544, 0, 16, 0);
data(17) = fi(277, 0, 16, 0);
data(18) = fi(1135, 0, 16, 0);
data(19) = fi(1480, 0, 16, 0);
data(20) = fi(1542, 0, 16, 0);
data(21) = fi(1536, 0, 16, 0);
data(22) = fi(785, 0, 16, 0);
data(23) = fi(0, 0, 16, 0);
if(read == 1)
   instr_out = data(addr+1);
else
   instr_out = fi(0, 0, 16, 0);
end
```

Пример 4. М-файл функции блока памяти программ (Memory) в системе Matlab/Simulink

```
function IR_out = Instruction_Reg(func, IR_in)
% A 16-bit Instruction Register with the following func:
% func == 0 => reset
% func == 1 => store into IR
% func == 2 => read from IR
% otherwise, preserve old value and return 0
persistent IR_value;
if(isempty(IR_value))
   IR_value = fi(0, 0, 16, 0);
end
IR_out = fi(0, 0, 16, 0);
switch(uint8(func))
   case 0,
      % reset
      IR_value = fi(0, 0, 16, 0);
   case 1,
      % store into IR
      IR_value = fi(IR_in, 0, 16, 0);
   case 2,
      % read IR
```

```
    IR_out = fi(IR_value, 0, 16, 0);
end
```

Пример 5. М-файл функции блока регистра инструкций (Instruction_Reg) в системе Matlab/Simulink

```
function Reg_out_A = RegisterA(func, Reg_in_A_1, Reg_in_A_2)
% func == 0 => reset;
% func == 1 => store into RegisterA from port 1;
% func == 2 => store into RegisterA from port 2;
% func == 3 => read from RegisterA;
% HDL specific fimath
hdl_fm = fimath(...
'RoundMode', 'floor',...
'OverflowMode', 'wrap',...
'ProductMode', 'FullPrecision', 'ProductWordLength', 32,...
'SumMode', 'FullPrecision', 'SumWordLength', 32,...
'CastBeforeSum', true);
persistent Reg_value;
if(isempty(Reg_value))
   Reg_value = fi(0, 0, 8, 0, hdl_fm);
end
Reg_out_A = fi(Reg_value, 0, 8, 0, hdl_fm);
switch(uint8(func))
   case 0,
      % reset
      Reg_value = fi(0, 0, 8, 0, hdl_fm );
   case 1,
      % store into Reg_A from port 1
      Reg_value = Reg_in_A_1;
   case 2,
      % store into Reg_A from port 2
      Reg_value = Reg_in_A_2;
   case 3,
      % read Reg_A
      Reg_out_A = Reg_value;
end
```

Пример 6. М-файл функции блока регистра общего назначения A (RegisterA) в системе Matlab/Simulink

```
function [outA, outB] = alu(func,inA,inB)
% This 8-bit ALU supports the following operations:
% MOV, XCHG, ADD, SUB, AND, OR, XOR, DEC
% func = 0 => MOV A,B
% func = 1 => MOV B,A
% func = 2 => XCHG A,B
% func = 3 => ADD A,B
% func = 4 => SUB A,B
% func = 5 => AND A,B
% func = 6 => OR A,B
% func = 7 => XOR A,B
% func = 8 => DEC A
% Simply pass the inA, when there is no designated func
outA = fi(inA, 0, 8, 0);
% Simply pass the inB, when there is no designated func
outB = fi(inB, 0, 8, 0);
switch (uint8(func))
            case 0, %MOV A,B
                 outA = fi(inB, 0, 8, 0);
            case 1, %MOV B,A
                 outB = fi(inA, 0, 8, 0);
            case 2, %XCHG A,B
                X_temp = fi(inB, 0, 8, 0);
                outB = fi(inA, 0, 8, 0);
                outA = fi(X_temp, 0, 8, 0);
            case 3, %ADD A,B
                outA = fi(inA + inB, 0, 8, 0);
            case 4, %SUB A,B
                outA = fi(inA - inB, 0, 8, 0);
            case 5, %AND A,B
                outA = fi(bitand(inA,inB), 0, 8, 0);
            case 6, %OR A,B
                outA = fi(bitor(inA,inB), 0, 8, 0);
            case 7, %XOR A,B
                outA = fi(bitxor(inA,inB), 0, 8, 0);
            case 8, %DEC A
                outA = fi(inA - 1, 0, 8, 0);
end
```

Пример 7. М-файл функции блока АЛУ в системе Matlab/Simulink

На рис.4.31 показаны временные диаграммы работы процессора с управляющим автоматом в системе Matlab/Simulink. По оси y откладываются целые беззнаковые десятичные числа (которые преобразуются в процессе вычислений в формат с фиксированной запятой), а по оси x время моделирования.

На рис.4.31, а, видно, что значения накопленные счетчиком команд непрерывно увеличиваются и только в случае команды JMP 7 (команда выполняется в программе 3 раза), счетчик изменяет свое значение, на значение операнда, содержащееся в команде, т.е. на 7. На рис.4.31, б показано содержимое блока РСН, на рис.4.31, в – содержимое памяти программ; на рис.4.31, г – содержимое РОН А.

В системе Matlab/Simulink разработан учебный вариант 8-ми разрядного процессора, позволяющего проводить вычисления в формате с фиксированной запятой, с управляющим автоматом на шесть состояний. Преимуществом такой архитектуры является ее адаптивность к последующим модификациям, например, в случае если потребуется добавить дополнительные команды. Недостатком является отсутствие памяти данных, поддержка незначительного числа команд, а также то, что процессор оперирует только с целыми положительными числами и отсутствие конвейера команд.

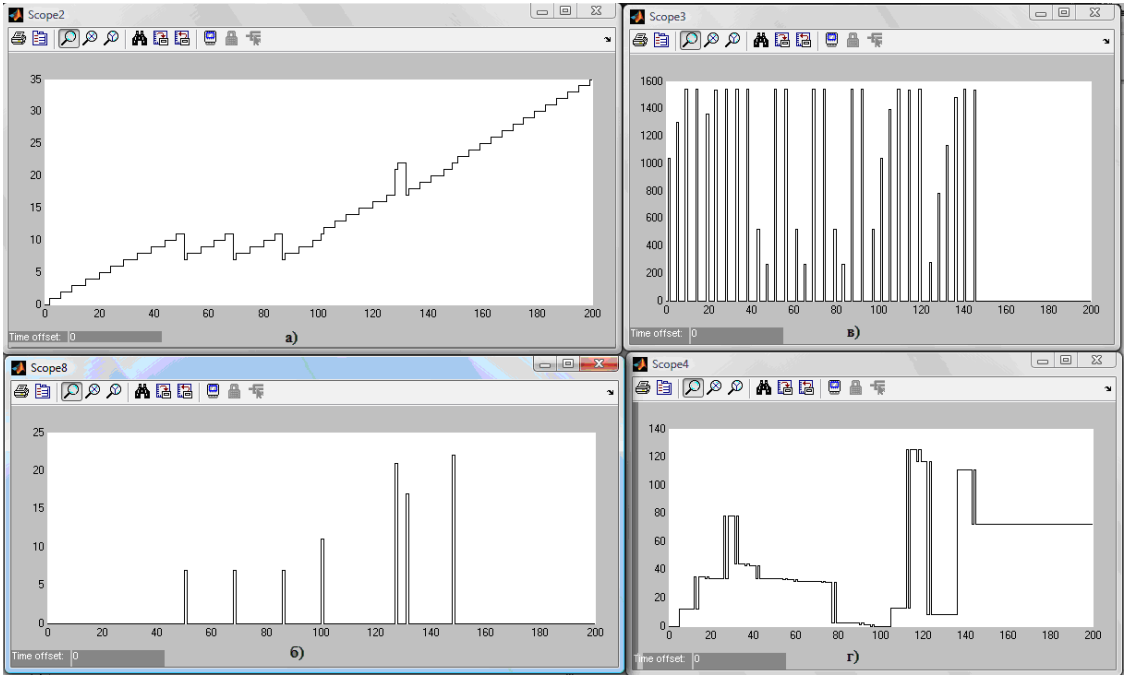


Рис.4.31. Временные диаграммы работы процессора с управляющим автоматом в системе Matlab/Simulink: а – счетчик команд; б – блок специального назначения; в – память программ; г – РОН А

4.5. Проектирование учебного процессора с фиксированной запятой в САПР ПЛИС Quartus II

В данном разделе предлагается на основе системы команд процессора с циклом работы в два такта и на основе модели процессора с управляющим автоматом на шесть состояний, позволяющим проводить вычисления с фиксированной зяпятой, реализованной в системе Matlab/Simulink (раздел 4.4), разработать процессор в базисе ПЛИС Stratix III компании Altera с использованием САПР Quartus II. Особенностью выше рассмотренной модели процессора является распределенная система управления функциональными блоками, т.е. каждый блок имеет свой локальный управляющий сигнал (шину), которым управляет цифровой автомат. Основные функциональные блоки проектируемого процессора описаны на языке VHDL, код языка которого был сгенерирован в автоматическом режиме с помощью Simulink HDL Coder системы Matlab/Simulink. На рис.4.32 представлена электрическая схема процессора в САПР ПЛИС Quartus II версии 8.1.

Проектируемый процессор состоит из следующих функциональных блоков (рис.4.32): управляющий автомат (блок CPU_Controller, пример 1); регистр специального назначения (РСН, блок PC_Incrementer, пример 2), необходим для обеспечения “прыжковых” команд, таких как JMP, JMPZ, CALL и RET; счетчик команд (блок Program_Counter, пример 3); память программ - ПЗУ процессора (блок Instruction_ROM, пример 4); регистра инструкций (блок Instruction_Register, пример 5); двух регистров общего назначения (РОН, блок RegisterA, пример 6); АЛУ процессора (блок ALU, пример 7). Регистры на D-триггерах (четыре 8-ми (блок dff8) и один 16-ти разрядный (блок dff16)), тактируемые фронтом синхросигнала разработаны дополнительно (см. раздел 4.4) с использованием мегафункции LPM_FF.

На рис.4.33 и рис.4.34 показаны временные диаграммы работы процессора с управляющим автоматом в САПР Quartus II. Осуществляется тестирование команд MOV A,12; MOV B,23; ADD A,B и команд JMPZ11, JMP7.

Язык VHDL является языком со строгим контролем типов. Поэтому бывает необходимо преобразовать сигнал одного типа в сигнал другого типа. Даже при выполнении простых действий, если типы объектов не совпадают, может потребоваться обращение к функции преобразования типов. Различают два вида преобразования типа: переход типа и вызов функции преобразования типа. Переход типа применяется для преобразования тесно связанных типов или подтипов. Если типы не тесно связанные, то необходимо выполнить вызов функции преобразования типа.

Пакет numeric_std содержит стандартный набор арифметических, логических функций и функций сравнения для работы с типами signed, unsigned, integer, std_ulogic, std_logic, std_logic_vector. В пакете numeric_std существует функция преобразования векторного типа to_unsigned с операндами arg (тип integer), size (тип natural, битовая ширина) и типом результата unsigned. Тип unsigned интерпретируется как двоичное представление числа без знака, а тип signed обозначает двоичные числа со знаком в дополнительном коде. Например: CPU_state_temp := to_unsigned(3, 8);. Число 3 из одномерного массива целых чисел преобразуется в восьми разрядное двоичное число без знака, которое присваивается переменной CPU_state_temp. А переменная CPU_state_temp объявлена как массив двоичных чисел без знака: VARIABLE CPU_state_temp: unsigned(7 DOWNTO 0);. Это есть явное преобразование тесно связанных между собой типов.

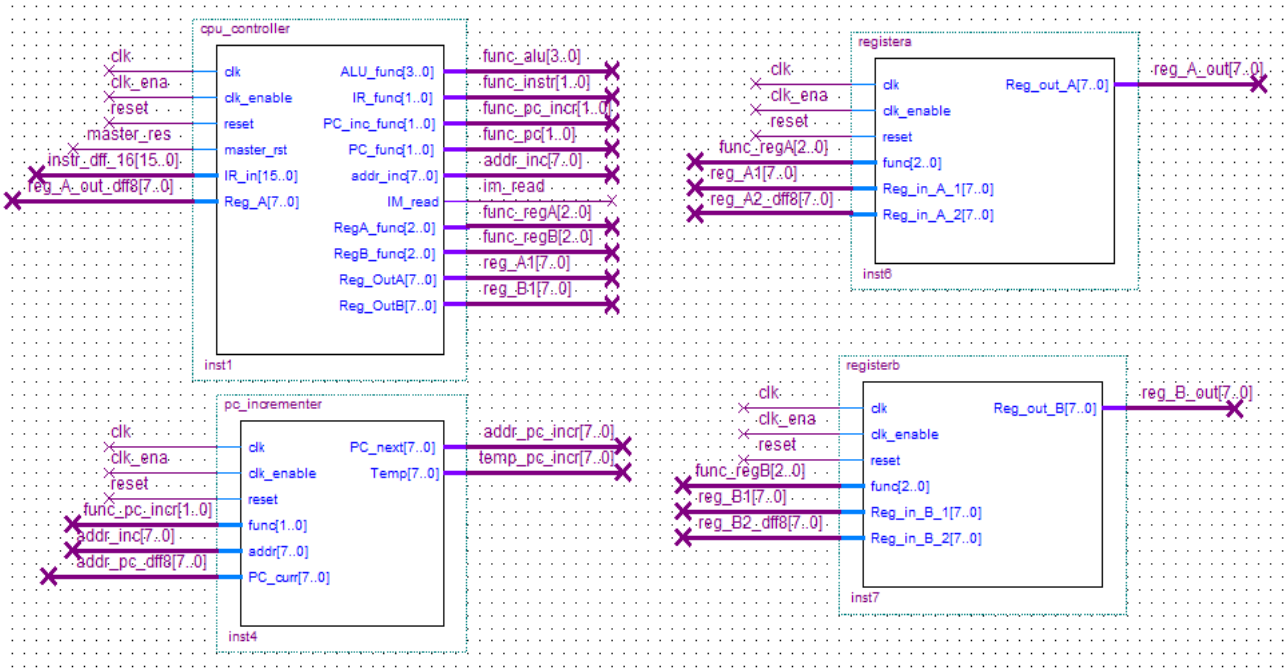


Рис.4.32. Схема процессора с управляющим автоматом для вычислений в формате с фиксированной запятой в САПР Quartus II, построенная с использованием модели разработанной в системе Matlab/Simulink

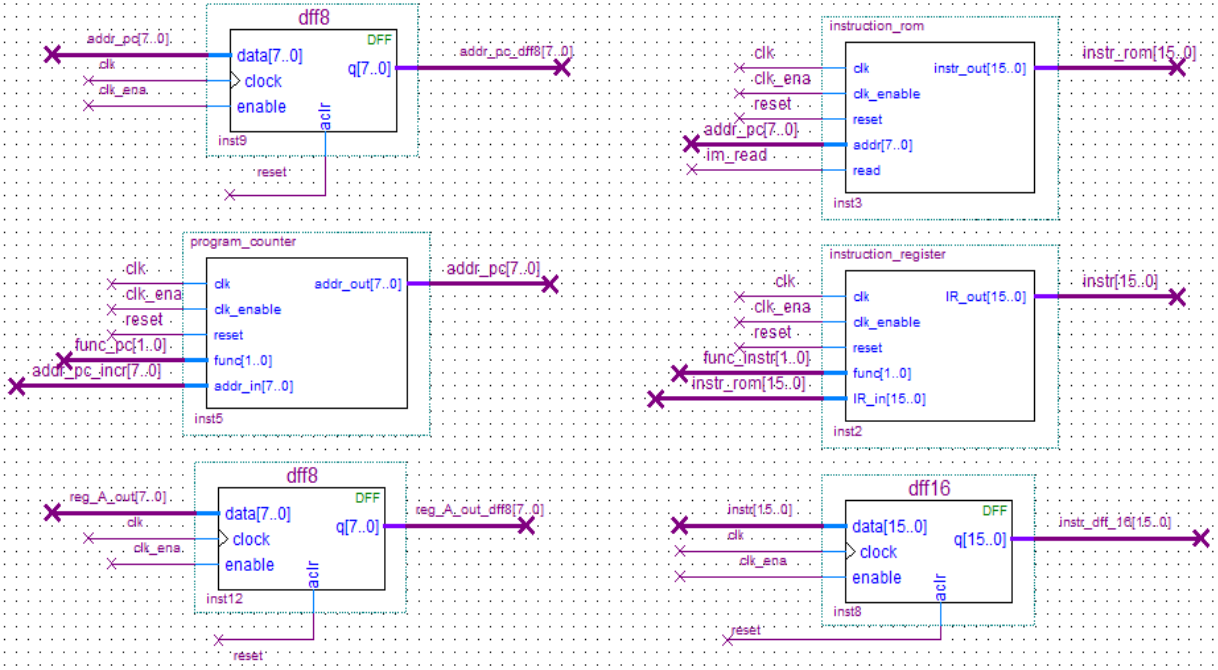


Рис.4.32. Схема процессора с управляющим автоматом для вычислений в формате с фиксированной запятой в САПР Quartus II, построенная с использованием модели разработанной в системе Matlab/Simulink (продолжение)

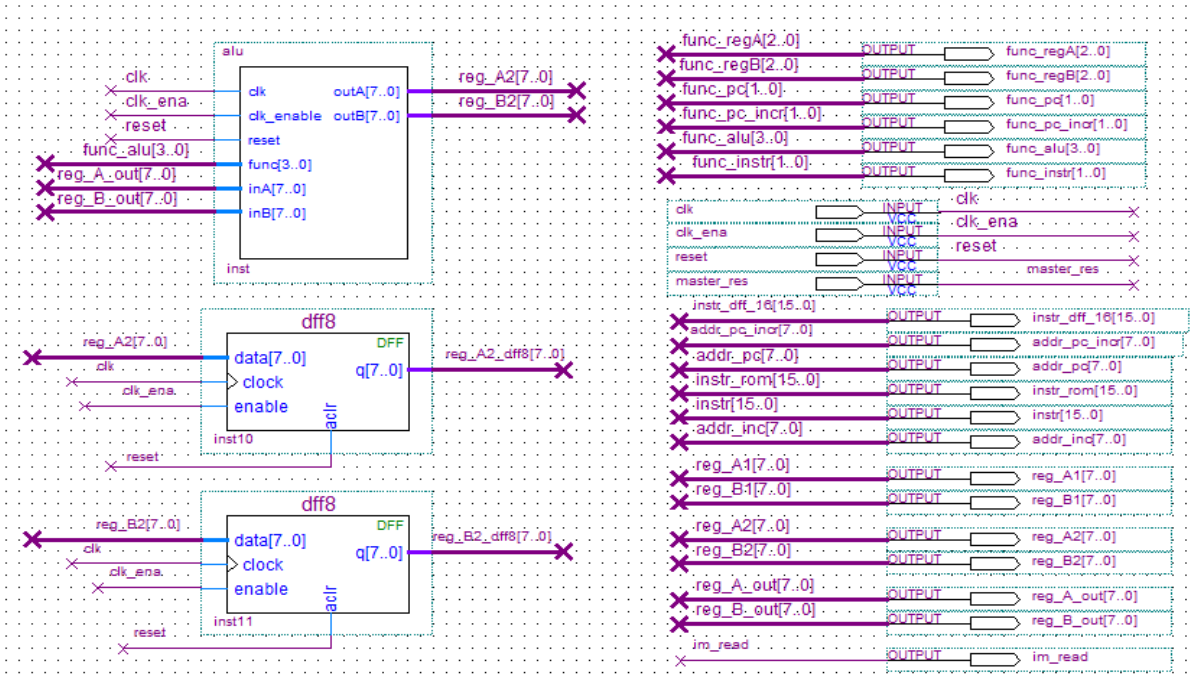


Рис.4.32. Схема процессора с управляющим автоматом для вычислений в формате с фиксированной запятой в САПР Quartus II, построенная с использованием модели разработанной в системе Matlab/Simulink (окончание)

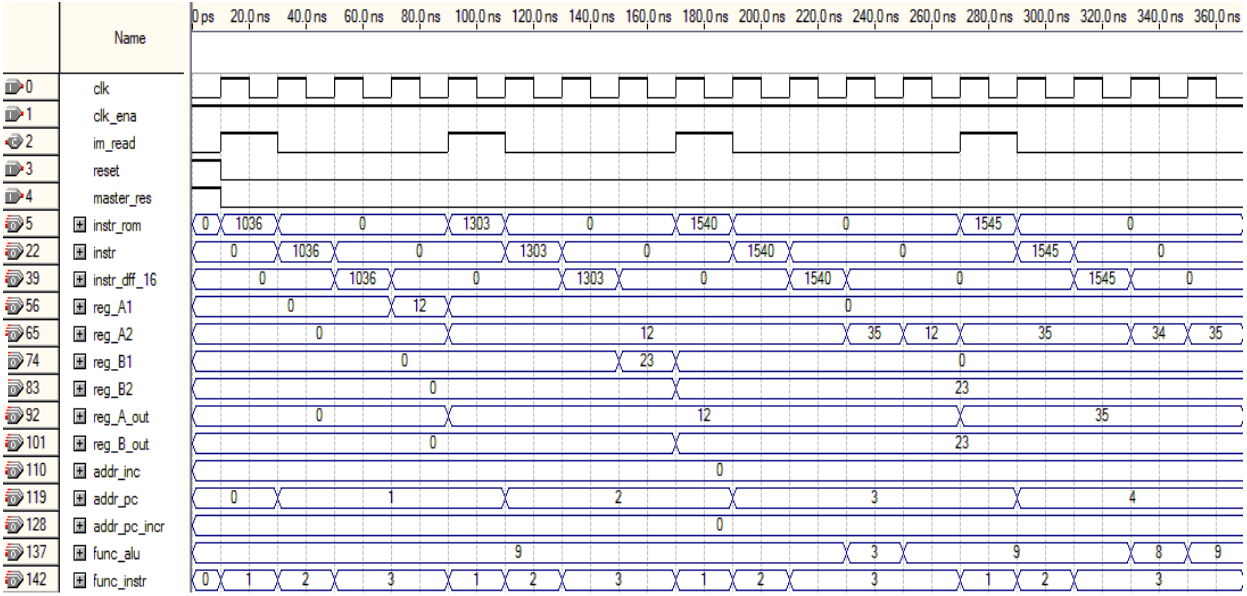


Рис.4.33. Временные диаграммы работы процессора с управляющим автоматом в САПР Quartus II. Тестирование команд MOV A,12; MOV B,23; ADD A,B

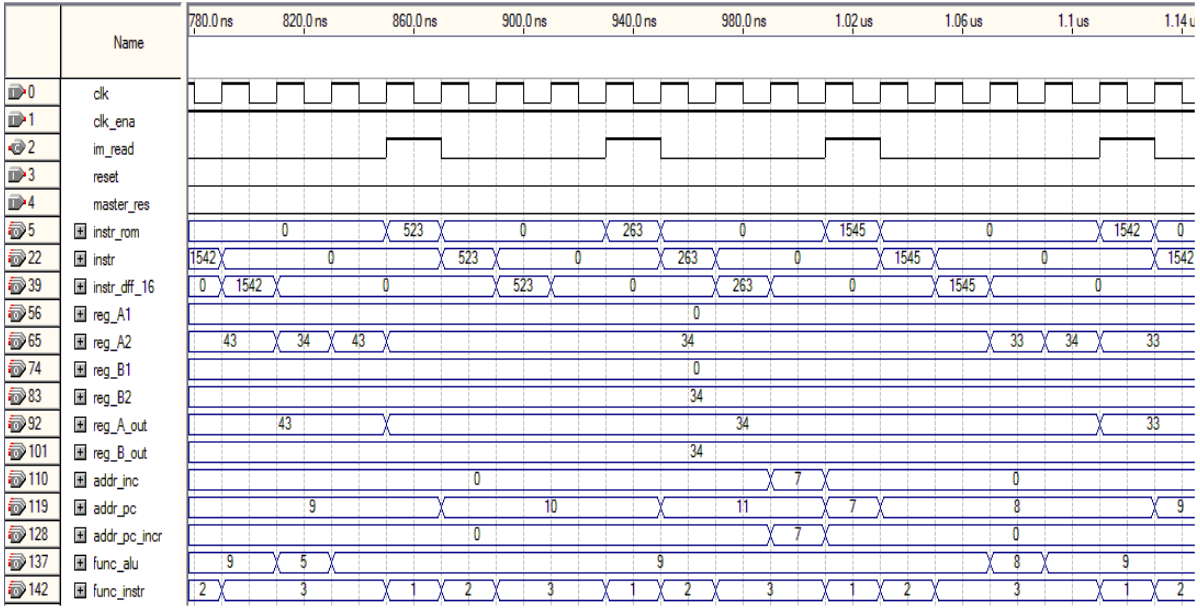


Рис.4.34. Временные диаграммы работы процессора с управляющим автоматом в САПР Quartus II. Тестирование команд JMPZ11, JMP7

Или IR_func <= std_logic_vector(to_unsigned(3, 2));. Десятичное число 3 преобразуется в двоичное число “11” типа unsigned, затем тип unsigned неявно преобразуется в тип std_logic_vector. Сигналу IR_func будет назначено двоичное число “11”.

При генерации кода языка VHDL блока АЛУ используется дополнительная функция tmw_to_signed, которая преобразует двоичное число типа unsigned в двоичное число типа signed (пример 7) с шириной битовой шины типа integer, что необходимо для обеспечения операции вычитания (вызов функции преобразования типа):

```
FUNCTION tmw_to_signed(arg: unsigned; width: integer) RETURN
signed IS
--SUB A,B
ina_1 := tmw_to_signed(unsigned(inA), 9) - tmw_to_signed(unsigned(inB), 9);
```

Оператор srl введенный в стандарте VHDL93 осуществляет операцию логического сдвига одномерного массива (левый оператор) с элементами типа bit в право, на число указанное правым оператором типа integer. Например, одномерный массив main_opcode_temp типа unsigned(15 DOWNTO 0), представляющий из себя 16-ти битовое двоичное число, сдвигается вправо на 8 бит. Например:

```
main_opcode_temp := unsigned(IR_in); cr := main_opcode_temp srl 8;.
```

Логический оператор AND (тип левого операнда, правого и тип результата unsigned) используется для выделения 8-ми разрядного сигнала (операнда) address_data_next из 16-ти разрядной команды микропроцессора путем логического умножения сигнала major_opcode_temp с маской to_unsigned(255, 16). Например:

```
-- IR_in <8..1>
```

```
c_uint := main_opcode_temp AND to_unsigned(255, 16);
IF c_uint(15 DOWNTO 8) /= "00000000" THEN
   address_data_next <= "11111111";
ELSE
   address_data_next <= c_uint(7 DOWNTO 0);
END IF;
```

В примере 3 используется оператор конкатенации & который предопределен для всех одноразмерных массивов. Этот оператор выстраивает массивы путем комбинирования с их операндами. Оператор & используется для добавления одиночного элемента в конец массива PC_value типа unsigned(7 DOWNTO 0):

```
-- increment PC
ain := resize(PC_value & '0' & '0' & '0' & '0' & '0' & '0' & '0', 16);
ain_0 := ain + 128;
IF (ain_0(15) /= '0') OR (ain_0(14 DOWNTO 7) = "11111111") THEN
   PC_value_next <= "11111111";
ELSE
PC_value_next <= ain_0(14 DOWNTO 7) + ("0" & (ain_0(6)));
END IF;
```

Функция изменения размера resize (тип левого оператора unsigned, колличество позиций типа natural, тип результата unsigned) позволяет из восьми разрядного сигнала PC_value сконструировать локальную переменную ain типа unsigned(15 DOWNTO 0). Функция ‘+’ позволяет осуществить ариметическую операцию сложения, если тип левого оператора unsigned а правого integer с результатом unsigned.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY CPU_Controller IS
   PORT (
      clk : IN std_logic;
```

```
      clk_enable : IN std_logic;
      reset : IN std_logic;
      master_rst : IN std_logic;
      IR_in : IN std_logic_vector(15 DOWNTO 0);
      Reg_A : IN std_logic_vector(7 DOWNTO 0);
      ALU_func : OUT std_logic_vector(3 DOWNTO 0);
      IR_func : OUT std_logic_vector(1 DOWNTO 0);
      PC_inc_func : OUT std_logic_vector(1 DOWNTO 0);
      PC_func : OUT std_logic_vector(1 DOWNTO 0);
      addr_inc : OUT std_logic_vector(7 DOWNTO 0);
      IM_read : OUT std_logic;
      RegA_func : OUT std_logic_vector(2 DOWNTO 0);
      RegB_func : OUT std_logic_vector(2 DOWNTO 0);
      Reg_OutA : OUT std_logic_vector(7 DOWNTO 0);
      Reg_OutB : OUT std_logic_vector(7 DOWNTO 0));
END CPU_Controller;

ARCHITECTURE fsm_SFHDL OF CPU_Controller IS

   SIGNAL CPU_state : unsigned(7 DOWNTO 0);
   SIGNAL major_opcode : unsigned(3 DOWNTO 0);
   SIGNAL main_opcode : unsigned(15 DOWNTO 0);
   SIGNAL minor_opcode : unsigned(3 DOWNTO 0);
   SIGNAL address_data : unsigned(7 DOWNTO 0);
   SIGNAL CPU_state_next : unsigned(7 DOWNTO 0);
   SIGNAL major_opcode_next : unsigned(3 DOWNTO 0);
   SIGNAL main_opcode_next : unsigned(15 DOWNTO 0);
   SIGNAL minor_opcode_next : unsigned(3 DOWNTO 0);
   SIGNAL address_data_next : unsigned(7 DOWNTO 0);
BEGIN
   initialize_CPU_Controller : PROCESS (reset, clk)
      -- local variables
   BEGIN
      IF reset = '1' THEN
         CPU_state <= to_unsigned(0, 8);
         main_opcode <= to_unsigned(0, 16);
         major_opcode <= to_unsigned(0, 4);
```

```
            minor_opcode <= to_unsigned(0, 4);
            address_data <= to_unsigned(0, 8);
        ELSIF clk'EVENT AND clk= '1' THEN
            IF clk_enable= '1' THEN
                CPU_state <= CPU_state_next;
                major_opcode <= major_opcode_next;
                main_opcode <= main_opcode_next;
                minor_opcode <= minor_opcode_next;
                address_data <= address_data_next;
            END IF;
        END IF;
    END PROCESS initialize_CPU_Controller;

    CPU_Controller : PROCESS (CPU_state, major_opcode,
main_opcode, minor_opcode,
address_data, master_rst, IR_in, Reg_A)
        -- local variables
        VARIABLE c_uint : unsigned(15 DOWNTO 0);
        VARIABLE b_c_uint : unsigned(3 DOWNTO 0);
        VARIABLE cr : unsigned(15 DOWNTO 0);
        VARIABLE CPU_state_temp : unsigned(7 DOWNTO 0);
        VARIABLE major_opcode_temp : unsigned(3 DOWNTO 0);
        VARIABLE main_opcode_temp : unsigned(15 DOWNTO 0);
        VARIABLE reg_a_0 : unsigned(7 DOWNTO 0);
    BEGIN
        minor_opcode_next <= minor_opcode;
        address_data_next <= address_data;
        CPU_state_temp := CPU_state;
        major_opcode_temp := major_opcode;
        main_opcode_temp := main_opcode;
        -- CPU Controller
        -- 16-bit Instruction Encoding:
        -- ------------minor_opcode---------
        --  NOP:          00000 000 <00000000>
        --  JMP:          00000 001 <8-bit>
        --  JMPZ:         00000 010 <8-bit>
        --  CALL:         00000 011 <8-bit>
```

```
--  MOV A,xx:  00000 100 <8-bit>
--  MOV B,xx:  00000 101 <8-bit>
--  RET:           00000 110 <00000000>
-- -------------------------------------
--  MOV A,B:    00000 110 <00000001>
--  MOV B,A:    00000 110 <00000010>
--  XCHG A,B:  00000 110 <00000011>
--  ADD A,B:     00000 110 <00000100>
--  SUB A,B:      00000 110 <00000101>
--  AND A,B:     00000 110 <00000110>
--  OR A,B:       00000 110 <00000111>
--  XOR A,B:     00000 110 <00001000>
--  DEC A:        00000 110 <00001001>
IF master_rst /= '0' THEN
   CPU_state_temp := to_unsigned(0, 8);
END IF;
PC_inc_func <= std_logic_vector(to_unsigned(0, 2));
IR_func <= std_logic_vector(to_unsigned(3, 2));
PC_func <= std_logic_vector(to_unsigned(3, 2));
IM_read <= '0';
addr_inc <= std_logic_vector(to_unsigned(0, 8));
Reg_OutA <= std_logic_vector(to_unsigned(0, 8));
Reg_OutB <= std_logic_vector(to_unsigned(0, 8));
RegA_func <= std_logic_vector(to_unsigned(4, 3));
RegB_func <= std_logic_vector(to_unsigned(4, 3));
ALU_func <= std_logic_vector(to_unsigned(9, 4));
-- NOP
-- main_code: <16..1>
-- major_opcode: <16..9>
-- minor_opcode: <12..9>
-- address_data: <8..1>
CASE CPU_state_temp IS
   WHEN "00000000" =>
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      --  RESETTING OUTPUTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      PC_inc_func <= std_logic_vector(to_unsigned(0, 2));
```

```
            PC_func <= std_logic_vector(to_unsigned(0, 2));
            IR_func <= std_logic_vector(to_unsigned(0, 2));
            RegA_func <= std_logic_vector(to_unsigned(0, 3));
            RegB_func <= std_logic_vector(to_unsigned(0, 3));
            CPU_state_temp := to_unsigned(1, 8);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            --  FETCH
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         WHEN "00000001" =>
            -- Read from IM (ROM)
            IM_read <= '1';
            -- PC increment PC+1
            PC_func <= std_logic_vector(to_unsigned(2, 2));
            -- store into IR
            IR_func <= std_logic_vector(to_unsigned(1, 2));
            CPU_state_temp := to_unsigned(2, 8);
         WHEN "00000010" =>
            -- Read from IR
            IR_func <= std_logic_vector(to_unsigned(2, 2));
            -- Accommodating for the 'unit delay' from IR_out to IR_in
            CPU_state_temp := to_unsigned(3, 8);
         WHEN "00000011" =>
            -- IR_in <16..1>
            main_opcode_temp := unsigned(IR_in);
            -- IR_in <16..9>
            cr := main_opcode_temp srl 8;
            IF cr(15 DOWNTO 4) /= "000000000000" THEN
               major_opcode_temp := "1111";
            ELSE
               major_opcode_temp := cr(3 DOWNTO 0);
            END IF;
  -- for instructions NOP,JMP,JMPZ,CALL,MOV A,xx MOV B,xx,RET
            -- IR_in <12..9>
            b_c_uint := major_opcode_temp AND to_unsigned(15, 4);
            minor_opcode_next <= b_c_uint;
            -- IR_in <8..1>
            c_uint := main_opcode_temp AND to_unsigned(255, 16);
```

```
            IF c_uint(15 DOWNTO 8) /= "00000000" THEN
               address_data_next <= "11111111";
            ELSE
               address_data_next <= c_uint(7 DOWNTO 0);
            END IF;
            -- Go to the decode stage
            CPU_state_temp := to_unsigned(4, 8);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            --  DECODE AND EXECUTE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         WHEN "00000100" =>
            CASE minor_opcode IS
               WHEN "0000" =>
                  -- NOP
                  CPU_state_temp := to_unsigned(1, 8);
               WHEN "0001" =>
                  -- JMP
                  addr_inc <= std_logic_vector(address_data);
                  PC_inc_func <= std_logic_vector(to_unsigned(1, 2));
                  PC_func <= std_logic_vector(to_unsigned(1, 2));
                  CPU_state_temp := to_unsigned(1, 8);
               WHEN "0010" =>
                  --JMPZ
                  reg_a_0 := unsigned(Reg_A);
                  IF reg_a_0 = 0 THEN
                     addr_inc <= std_logic_vector(address_data);
                     PC_inc_func <= std_logic_vector(to_unsigned(1, 2));
                     PC_func <= std_logic_vector(to_unsigned(1, 2));
                  END IF;
                  CPU_state_temp := to_unsigned(1, 8);
               WHEN "0011" =>
                  -- CALL
                  addr_inc <= std_logic_vector(address_data);
                  PC_inc_func <= std_logic_vector(to_unsigned(1, 2));
                  PC_func <= std_logic_vector(to_unsigned(1, 2));
                  CPU_state_temp := to_unsigned(1, 8);
               WHEN "0100" =>
```

```
   --MOV A,xx
   Reg_OutA <= std_logic_vector(address_data);
   RegA_func <= std_logic_vector(to_unsigned(1, 3));
   CPU_state_temp := to_unsigned(1, 8);
WHEN "0101" =>
   --MOV B,xx
   Reg_OutB <= std_logic_vector(address_data);
   RegB_func <= std_logic_vector(to_unsigned(1, 3));
   CPU_state_temp := to_unsigned(1, 8);
WHEN "0110" =>
   CASE address_data IS
      WHEN "00000000" =>
         --RET
        PC_inc_func <= std_logic_vector(to_unsigned(2, 2));
         PC_func <= std_logic_vector(to_unsigned(2, 2));
         CPU_state_temp := to_unsigned(5, 8);
      WHEN "00000001" =>
         --MOV A,B
         ALU_func <= std_logic_vector(to_unsigned(0, 4));
         RegA_func <= std_logic_vector(to_unsigned(2, 3));
         RegB_func <= std_logic_vector(to_unsigned(2, 3));
         CPU_state_temp := to_unsigned(5, 8);
      WHEN "00000010" =>
         --MOV B,A
         ALU_func <= std_logic_vector(to_unsigned(1, 4));
         RegA_func <= std_logic_vector(to_unsigned(2, 3));
         RegB_func <= std_logic_vector(to_unsigned(2, 3));
         CPU_state_temp := to_unsigned(5, 8);
      WHEN "00000011" =>
         --XCHG A,B
         ALU_func <= std_logic_vector(to_unsigned(2, 4));
         RegA_func <= std_logic_vector(to_unsigned(2, 3));
         RegB_func <= std_logic_vector(to_unsigned(2, 3));
         CPU_state_temp := to_unsigned(5, 8);
      WHEN "00000100" =>
         --ADD A,B
         ALU_func <= std_logic_vector(to_unsigned(3, 4));
```

```
            RegA_func <= std_logic_vector(to_unsigned(2, 3));
            RegB_func <= std_logic_vector(to_unsigned(2, 3));
            CPU_state_temp := to_unsigned(5, 8);
          WHEN "00000101" =>
            --SUB A,B
            ALU_func <= std_logic_vector(to_unsigned(4, 4));
            RegA_func <= std_logic_vector(to_unsigned(2, 3));
            RegB_func <= std_logic_vector(to_unsigned(2, 3));
            CPU_state_temp := to_unsigned(5, 8);
          WHEN "00000110" =>
            --AND A,B
            ALU_func <= std_logic_vector(to_unsigned(5, 4));
            RegA_func <= std_logic_vector(to_unsigned(2, 3));
            RegB_func <= std_logic_vector(to_unsigned(2, 3));
            CPU_state_temp := to_unsigned(5, 8);
          WHEN "00000111" =>
            --OR A,B
            ALU_func <= std_logic_vector(to_unsigned(6, 4));
            RegA_func <= std_logic_vector(to_unsigned(2, 3));
            RegB_func <= std_logic_vector(to_unsigned(2, 3));
            CPU_state_temp := to_unsigned(5, 8);
          WHEN "00001000" =>
            --XOR A,B
            ALU_func <= std_logic_vector(to_unsigned(7, 4));
            RegA_func <= std_logic_vector(to_unsigned(2, 3));
            RegB_func <= std_logic_vector(to_unsigned(2, 3));
            CPU_state_temp := to_unsigned(5, 8);
          WHEN "00001001" =>
            --DEC A
            ALU_func <= std_logic_vector(to_unsigned(8, 4));
            RegA_func <= std_logic_vector(to_unsigned(2, 3));
            CPU_state_temp := to_unsigned(5, 8);
          WHEN OTHERS =>
            NULL;
        END CASE;
      WHEN OTHERS =>
        NULL;
```

```
            END CASE;
          WHEN "00000101" =>
            RegA_func <= std_logic_vector(to_unsigned(2, 3));
            RegB_func <= std_logic_vector(to_unsigned(2, 3));
            CPU_state_temp := to_unsigned(1, 8);
          WHEN OTHERS =>
            NULL;
        END CASE;
        CPU_state_next <= CPU_state_temp;
        major_opcode_next <= major_opcode_temp;
        main_opcode_next <= main_opcode_temp;
    END PROCESS CPU_Controller;
END fsm_SFHDL;
```

Пример 1. Код языка VHDL управляющего автомата проектируемого процессора, сгенерированный в автоматическом режиме с помощью Simulink HDL Coder системы Matlab/Simulink

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY PC_Incrementer IS
    PORT (
        clk : IN std_logic;
        clk_enable : IN std_logic;
        reset : IN std_logic;
        func : IN std_logic_vector(1 DOWNTO 0);
        addr : IN std_logic_vector(7 DOWNTO 0);
        PC_curr : IN std_logic_vector(7 DOWNTO 0);
        PC_next : OUT std_logic_vector(7 DOWNTO 0);
        Temp : OUT std_logic_vector(7 DOWNTO 0));
END PC_Incrementer;

ARCHITECTURE fsm_SFHDL OF PC_Incrementer IS

    SIGNAL PC_Temp : unsigned(7 DOWNTO 0);
    SIGNAL PC_Temp_next : unsigned(7 DOWNTO 0);
```

```
BEGIN
   initialize_PC_Incrementer : PROCESS (reset, clk)
      -- local variables
   BEGIN
      IF reset = '1' THEN
         PC_Temp <= to_unsigned(0, 8);
      ELSIF clk'EVENT AND clk= '1' THEN
         IF clk_enable= '1' THEN
            PC_Temp <= PC_Temp_next;
         END IF;
      END IF;
   END PROCESS initialize_PC_Incrementer;

   PC_Incrementer : PROCESS (PC_Temp, func, addr, PC_curr)
      -- local variables
      VARIABLE PC_Temp_temp : unsigned(7 DOWNTO 0);
   BEGIN
      PC_Temp_temp := PC_Temp;
      -- func = 0 => reset PC_Inc
      -- func = 1 => store into PC_Inc when JMP, JMPZ, CALL
      -- func = 2 => load from PC_Inc when RET
      PC_next <= PC_curr;
      Temp <= std_logic_vector(to_unsigned(0, 8));
      CASE func IS
         WHEN "00" =>
            -- reset PC_Inc
            PC_next <= std_logic_vector(to_unsigned(0, 8));
         WHEN "01" =>
            -- store into PC_Inc when JMP, JMPZ, CALL
            PC_next <= addr;
            PC_Temp_temp := unsigned(PC_curr);
            Temp <= std_logic_vector(PC_Temp_temp);
         WHEN "10" =>
            -- load from PC_Inc when RET
            PC_next <= std_logic_vector(PC_Temp);
         WHEN OTHERS =>
            NULL;
```

```
      END CASE;
      PC_Temp_next <= PC_Temp_temp;
   END PROCESS PC_Incrementer;
END fsm_SFHDL;
```

Пример 2. Регистр специального назначения процессора на языке VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY Program_Counter IS
   PORT (
      clk : IN std_logic;
      clk_enable : IN std_logic;
      reset : IN std_logic;
      func : IN std_logic_vector(1 DOWNTO 0);
      addr_in : IN std_logic_vector(7 DOWNTO 0);
      addr_out : OUT std_logic_vector(7 DOWNTO 0));
END Program_Counter;
ARCHITECTURE fsm_SFHDL OF Program_Counter IS
   SIGNAL PC_value : unsigned(7 DOWNTO 0);
   SIGNAL PC_value_next : unsigned(7 DOWNTO 0);
BEGIN
   initialize_Program_Counter : PROCESS (reset, clk)
      -- local variables
   BEGIN
      IF reset = '1' THEN
         PC_value <= to_unsigned(0, 8);
      ELSIF clk'EVENT AND clk= '1' THEN
         IF clk_enable= '1' THEN
            PC_value <= PC_value_next;
         END IF;
      END IF;
   END PROCESS initialize_Program_Counter;
   Program_Counter : PROCESS (PC_value, func, addr_in)
      -- local variables
      VARIABLE ain : unsigned(15 DOWNTO 0);
```

```
      VARIABLE ain_0 : unsigned(15 DOWNTO 0);
   BEGIN
      PC_value_next <= PC_value;
      -- Program Counter
      -- func = 0 => reset PC
      -- func = 1 => load PC
      -- func = 2 => increment PC
      addr_out <= std_logic_vector(PC_value);
      CASE func IS
         WHEN "00" =>
            -- reset
            PC_value_next <= to_unsigned(0, 8);
         WHEN "01" =>
            -- store into PC
            PC_value_next <= unsigned(addr_in);
         WHEN "10" =>
            -- increment PC
  ain := resize(PC_value & '0' & '0' & '0' & '0' & '0' & '0' & '0', 16);
            ain_0 := ain + 128;
IF (ain_0(15) /= '0') OR (ain_0(14 DOWNTO 7) = "11111111") THEN
               PC_value_next <= "11111111";
            ELSE
PC_value_next <= ain_0(14 DOWNTO 7) + ("0" & (ain_0(6)));
            END IF;
         WHEN OTHERS =>
            NULL;
      END CASE;
   END PROCESS Program_Counter;
END fsm_SFHDL;
```

Пример 3. Счетчик команд микропроцессора на языке VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY Instruction_ROM IS
   PORT (
      clk : IN std_logic;
      clk_enable : IN std_logic;
```

```
      reset : IN std_logic;
      addr : IN std_logic_vector(7 DOWNTO 0);
      read : IN std_logic;
      instr_out : OUT std_logic_vector(15 DOWNTO 0));
END Instruction_ROM;
ARCHITECTURE fsm_SFHDL OF Instruction_ROM IS
  -- TMW_TO_SIGNED
  FUNCTION tmw_to_signed(arg: unsigned; width: integer) RETURN signed
IS
  BEGIN
    IF arg(arg'right) = 'U' OR  arg(arg'right) = 'X' THEN
      RETURN to_signed(1, width);
    END IF;
    RETURN to_signed(to_integer(arg), width);
  END FUNCTION;
  TYPE T_UFIX_16_256 IS ARRAY (255 DOWNTO 0) of unsigned(15
DOWNTO 0);
  SIGNAL data : T_UFIX_16_256;
  SIGNAL data_next : T_UFIX_16_256;
BEGIN
  initialize_Instruction_ROM : PROCESS (reset, clk)
    -- local variables
    VARIABLE b_0 : INTEGER;
  BEGIN
    IF reset = '1' THEN
      FOR b IN 0 TO 255 LOOP
        data(b) <= to_unsigned(0, 16);
      END LOOP;
    ELSIF clk'EVENT AND clk= '1' THEN
      IF clk_enable= '1' THEN
        FOR b_0 IN 0 TO 255 LOOP
          data(b_0) <= data_next(b_0);
        END LOOP;
      END IF;
    END IF;
  END PROCESS initialize_Instruction_ROM;
  Instruction_ROM : PROCESS (data, addr, read)
    -- local variables
    VARIABLE data_temp : T_UFIX_16_256;
  BEGIN
    FOR b IN 0 TO 255 LOOP
```

```
      data_temp(b) := data(b);
    END LOOP;
    data_temp(0) := to_unsigned(1036, 16);
    data_temp(1) := to_unsigned(1303, 16);
    data_temp(2) := to_unsigned(1540, 16);
    data_temp(3) := to_unsigned(1545, 16);
    data_temp(4) := to_unsigned(1358, 16);
    data_temp(5) := to_unsigned(1539, 16);
    data_temp(6) := to_unsigned(1541, 16);
    data_temp(7) := to_unsigned(1545, 16);
    data_temp(8) := to_unsigned(1542, 16);
    data_temp(9) := to_unsigned(523, 16);
    data_temp(10) := to_unsigned(263, 16);
    data_temp(11) := to_unsigned(1037, 16);
    data_temp(12) := to_unsigned(1397, 16);
    data_temp(13) := to_unsigned(1543, 16);
    data_temp(14) := to_unsigned(1539, 16);
    data_temp(15) := to_unsigned(1544, 16);
    data_temp(16) := to_unsigned(277, 16);
    data_temp(17) := to_unsigned(1135, 16);
    data_temp(18) := to_unsigned(1480, 16);
    data_temp(19) := to_unsigned(1542, 16);
    data_temp(20) := to_unsigned(1536, 16);
    data_temp(21) := to_unsigned(785, 16);
    data_temp(22) := to_unsigned(0, 16);
    IF read = '1' THEN
instr_out <= std_logic_vector(data_temp(to_integer(tmw_to_signed(unsigned(addr) + 1,
32) - 1)));
    ELSE
      instr_out <= std_logic_vector(to_unsigned(0, 16));
    END IF;
    FOR c IN 0 TO 255 LOOP
      data_next(c) <= data_temp(c);
    END LOOP;
  END PROCESS Instruction_ROM;
END fsm_SFHDL;
```

Пример 4. Память программ процессора на языке VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

```
ENTITY Instruction_Register IS
   PORT (
      clk : IN std_logic;
      clk_enable : IN std_logic;
      reset : IN std_logic;
      func : IN std_logic_vector(1 DOWNTO 0);
      IR_in : IN std_logic_vector(15 DOWNTO 0);
      IR_out : OUT std_logic_vector(15 DOWNTO 0));
END Instruction_Register;

ARCHITECTURE fsm_SFHDL OF Instruction_Register IS
   SIGNAL IR_value : unsigned(15 DOWNTO 0);
   SIGNAL IR_value_next : unsigned(15 DOWNTO 0);
BEGIN
   initialize_Instruction_Register : PROCESS (reset, clk)
      -- local variables
   BEGIN
      IF reset = '1' THEN
         IR_value <= to_unsigned(0, 16);
      ELSIF clk'EVENT AND clk= '1' THEN
         IF clk_enable= '1' THEN
            IR_value <= IR_value_next;
         END IF;
      END IF;
   END PROCESS initialize_Instruction_Register;
   Instruction_Register : PROCESS (IR_value, func, IR_in)
      -- local variables
   BEGIN
      IR_value_next <= IR_value;
      -- A 16-bit Instruction Register with the following func:
      -- func == 0 => reset
      -- func == 1 => store into IR
      -- func == 2 => read from IR;
      -- otherwise, preserve old value and return 0
      IR_out <= std_logic_vector(to_unsigned(0, 16));
      CASE func IS
         WHEN "00" =>
```

```
            -- reset
            IR_value_next <= to_unsigned(0, 16);
          WHEN "01" =>
            -- store into IR
            IR_value_next <= unsigned(IR_in);
          WHEN "10" =>
            -- read IR
            IR_out <= std_logic_vector(IR_value);
          WHEN OTHERS =>
            NULL;
        END CASE;
    END PROCESS Instruction_Register;
END fsm_SFHDL;
```

Пример 5. Блок регистра инструкций микропроцессора на языке VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY RegisterA IS
    PORT (
        clk : IN std_logic;
        clk_enable : IN std_logic;
        reset : IN std_logic;
        func : IN std_logic_vector(2 DOWNTO 0);
        Reg_in_A_1 : IN std_logic_vector(7 DOWNTO 0);
        Reg_in_A_2 : IN std_logic_vector(7 DOWNTO 0);
        Reg_out_A : OUT std_logic_vector(7 DOWNTO 0));
END RegisterA;

ARCHITECTURE fsm_SFHDL OF RegisterA IS

    SIGNAL Reg_value : unsigned(7 DOWNTO 0);
    SIGNAL Reg_value_next : unsigned(7 DOWNTO 0);

BEGIN
```

```
initialize_RegisterA : PROCESS (reset, clk)
   -- local variables
BEGIN
   IF reset = '1' THEN
      Reg_value <= to_unsigned(0, 8);
   ELSIF clk'EVENT AND clk= '1' THEN
      IF clk_enable= '1' THEN
         Reg_value <= Reg_value_next;
      END IF;
   END IF;
END PROCESS initialize_RegisterA;

RegisterA : PROCESS (Reg_value, func, Reg_in_A_1, Reg_in_A_2)
   -- local variables
BEGIN
   Reg_value_next <= Reg_value;
   -- func == 0 => reset;
   -- func == 1 => store into RegisterA from port 1;
   -- func == 2 => store into RegisterA from port 2;
   -- func == 3 => read from RegisterA;
   -- HDL specific fimath
   Reg_out_A <= std_logic_vector(Reg_value);
   CASE func IS
      WHEN "000" =>
         -- reset
         Reg_value_next <= to_unsigned(0, 8);
      WHEN "001" =>
         -- store into Reg_A from port 1
         Reg_value_next <= unsigned(Reg_in_A_1);
      WHEN "010" =>
         -- store into Reg_A from port 2
         Reg_value_next <= unsigned(Reg_in_A_2);
      WHEN "011" =>
         -- read Reg_A
         Reg_out_A <= std_logic_vector(Reg_value);
      WHEN OTHERS =>
         NULL;
```

```
      END CASE;
   END PROCESS RegisterA;
END fsm_SFHDL;
```

Пример 6. Блок регистра общего назначения A микропроцессора на языке VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY ALU IS
   PORT (
      clk : IN std_logic;
      clk_enable : IN std_logic;
      reset : IN std_logic;
      func : IN std_logic_vector(3 DOWNTO 0);
      inA : IN std_logic_vector(7 DOWNTO 0);
      inB : IN std_logic_vector(7 DOWNTO 0);
      outA : OUT std_logic_vector(7 DOWNTO 0);
      outB : OUT std_logic_vector(7 DOWNTO 0));
END ALU;
ARCHITECTURE fsm_SFHDL OF ALU IS
   -- TMW_TO_SIGNED
   FUNCTION tmw_to_signed(arg: unsigned; width: integer) RETURN signed
IS
   BEGIN
      IF arg(arg'right) = 'U' OR  arg(arg'right) = 'X' THEN
         RETURN to_signed(1, width);
      END IF;
      RETURN to_signed(to_integer(arg), width);
   END FUNCTION;
   -- TMW_TO_UNSIGNED
   FUNCTION tmw_to_unsigned(arg: signed; width: integer) RETURN unsigned
IS
      CONSTANT ARG_LEFT: INTEGER := ARG'LENGTH-1;
      ALIAS XARG: SIGNED(ARG_LEFT downto 0) is ARG;
      VARIABLE result : unsigned(width-1 DOWNTO 0);
      VARIABLE argSize : integer;
   BEGIN
      IF XARG(XARG'high-1) = 'U' OR  arg(arg'right) = 'X' THEN
         RETURN to_unsigned(1, width);
```

```
      END IF;
      IF (ARG_LEFT < width-1) THEN
         result := (OTHERS => XARG(ARG_LEFT));
         result(ARG_LEFT downto 0) := unsigned(XARG);
      ELSE
         result(width-1 downto 0) := unsigned(XARG(width-1 downto 0));
      END IF;
      RETURN result;
   END FUNCTION;
BEGIN
   ALU : PROCESS (func, inA, inB)
      -- local variables
      VARIABLE X_temp : unsigned(7 DOWNTO 0);
      VARIABLE ina_0 : unsigned(7 DOWNTO 0);
      VARIABLE ina_1 : signed(8 DOWNTO 0);
      VARIABLE ina_2 : signed(8 DOWNTO 0);
   BEGIN
      -- This 8-bit ALU supports the following operations:
      -- MOV, XCHG, ADD, SUB, AND, OR, XOR, DEC
      -- func = 0 => MOV A,B
      -- func = 1 => MOV B,A
      -- func = 2 => XCHG A,B
      -- func = 3 => ADD A,B
      -- func = 4 => SUB A,B
      -- func = 5 => AND A,B
      -- func = 6 => OR A,B
      -- func = 7 => XOR A,B
      -- func = 8 => DEC A
      -- Simply pass the inA, when there is no designated func
      outA <= inA;
      -- Simply pass the inB, when there is no designated func
      outB <= inB;
      CASE func IS
         WHEN "0000" =>
            --MOV A,B
            outA <= inB;
         WHEN "0001" =>
            --MOV B,A
            outB <= inA;
         WHEN "0010" =>
            --XCHG A,B
```

```
            X_temp := unsigned(inB);
            outB <= inA;
            outA <= std_logic_vector(X_temp);
         WHEN "0011" =>
            --ADD A,B
            ina_0 := unsigned(inA) + unsigned(inB);
            outA <= std_logic_vector(ina_0);
         WHEN "0100" =>
            --SUB A,B
ina_1 := tmw_to_signed(unsigned(inA), 9) - tmw_to_signed(unsigned(inB), 9);
            IF ina_1(8) = '1' THEN
               outA <= "00000000";
            ELSE
outA <= std_logic_vector(resize(unsigned(ina_1(7 DOWNTO 0)), 8));
            END IF;
         WHEN "0101" =>
            --AND A,B
outA <= std_logic_vector(tmw_to_unsigned(tmw_to_signed(unsigned(inA), 32)
AND tmw_to_signed(unsigned(inB), 32), 8));
         WHEN "0110" =>
            --OR A,B
outA <= std_logic_vector(tmw_to_unsigned(tmw_to_signed(unsigned(inA), 32)
OR tmw_to_signed(unsigned(inB), 32), 8));
         WHEN "0111" =>
            --XOR A,B
outA <= std_logic_vector(tmw_to_unsigned(tmw_to_signed(unsigned(inA), 32)
XOR tmw_to_signed(unsigned(inB), 32), 8));
         WHEN "1000" =>
            --DEC A
            ina_2 := tmw_to_signed(unsigned(inA), 9) - 1;
            IF ina_2(8) = '1' THEN
               outA <= "00000000";
            ELSE
           outA <= std_logic_vector(resize(unsigned(ina_2(7 DOWNTO 0)), 8));
            END IF;
         WHEN OTHERS =>
            NULL;
      END CASE;
   END PROCESS ALU;
END fsm_SFHDL;
```

Пример 7. Блок АЛУ микропроцессора на языке VHDL

Процессор, позволяющий проводить вычисления в формате с фиксированной зяпятой, код языка которого был получен с использованием Simulink HDL Coder системы визуального иммитационного моделирования Matlab/Simulink, показал свою работоспособность в САПР Quartus II компании Altera. Процессор может быть успешно размещен в ПЛИС Stratix III EP3SL50F484C2, и занимает менее 1 % ресурсов адаптивных таблиц перекодировок (ALUT, 209) для реализации комбинационной логики и менее 1 % ресурсов последовательностной логики (регистров, 105).

Автоматически сгенерированный код языка VHDL с использованием Simulink HDL Coder системы Matlab/Simulink, позволяет значительно ускорить процесс разработки пользовательских микропроцессорных ядер, для реализации их в базисе ПЛИС.

К недостаткам следует отнести наличие достаточно большого числа явных преобразований тесно связанных между собой типов, что определяется форматом представления исходных данных системы Matlab/Simulink.

4.6. Проектирование микропроцессорных ядер с конвейерной архитектурой для реализации в базисе ПЛИС

Воспользуемся системой команд синхронного процессора с циклом работы в два такта и спроектируем микропроцессорное ядро с использованием конвейерной архитектуры. Типовой конвейер микропроцессора содержит пять стадии: выборка инструкции; декодирование инструкции; адресация и выборка операнда из ОЗУ; выполнение арифметических операций; сохранение результата операции. Каждый этап команды рассматривается как каскад конвейера. Таким образом, можно организовать наложение команд, при

котором новая команда будет начинать выполняться в первый момент каждого такта. Благодаря использованию внутреннего параллелизма потока команд конвейерная обработка позволяет существенно снизить в среднем время выполнения одной команды. Пропускная способность машины с конвейерной обработкой определяется числом команд, пропущенных через конвейер за единицу времени.

Для реализации процессора необходима память, в которой будут храниться команды микропроцессора (память программ) и инструкции для управляющего автомата. Проектируемая память имеет асинхронный сигнал сброса reset, состоит из двух массивов памяти емкостью 4096 бит. Ниже приведен код языка VHDL асинхронной памяти (пример 1). ПЗУ разделено на 2 области и обладает двумя адресными шинами addr_cmd[15..0] и addr_avt[15..0]. По шине avt_out[15..0] передаются инструкции управляющего автомата, а по шине cmd_out[15..0] передаются команды микропроцессора.

Разработаем для микропроцессорного ядра управляющий автомат, на девять состояний (рис.4.35). Использование управляющего автомата удовлетворяет современной концепции синхронного кодирования при реализации цифровых устройств в базисе ПЛИС.

Управляющий автомат имеет вход синхронизации clk и асинхронного сброса rst, который устанавливает автомат в начальное состояние INST. В начальном состоянии INST по шине instr[15..0] происходит загрузка из ПЗУ инструкции для управляющего автомата в регистр инструкций, в котором выделяются разряды [15..12] (шина instr[15..12]) для декодирования движений по веткам автомата и разряды [11..9] (шина instr[11..9]) для декодирования логико-арифметических операций.

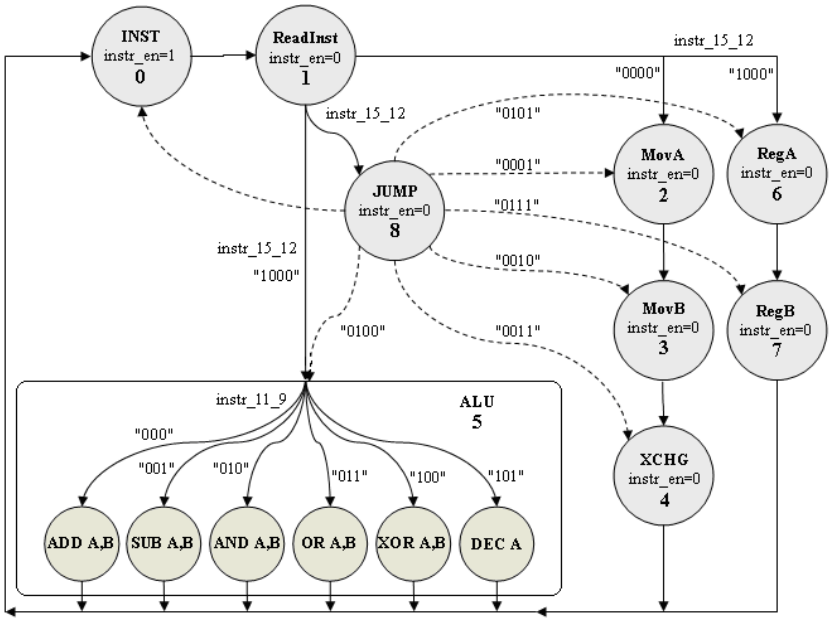


Рис.4.35. Блок-схема управляющего автомата микропроцессорного ядра на 9 состояний

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY rom_syn IS
PORT (addr_cmd : IN std_logic_vector(15 DOWNTO 0);
        addr_avt : IN std_logic_vector(15 DOWNTO 0);
        cmd_out : OUT std_logic_vector(15 DOWNTO 0);
        avt_out : OUT std_logic_vector(15 DOWNTO 0));
END rom_syn;
ARCHITECTURE a OF rom_syn IS
TYPE T_UFIX_16_256 IS ARRAY (255 DOWNTO 0) of unsigned(15
DOWNTO 0);
BEGIN
        PROCESS (addr_cmd)
        VARIABLE b : INTEGER;
        VARIABLE c : INTEGER;
```

```
        VARIABLE data_temp : T_UFIX_16_256;
BEGIN
        FOR b IN 128 TO 255 LOOP
        data_temp(b) := to_unsigned(0, 16);
        END LOOP;
        data_temp(128) := to_unsigned(0, 16);
        data_temp(129) := to_unsigned(1165, 16);
        data_temp(130) := to_unsigned(1358, 16);
        data_temp(131) := to_unsigned(1540, 16);
        data_temp(132) := to_unsigned(1541, 16);
        data_temp(133) := to_unsigned(1542, 16);
        data_temp(134) := to_unsigned(1543, 16);
        data_temp(135) := to_unsigned(1544, 16);
        data_temp(136) := to_unsigned(1545, 16);
        data_temp(137) := to_unsigned(1539, 16);
        data_temp(138) := to_unsigned(1546, 16);
        data_temp(139) := to_unsigned(1547, 16);
        cmd_out <=
std_logic_vector(data_temp(to_integer(unsigned(addr_cmd))));
END PROCESS;
PROCESS (addr_avt)
        VARIABLE data_temp : T_UFIX_16_256;
BEGIN
        FOR c IN 0 TO 255 LOOP
        data_temp(c) := to_unsigned(0, 16);
END LOOP;
        data_temp(0) := "0000000000000000";
        data_temp(1) := "1000000000000000";
        data_temp(2) := "1000001000000000";
        data_temp(3) := "1000010000000000";
        data_temp(4) := "1000011000000000";
        data_temp(5) := "1000100000000000";
        data_temp(6) := "1000101000000000";
        data_temp(7) := "0001000000000000";
        data_temp(8) := "0010000000000000";
        data_temp(9) := "0011000000000000";
        data_temp(10) := "0100000000000000";
```

```
        data_temp(11) := "1001000000000000";
        avt_out <=
std_logic_vector(data_temp(to_integer(unsigned(addr_avt))));
END PROCESS;
END a;
```

Пример 1. Код языка VHDL блока асинхронной памяти

На выходе автомата ip[15..0] с помощью битов контроля (внутренний сигнал control_signal) формируется адрес команды, хранящийся в ПЗУ. Высокий уровень сигнала instr_en разрешает получение новой инструкции из ПЗУ для управляющего автомата, и увеличивает содержимое счетчика на единицу. Сигнал num_state[3..0] показывает номер состояния, в котором находится управляющий автомат.

Следующее состояние, в которое переходит автомат по переднему фронту синхроимпульса clk – ReadInst. В состоянии ReadInst происходит чтение полученной инструкции и выбор следующего состояния автомата. В состояниях INST и ReadInst в АЛУ недолжно выполнять логико-арифметических операций. Автомат реализует команду NOP.

С приходом фронта синхроимпульса автомат перейдет в одну из возможных веток (всего возможны 4 ветки), например, в состояние MovA если на шине instr_15_12 присутствует код “0000” (ветка с состояниями MovA, MovB, XCHG для выполнения АЛУ трех регистровых операций пересылки), в состояние ALU если “1000” (в этом состоянии АЛУ выполняет шесть логико-арифметических операций), в состояние RegA если “1001” (ветка с состояниями RegA, RegB, АЛУ выполняет загрузку РОН А и В с входного порта) или в состояние JUMP, если на шине instr_15_12 присутствует любое другое значение. В состоянии JUMP, в зависимости от кода на шине instr_15_12 автомат может “перепрыгнуть” в другое состояние (возможные переходы показаны пунктирными линиями), при этом АЛУ не выполняет операций, а автомат реализует команду NOP.

В состоянии MovA происходит непосредственная загрузка в регистр A операнда, заданного младшим байтом команды. Следующим состоянием, в котором произойдет загрузка операнда в регистр B будет MovB. В состоянии XCHG произойдет обмен содержимого в регистрах A и B. После этого автомат возвращается в состояние INST и читает следующую инструкцию instr из памяти. В состоянии ALU код на шине instr_11_9 выбирает логико-арифметическую операцию, которая будет выполнена в АЛУ. В состояниях RegA и RegB происходит загрузка данных в регистры A и B с входного порта. VHDL описание проектируемого автомата с использованием двухпроцессорного шаблона показано ниже (пример 2):

```
ARCHITECTURE behave OF Control IS
-- Definition of the state names
TYPE state_type IS (Inst, ReadInst, MovA, MovB, XCHG, ALU, JUMP,
RegA, RegB);
SIGNAL state, next_state : state_type;
Signal control_signal: std_logic_vector(15 downto 0);
BEGIN
-- State process
PROCESS(clk, rst)
BEGIN
IF rst = '1' THEN
state <= Inst;
ELSIF clk'event and clk='1' THEN
state <= next_state;
END IF;
END PROCESS;
-- Logic Process
PROCESS(state)
BEGIN
CASE state IS
--Instruction
WHEN Inst =>
```

```
control_signal <= "0000000010000000"; -- 128(D) NOP
num_state <= "0000";
instr_en <= '1';
next_state <= ReadInst;
--Read Instruction
WHEN ReadInst =>
control_signal <= "0000000010000000"; -- 128(D) NOP
num_state <= "0001";
instr_en <= '0';
IF instr_15_12 = "0000" THEN next_state <=MovA;
ELSIF instr_15_12 = "1000" THEN next_state <=ALU;
ELSIF instr_15_12 = "1001" THEN next_state<=RegA;
ELSE next_state <= JUMP;
END IF;
--MovA
WHEN MovA =>
control_signal <= "0000000010000001"; -- 129(D) MOV A,xx
next_state <= MovB;
num_state <= "0010";
instr_en <= '0';
--MovB
WHEN MovB =>
control_signal <= "0000000010000010"; -- 130(D) MOV B,xx
next_state <= XCHG;
num_state <= "0011";
instr_en <= '0';
-- XCHG
WHEN XCHG =>
next_state <= Inst;
num_state <= "0100";
instr_en <= '0';
control_signal <= "0000000010001001"; --137(D) XCHG A,B
-- ALU
WHEN ALU =>
instr_en <= '0';
num_state <= "0101";
IF instr_11_9 = "000" THEN
```

```
control_signal <= "0000000010000011"; -- 131(D) ADD A,B
next_state <= INST;
ELSIF instr_11_9 = "001" THEN
control_signal <= "0000000010000100"; -- 132(D) SUB A,B
next_state <= INST;
ELSIF instr_11_9 = "010" THEN
control_signal <= "0000000010000101"; -- 133(D) AND A,B
next_state <= INST;
ELSIF instr_11_9 = "011" THEN
control_signal <= "0000000010000110"; -- 134(D) OR A,B
next_state <= INST;
ELSIF instr_11_9= "100" THEN
control_signal <= "0000000010000111"; -- 135(D) XOR A,B
next_state <= INST;
ELSIF instr_11_9 = "101" THEN
control_signal <= "0000000010001000"; -- 136(D) DEC A
next_state <= INST;
END IF;
--RegA
WHEN RegA =>
next_state <= RegB;
num_state <= "0110";
instr_en <= '0';
control_signal <= "0000000010001010"; --138(D) MOV A,indata
--RegB
WHEN RegB =>
next_state <= INST;
num_state <= "0111";
instr_en <= '0';
control_signal <= "0000000010001011"; --139(D) MOV B,indata
--JUMP
WHEN JUMP =>
control_signal <= "0000000010000000"; --128(D) NOP
instr_en <= '0';
num_state <= "1000";
IF instr_15_12 = "0001" THEN next_state <= MovA;
ELSIF instr_15_12 = "0010" THEN next_state <= MovB;
```

```
ELSIF instr_15_12 = "0011" THEN next_state <= ALU;
ELSIF instr_15_12 = "0100" THEN next_state <= XCHG;
ELSIF instr_15_12 = "0110" THEN next_state <= RegA;
ELSIF instr_15_12 = "0111" THEN next_state <= RegB;
ELSE next_state <= Inst;
END IF;
END case;
END process;
ip <= control_signal;
END behave;
```

Пример 2. Фрагмент кода языка VHDL управляющего автомата

Синхронное АЛУ выполняет различные логико-арифметические операции над операндами, значения которых сохраняются в регистрах-защелках A и B. В этом блоке реализованы следующие команды (команды JMPZ, CALL, RET не поддерживаются, добавлены две новые команды с кодом 1546(D) (MOV A,indata) и 1547(D) (MOV B,indata) для загрузки РОН A и B с входного порта): Mov A,xx; Mov B,xx; XCHG A,B; ADD A,B; SUB A,B; AND A,B; OR A,B; XOR A,B; DEC A; Reg A; Reg B (пример 3).

```
signal regA,regB,indata: std_logic_vector(7 downto 0);
BEGIN
PROCESS (clk,res)
BEGIN
regA<=a;
regB<=b;
indata<=input;
if (res = '1') then
                    regA <="00000000";
                    regB <="00000000";
elsif (clk'event and clk='1') then
case conv_integer(cmd) is
when 1024 to 1279 => regA<=cmd(7 downto 0); enaa<='1'; enab<='0';
```

```
when 1280 to 1535 => regB<=cmd(7 downto 0); enab<='1'; enaa<='0';
when 1537=>regA<=regB; enaa<='1'; enab<='0';
when 1538=>regB<=regA; enaa<='0'; enab<='1';
when 1539=>regA<=regB; regB<=regA;
enaa<='1'; enab<='1';
when 1540=>regA<=regA+regB; enaa<='1'; enab<='0';
when 1541=>regA<=regA-regB; enaa<='1'; enab<='0';
when 1542=>regA<=regA and regB;
enaa<='1'; enab<='0';
when 1543=>regA<=regA or regB;
enaa<='1'; enab<='0';
when 1544=>regA<=regA xor regB;
enaa<='1'; enab<='0';
when 1545=>regA<=regA-1; enaa<='1'; enab<='0';
when 1546=>regA<=indata; enaa<='1'; enab<='0';
when 1547=>regB<=indata; enaa<='0'; enab<='1';
when others=> dataa<=regA; datab<=regB;
enaa<='0'; enab<='0';
end case;
end if;
dataa<=regA; datab<=regB;
end process;
```

Пример 3. Фрагмент кода языка VHDL блока АЛУ

На рис.4.36 показана схема микропроцессорного ядра с конвейерной архитектурой. В первом состоянии управляющего автомата происходит чтение инструкции из ПЗУ (instr[15..0]) и выделение из нее полей – instr_15_12[3..0] и instr_11_9[2..0]. Адрес этой инструкции для автомата формирует счетчик (шина pc[15..0]), прибавляющий 1 к предыдущему адресу, когда автомат выполнит цикл команд и вернется в состояние INST. Автомат для каждого своего состояния вырабатывает адрес нужной команды хранящейся в ПЗУ программ (шина ip[15..0]) с помощью битов контроля (сигнал control_signal). Эта команда по шине команд cmd_out[15..0] передается в АЛУ, где выполняется требуемая

операция, результаты помещаются в регистры. Схема имеет 2 регистра (восьмиразрядный регистр – защелку) общего назначения A и B, данные из которых попадают в АЛУ для выполнения следующей операции.

На рис.4.37 приведены временные диаграммы работы микропроцессорного ядра. В начальном состоянии управляющего автомата (INST) происходит запись инструкции в блок выделения полей (блок instreg) и из ПЗУ программ извлекается команда NOP с кодом 0, при которой нет операций. С приходом переднего фронта синхросигнала clk автомат переключается в состояние ReadInst, в котором читается полученная инструкция и выбирается следующее состояние.

Во втором состоянии выполняется команда Mov A,xx, которая загружает в регистр A значение, заданное младшим байтом команды. Из ПЗУ программ была получена команда 48D(H) (1165(D)) и в регистр A было загружено число 8D(H) или 141 в десятичной системе. Согласно схеме на рис.4.35 следующее состояние, которое принимает автомат – состояние номер 3 (MovB). В этом состоянии выполняется команда Mov B,xx. Команда загружает в регистр B значение, заданное младшим байтом команды. Тестирование команды пересылки Mov B,xx показано на рис.4.37. Из ПЗУ была получена команда 54E(H) и в регистр B было загружено число 4E(H) или 78 в десятичной системе. Согласно схеме на рис.4.35 следующее состояние автомата - XCHG. Из ПЗУ была извлечена команда 603(H) и регистры A и B обменялись значениями. При этом на шинах instr_15_12 = “0000” и instr_11_9 = “000”.

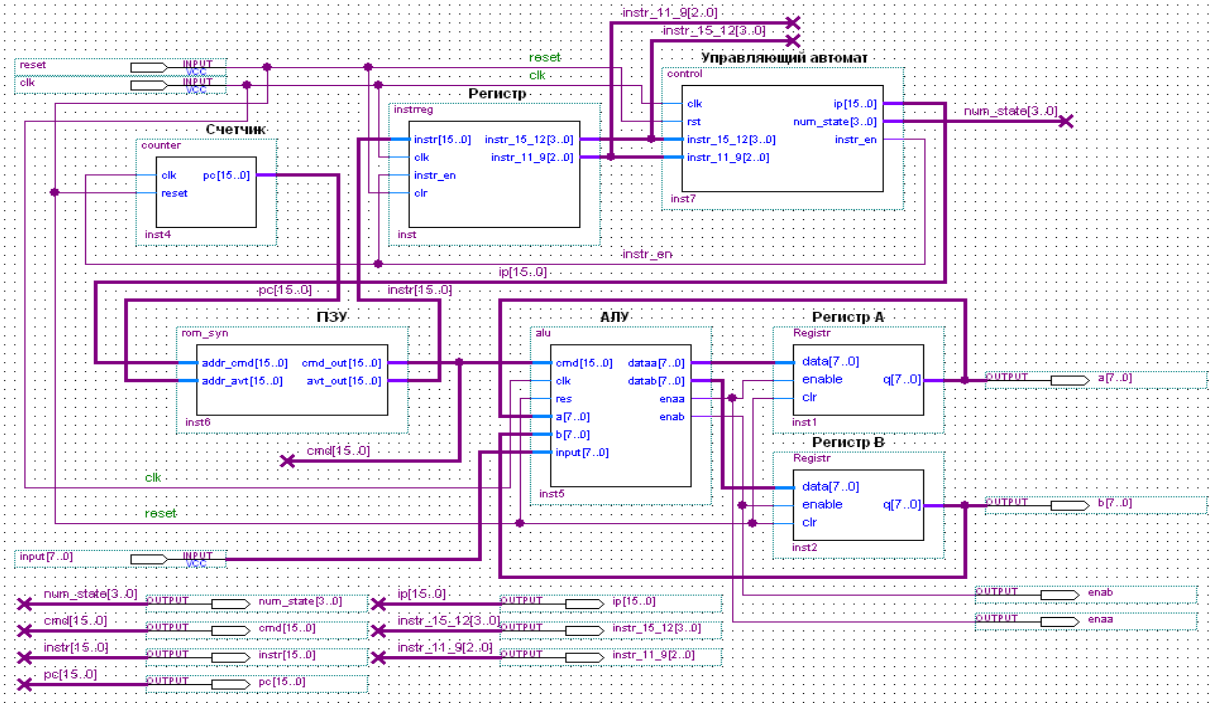


Рис.4.36. Схема микропроцессорного ядра с конвейерной архитектурой в графическом редакторе САПР ПЛИС Quartus II

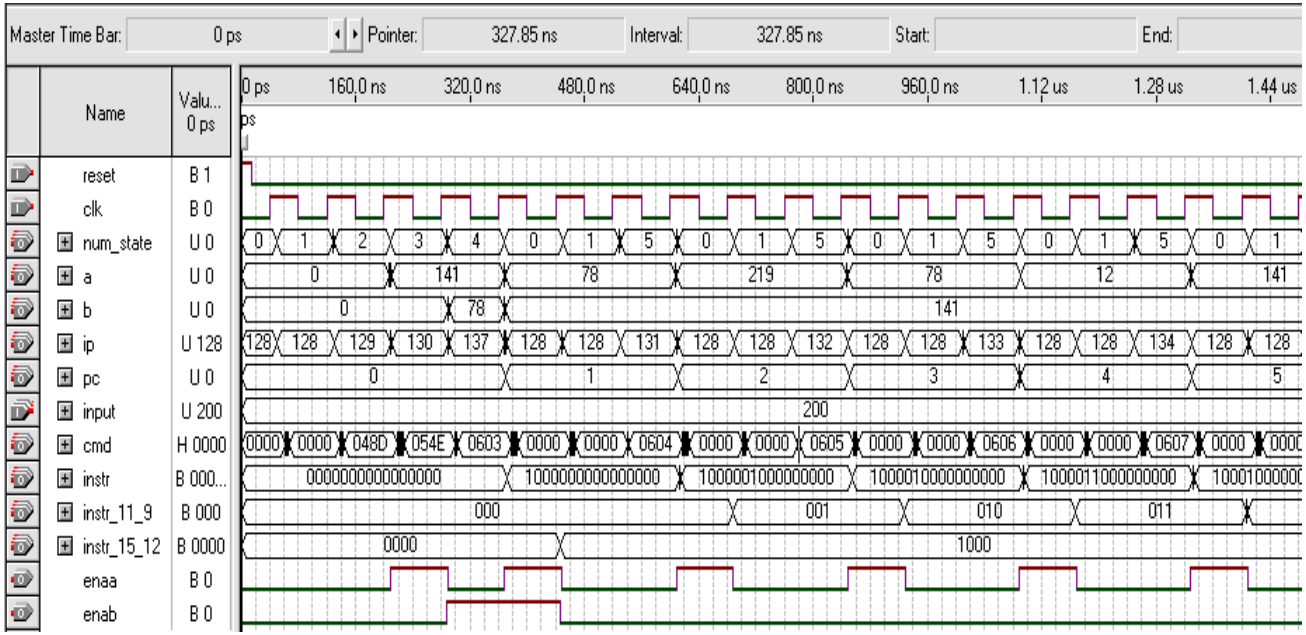


Рис.4.37. Временные диаграммы работы микропроцессорного ядра с конвейерной архитектурой в векторном редакторе САПР ПЛИС Quartus II

Код команды ADD – 604(H) или 1540(D). При этом на шинах instr_15_12 = “1000” а instr_11_9 = “000”. Значения регистров A и B были сложены, и результат помещен в регистр A. Команда SUB A,B выполнила вычитание значений в регистрах A и B, результат помещен в регистр A (код команды – 605(H)). Команда AND A,B, выполняющая операцию побитного логического И значений в регистрах A и B, также показана на рис.4.37. Команда логическое И с кодом 606(H) работает верно. Результат команды был помещен в регистр A. Команда логическое ИЛИ с кодом 607(H), выполняет операцию побитное логическое ИЛИ (команда OR A,B). Результат выполнения команды помещен в регистр A. Команда XOR A,B выполняет побитное логическое исключающее ИЛИ значений в регистрах A и B. Результат помещен в регистр A (код 608(H)).

Особенностью разработанного микропроцессорного ядра с конвейерной архитектурой является использование управляющего автомата и наличие двух блоков памяти: для хранения команд и для хранения инструкций управляющего автомата. При этом АЛУ выполняет только логико-арифметические операции, а прыжковые команды типа JMP реализует управляющий автомат. Проект микропроцессора на языке VHDL может быть успешно размещен в ПЛИС APEX20KE (EP20K160EB356-1), при этом общее число задействованных ресурсов составляет 70 %, с рабочей тактовой частотой до 33 МГц.

4.7. Использование ресурсов ПЛИС Stratix III фирмы Altera при проектировании микропроцессорных ядер

В данном разделе рассматривается использование ресурсов ПЛИС Stratix III при проектировании различных вариантов микропроцессорного ядра, система команд и управляющий автомат взяты из раздела 4.1. Рассматриваются варианты: управляющий автомат и синхронное ПЗУ разработанны на языке VHDL для реализации в базисе ПЛИС APEX20KE (EP20K30ETC144), вариант 1; микропроцессорное ядро для вычислений с фиксированной запятой в системе Matlab/Simulink, код языка VHDL получен с применением Simulink HDL Coder, вариант 2; управляющий автомат и синхронное ПЗУ на язке VHDL в базисе ПЛИС Stratix III EP3SL50F484C2, вариант 3; управляющий автомат на языке VHDL и мегафункция синхронного ПЗУ RAM: 1-PORT в базисе ПЛИС Stratix III, вариант 4; управляющий автомат созданный с применением приложения StateFlow Matlab/Simulink и асинхронное ПЗУ, код языка VHDL получен с применением Simulink HDL Coder в базисе ПЛИС Stratix III EP3SL50F484C2, вариант 5 (табл.4.4).

Используемы ресурсы ПЛИС Stratix III EP3SL50F484C2 и ПЛИС APEX20KE EP20K30ETC144 показаны в таблице. Из анализа таблицы можно сделать выводы, что различные варианты микропроцессорного ядра занимают менее 1 % используемых ресурсов ПЛИС и более чем в два раза большей тактовой частоте чем при реализации в базисе ПЛИС APEX20KE.

Варианты 1, 3 и 4 наиболее схожи между собой, т.к. базируются на одном варианте управляющего автомата. Варианты 2 и 5 базируются лишь на системе команд. Варианты 1, 3 и 4 задействуют одинаковое количество триггеров (33 триггера), а по числу упакованных элементов комбинационной логики ПЛИС Stratix III превосходят ПЛИС

APEX20KE и обеспечивают выигрыш по быстродействию микропроцессорных ядер.

Наиболее удачный вариант по быстродействию – вариант 3, а по функциональной сложности, которая обеспечивает ряд преимуществ, такие как поддержка формата с фиксированной запятой и распределенная система управления блоками процессора – вариант 2. Вариант 2 не использует блоки встроенной памяти. Справедливости ради следует отметить, что данные примеры не позволяют в полной мере оценить используемые ресурсы ПЛИС Stratix III, т.к. проекты слишком малы и позволяют максимально загрузить ПЛИС.

Вариант 2 наиболее близок к архитектуре микропроцессорного ядра PicoBlaze, для реализации в базисе ПЛИС Spartan II, Virtex (рис.4.38) и является его упрощенной версией.

Вариант 2 состоит из следующих блоков: управляющий автомат (блок CPU_Controller); память программ - ПЗУ процессора (блок Memory); АЛУ процессора (блок alu); двух регистров общего назначения (РОН, блоки RegisterA и RegisterB); регистра специального назначения (РСН), выполняющего роль стека (блок PC_Inc); счетчика команд (блок PC); регистра инструкций (блок Instruction_Reg).

Архитектура микропроцессорного ядра PicoBlaze основана на концепции раздельных шин данных и команд (гарвардская или двухшинная архитектура). Память для хранения данных и память для хранения программы располагаются в разных местах, допуская полное совмещение во времени операций вызова команды из памяти ее выполнения, что позволяет добиться высокой скорости выполнения операций. Варианты микропроцессорных ядер (вариант 1-5) условно характеризуются одношинной структурой, т.к. в рассматриваемых вариантах отсутствует память данных.

Таблица 4.4

Используемы ресурсы ПЛИС Stratix III EP3SL50F484C2 и APEX20KE EP20K30ETC144

Stratix III/ APEX20KE	Управляющий автомат и синхронное ПЗУ на языке VHDL APEX20KE EP20K30 ETC144	Микропро-цессорное ядро для вычислений с фиксированной запятой Matlab/ Simulink Stratix III EP3SL50 F484C2	Управляю-щий автомат и синхронное ПЗУ на язке VHDL Stratix III EP3SL50 F484C2	Управляю-щий автомат на языке VHDL, мегафун-кция синхронного ПЗУ RAM: 1-PORT Stratix III EP3SL50 F484C2	Управляющий автомат созданный с применением StateFlow Matlab /Simulink и асинхронное ПЗУ на языке VHDL ПЛИС Stratix III EP3SL50 F484C2
1	2	3	4	5	6
Вариант	1	2	3	4	5
Combinational ALUTs/LUT	116	201	111	106	103
7 входов/4 6/3 5/2 4/1 <=3/0	101 9 23 16 0	8 22 45 29 97	0 6 25 49 31	0 22 45 15 24	1 13 29 27 33

Продолжение табл.4.4

1	2	3	4	5	6
Режимы работы ALUTs: normal mode	-	158	94	89	85
extended LUT mode	-	8	0	0	1
arithmetic mode	-	26	17	17	17
shared arithmetic mode	-	9	0	0	0
Total registers/LC Registers	33	93	33	33	59
ALM/LC	149	119	61	58	64
Memory Bits (1880064)/ (24576)	1536 (6 %)	0	0	4096 (< 1 %)	0
Максимальная тактовая частота, МГц	73.84	318.57 (Slow 1100 mV 85C Model)	327.98 (Slow 1100 mV 85C Model)	245.64 (Slow 1100 mV 85 Model)	275.33 (Slow 1100 mV 85C Model)

Микропроцессорное ядро PicoBlaze содержит 16 восьми - разрядных регистров входящие в блок РОН (в варианте 2 их два), 8 разрядное АЛУ, регистр статуса и регистр фиксации флагов при выполнении обработки прерываний (в варианте 2 отсутствуют), программный счетчик, блок управления вводом/выводом (в варианте 2 отсутствует), стек (15 уровней, в варианте 2 стек организован на регистре R), схема управления прерываниями (в варианте 2 отсутствует), блок управления выбором адреса следующей команды (в варианте 2 отсутствует), дешифратор команд и ПЗУ на основе блочной памяти ПЛИС Block SelectRAM.

Вариант 2 поддерживает лишь несколько команд из 3 групп команд ядра PicoBlaze (всего 6 групп: 1 группа - команды, управляющие последовательностью выполнения операций в программе и команды обработки подпрограмм, например, JUMP, CALL, RETURN; 2 группа – логические команды, например, поразрядное умножение AND; 3 группа – арифметические команды, например, команда получения сумму двух операндов без учета переноса ADD; 4 группа – команды сдвига; 5 группа - команды ввода/вывода; 6 группа – команды для обслуживания прерываний). Ядро PicoBlaze поддерживает всего 49 команд, время выполнения команд - постоянное. В табл.4.5 и табл.4.6 для сравнения показан формат команд переходов JUMP ядра PicoBlaze и варианта 2 с системой команд из раздела 4.1. Ядро PicoBlaze поддерживает 1 безусловную и 3 условных команд переходов. В варианте 2 из за отсутствия развитого АЛУ (например, не предусмотрены арифметические команды с учетом переноса/заема и др), регистра статуса и блока управления выбора следующего адреса, поддерживается лишь одна команда перехода с условием JMPZ.

Трассировочная архитектура MultiTrack, используемая в ПЛИС Stratix III, обеспечивает связь между различными кластерами логических элементов и характеризуется

определенным количеством шагов (hop), необходимых для того, чтобы соединить один LAB с другим. Чем меньше количество шагов и предсказуемее модель трассировки, тем выше производительность и легче оптимизация архитектуры с помощью инструментов САПР.

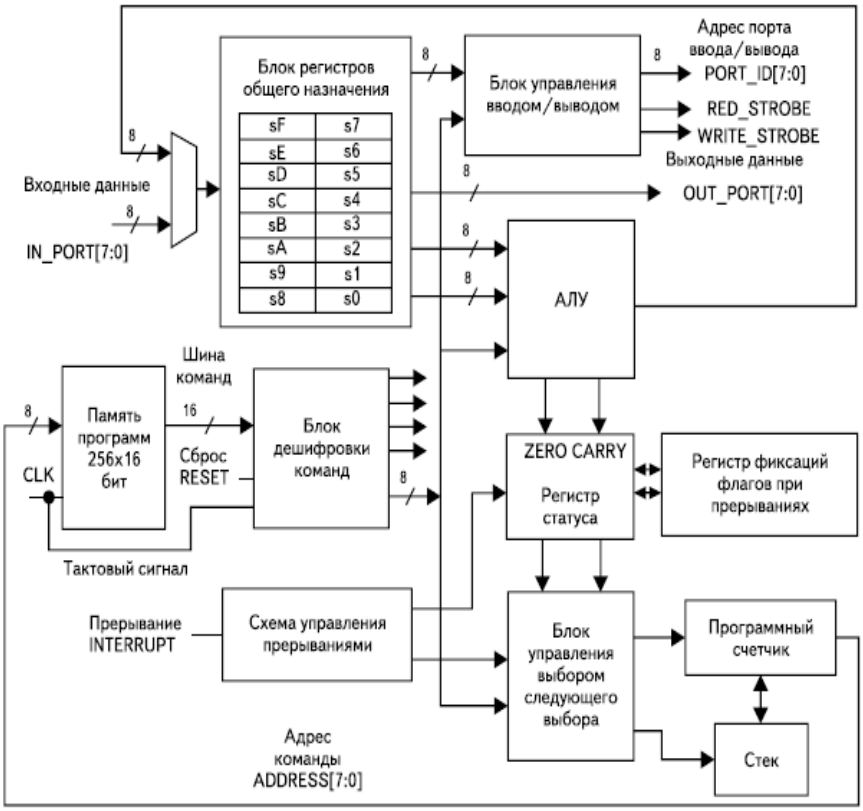


Рис.4.38. Архитектура микропроцессорного ядра PicoBlaze, для реализации в базисе ПЛИС Spartan II, Virtex

Таблица 4.5

Формат команды переходов JUMP микропроцессорного ядра PicoBlaze

Поле кода операции								Поле адреса переходов								Мнемоника	Выполняемая операция
1	0	0	0	x	x	0	1	A	A	A	A	A	A	A	A	JUMP aa	Безусловный переход
1	0	0	1	0	0	0	1	A	A	A	A	A	A	A	A	JUMP Z,aa	Переход при условии, что флаг ZERO Flag находится в установленном состоянии
1	0	0	1	0	1	0	1	A	A	A	A	A	A	A	A	JUMP NZ,aa	Переход при условии, что флаг ZERO Flag находится в сброшенном состоянии
1	0	0	1	1	0	0	1	A	A	A	A	A	A	A	A	JUMP C,aa	Переход при условии, что флаг CARRY Flag находится в установленном состоянии
1	0	0	1	1	1	0	1	A	A	A	A	A	A	A	A	JUMP NC,aa	Переход при условии, что флаг CARRY Flag находится в сброшенном состоянии
15	14	13	12	11	10	9	8	7	6		4	3	2	1	0	Номер разряда команды	

Таблица 4.6

Формат команды переходов микропроцессорного ядра, вариант 2

Поле кода операции								Поле адреса переходов								Мне-мо-ника	Выполняемая операция
0	0	0	0	0	0	0	1	A	A	A	A	A	A	A	A	JMP	Безусловный переход по адресу, заданному младшим байтом команды
0	0	0	0	0	0	1	0	A	A	A	A	A	A	A	A	JMPZ	Переход по адресу, заданному младшим байтом команды, если содержимое регистра A равно нулю
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда команды	

Архитектура трассировки межсоединений MultiTrack обеспечивает большую доступность ко всем окружающим LAB с помощью меньшего числа связей, что позволяет увеличить производительность, снизить энергопотребление и оптимизировать упаковку логики. На рис.4.39 различными цветами (темно-синний и синий) показано число шагов, требующихся для соединения LAB (сноска), с окружающими LAB для реализации микропроцессорного ядра по варианту 2. Таким образом, трассировка до всех LAB выполняется за два шага.

Популярное микропроцессорное ядро oc_oc8051 (микропроцессорное ядро 8051 с сайта независимых разработчиков на интернет ресурсе OpenCores) может быть загружено в ПЛИС Stratix III EP3SL340 85 раз. При этом для реализации ядра oc_oc8051 требуется 4115 логических элементов. Общая емкость ПЛИС Stratix III EP3SL340 составляет 337.5 К логических элементов. ПЛИС Stratix III в среднем на 35 % быстрее ПЛИС Virtex-5, проекты компилируются в три раза быстрее, чем при компиляции в ПЛИС Virtex-5, а коэффициент заполнения кристалла в среднем равен 95%. Микропроцессорное ядро PicoBlaze задействует 9 % логических ресурсов ПЛИС XC2S50E и 2.5 % ПЛИС XC2S300E.

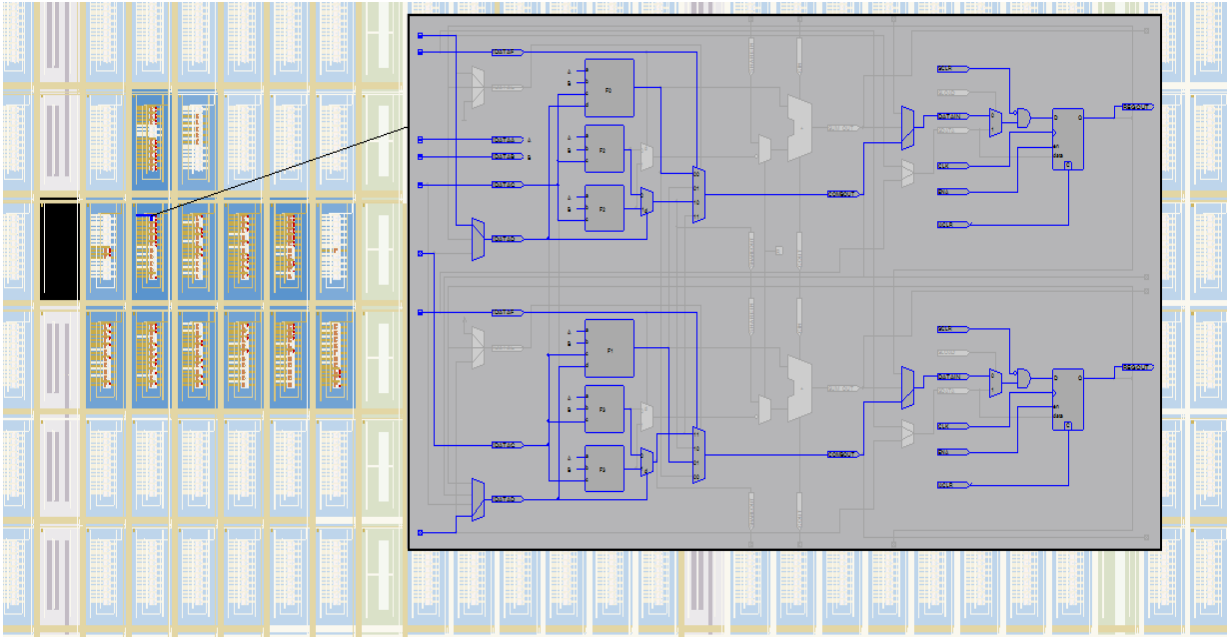


Рис.4.39. Трассировочные ресурсы ПЛИС Stratix III EP3SL50F484C2 задействованные при реализации микропроцессорного ядра по варианту 2

4.8. Проектирование микропроцессорных ядер с использованием приложения StateFlow системы Matlab/Simulink

Заменим описание управляющего автомата (М-файл, табл.1) микропроцессорного ядра визуально-графическим автоматом (граф переходов) построенным с помощью приложения (пакет расширения) StateFlow системы Matlab/Simulink. Далее с помощью приложения Simulink HDL coder сгенерируем код языка VHDL и реализуем модель микропроцессора в базисе ПЛИС Stratix III EP3SL50F484C2 фирмы Altera. Отличительной особенностью данного типа ПЛИС семейства Stratix III является структура памяти TriMatrix с рабочими тактовыми частотами до 600 МГц, которая содержит 108 блоков памяти типа M9K (емкость блока 9216 бит), 6 блоков памяти типа M144K (емкость 147456 бит) и 950 блоков памяти MLAB (емкость 320 бит). Общий объем встроенной памяти ОЗУ 1.836 Кбит, 47.5 К логических элементов (LE), 19 К адаптивных логических модулей (ALM), которые способны работать в различных режимах и 38 К триггеров.

В табл.4.7 приведен М-файл и код управляющего автомата на языке VHDL. Из табл.1 видно, что по М-файлу сгенерирован асинхронный цифровой автомат с использованием двух операторов выбора CASE, представляющий из себя комбинационный дешифратор.

Процессор состоит из следующих блоков (рис.4.40): ROM - ПЗУ команд процессора (память программ); COP – блок выделения полей команды (дешифратор команд); ALU – 8-разрядное АЛУ процессора (управляющий автомат); RON – блок регистров общего назначения (8-разрядные регистры A и B); RSN – блок регистров специального назначения, 8-разрядный регистр R (стек команд) для обеспечения

выполнения команд обращения к подпрограммам (CALL) и возврата (RET) и 8-разрядный регистр Ip (для хранения значений счетчика команд). На рис.4.40 также показана тестовая программа (система команд и содержимое файла прошивки ПЗУ такое же, как и в разделе 4.2)

Для построения ПЗУ используется функциональный блок Lookup Table (таблица соответствия). Все сигналы в процессоре представлены в формате uint8 (Unsigned integer fixed-point data type, целые числа без знака в формате с фиксированной запятой, с 8-ми битной шиной) кроме команд (сигналы cmd и InCmd), они представлены в формате uint16.

Управляющий автомат микропроцессора разработан с помощью приложения StateFlow (рис.4.41). StateFlow является интерактивным инструментом разработки в области моделирования сложных, управляемых событиями систем. Он тесно интегрирован с Matlab и Simulink и основан на теории конечных автоматов. Диаграмма StateFlow - графическое представление конечного автомата, где состояния и переходы формируют базовые конструктивные блоки проектируемой системы.

StateFlow-диаграмма построена из отдельных объектов, таких как состояние, переход, переход по умолчанию и др. Состояние - условия, в которых моделируемая система пребывает некоторое время, в течение которого она ведет себя одинаковым образом. В диаграмме переходов состояния представлены прямоугольными полями со скругленными углами. Например, состояние COMM является родителем состояний MOVAB, RET, MOVBA, XCHG, ADD, SUB, END, OR, XOR, DEC. На рис.4.42 показаны временные диаграмы работы модели микропроцессорного ядра в системе Matlab/Simulink, A, B, R, IP – выходы соответствующих регистров РОН и РСН.

TestProgramm:

		DEC	HEX
0:	MOV A,1	1025	401
1:	MOV B,17	1297	511
2:	CALL 5	773	305
3:	MOV B,18	1298	512
4:	MOV A,2	1026	402
5:	MOV A,3	1027	403
6:	MOV A,4	1028	404
7:	ADD A,B	1540	604
8:	MOV A,6	1030	406
9:	MOV A,7	1031	407
10:	RET	1536	600

0	0	NOP
01xxH	256-511	JMP
02xxH	512-767	JMPZ
03xxH	768-1023	CALL
04xxH	1024-1279	MOV A,xx
05xxH	1280-1537	MOV B,xx
0600H	1536	RET
0601H	1537	MOV A,B
0602H	1538	MOV B,A
0603H	1539	XCHG A,B
0604H	1540	ADD A,B
0605H	1541	SUB A,B
0606H	1542	AND A,B
0607H	1543	OR A,B
0608H	1544	XOR A,B
0609H	1545	DEC A

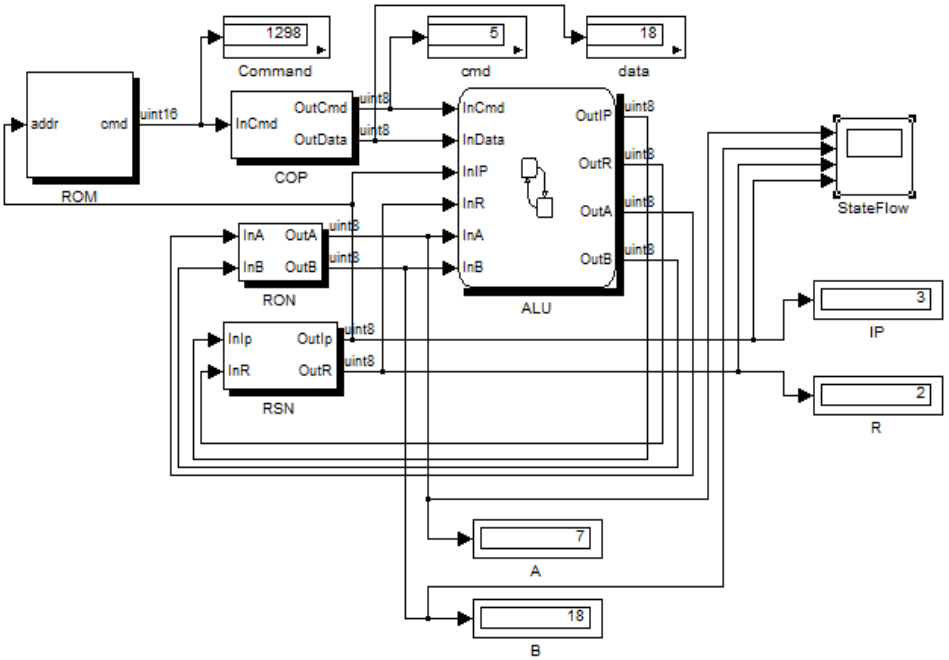


Рис.4.40. Модель микропроцессорного ядра с управляющим автоматом в системе Matlab/Simulink

Таблица 4.7

М-файл и код управляющего автомата на языке VHDL

М-файл управляющего автомата в системе Matlab/Simulink	Управляющий автомат на языке VHDL, код которого получен с помощью Simulink HDL coder
1	2

function [outA, outB, outR, outIp] = alu(inCmd,inData,inA,inB,inIp,inR)
 	
outA = inA;
outB = inB;
outR = inR;
outIp = inIp+1;
switch inCmd
%NOP
case 0
%JMP
case 1
outIp = inData;
%JMPZ
case 2
if inA == 0
outIp = inData;
end
% CALL
case 3
outR = inIp+1;
outIp = inData;
%MOV A,xx
case 4
outA = inData;
%MOV B,xx
case 5
outB = inData;
case 6
switch inData
%RET
case 0
outIp = inR;
%MOV A,B
case 1
outA = inB;
%MOV B,A
case 2
outB = inA;
%XCHG A,B
case 3
X = inB;
outB = inA;
outA = X;
	LIBRARY ieee;
	USE ieee.std_logic_1164.all;
	USE ieee.numeric_std.all;
	use ieee.std_logic_arith.all;
	use ieee.std_logic_unsigned.all;
	ENTITY ALU_entity IS
	PORT (
	inCmd : IN std_logic_vector(7 DOWNTO 0);
	inData : IN std_logic_vector(7 DOWNTO 0);
	inA : IN std_logic_vector(7 DOWNTO 0);
	inB : IN std_logic_vector(7 DOWNTO 0);
	inIp : IN std_logic_vector(7 DOWNTO 0);
	inR : IN std_logic_vector(7 DOWNTO 0);
	outA : OUT std_logic_vector(7 DOWNTO 0);
	outB : OUT std_logic_vector(7 DOWNTO 0);
	outR : OUT std_logic_vector(7 DOWNTO 0);
	outIp: OUT std_logic_vector(7 DOWNTO 0));
	END ALU_entity;
	ARCHITECTURE fsm_SFHDL OF ALU_entity IS
	BEGIN
	PROCESS (inCmd, inData, inA, inB, inIp, inR)
	BEGIN
	outA <= inA;
	outB <= inB;
	outR <= inR;
	outIp <= inIp+1;
	CASE inCmd IS
	WHEN "00000000" =>
	--NOP
	NULL;
	WHEN "00000001" =>
	--JMP
	outIp <= inData;
	WHEN "00000010" =>
	IF inA = 0 THEN
	--JMPZ
	outIp <= inData;
	END IF;
	WHEN "00000011" =>

Продолжение табл.4.7

1	2

%ADD A,B
case 4
outA = inA+inB;
%SUB A,B
case 5
outA = inA-inB;
%AND A,B
case 6
outA = bitand(inA,inB);
%OR A,B
case 7
outA = bitor(inA,inB);
%XOR A,B
case 8
outA = bitxor(inA,inB);
%DEC A
case 9
outA = inA-1;
end
end
	-- CALL
	outR <= inIp+1;
	outIp <= inData;
	WHEN "00000100" =>
	--MOV A,xx
	outA <= inData;
	WHEN "00000101" =>
	--MOV B,xx
	outB <= inData;
	WHEN "00000110" =>
	CASE inData IS
	WHEN "00000000" =>
	--RET
	outIp <= inR;
	WHEN "00000001" =>
	--MOV A,B
	outA <= inB;
	WHEN "00000010" =>
	--MOV B,A
	outB <= inA;
	WHEN "00000011" =>
	--XCHG A,B
	outB <= inA;
	outA <= inB;
	WHEN "00000100" =>
	--ADD A,B
	outA <= inA + inB;
	WHEN "00000101" =>
	--SUB A,B
	outA <= inA - inB;
	WHEN "00000110" =>
	--AND A,B
	outA <= inA AND inB;
	WHEN "00000111" =>
	--OR A,B
	outA <= inA OR inB;
	WHEN "00001000" =>
	--XOR A,B
	outA <= inA XOR inB;
	WHEN "00001001" =>
	--DEC A
	outA <= inA - 1;
	WHEN OTHERS => NULL;
	END CASE;
	WHEN OTHERS => NULL; END CASE; END PROCESS; END fsm_SFHDL;

Переход - это линия со стрелкой, соединяющая один графический объект с другим. В большинстве случаев переход представляет скачок системы из одного режима (состояния) в другой. Переход соединяет объект-источник с объектом-адресатом. Объект-источник - это место, где переход начинается, объект-адресат - это место, где переход заканчивается. Переходы по состояниям характеризуются метками. Метка может включать в себя имя события, условие, действие условия и/или действие перехода. Первоначально переходы помечаются символом (?).

Метки перехода имеют следующий основной формат: event[condition]{condition_action}/transition_action. Любая часть метки может отсутствовать. Условия - это булевы выражения, которые должны быть истинны для осуществления перехода. Условия заключаются в квадратные скобки ([]). Например, условие [InA==0] должно быть истинным для того, чтобы произошло действие условия и переход стал возможен.

Действия условий следуют за условиями и заключаются в фигурные скобки ({}). Они выполняются тогда, когда условие становится истинным, но перед тем, как переход осуществится. Если ни одно условие не определено, подразумеваемое условие принимается за истинное и действие выполняется. Если условие [InA==0] истинно, то действие {OutIP=InData} немедленно выполняется.

Линия со стрелкой и точкой на конце это безусловный переход. Безусловные переходы преимущественно используются для определения, какое последовательное состояние должно стать активным, когда есть неоднозначность между двумя или более ИЛИ-подсостояниями. Безусловные переходы имеют объект-адресат, но у них нет объекта-источника.

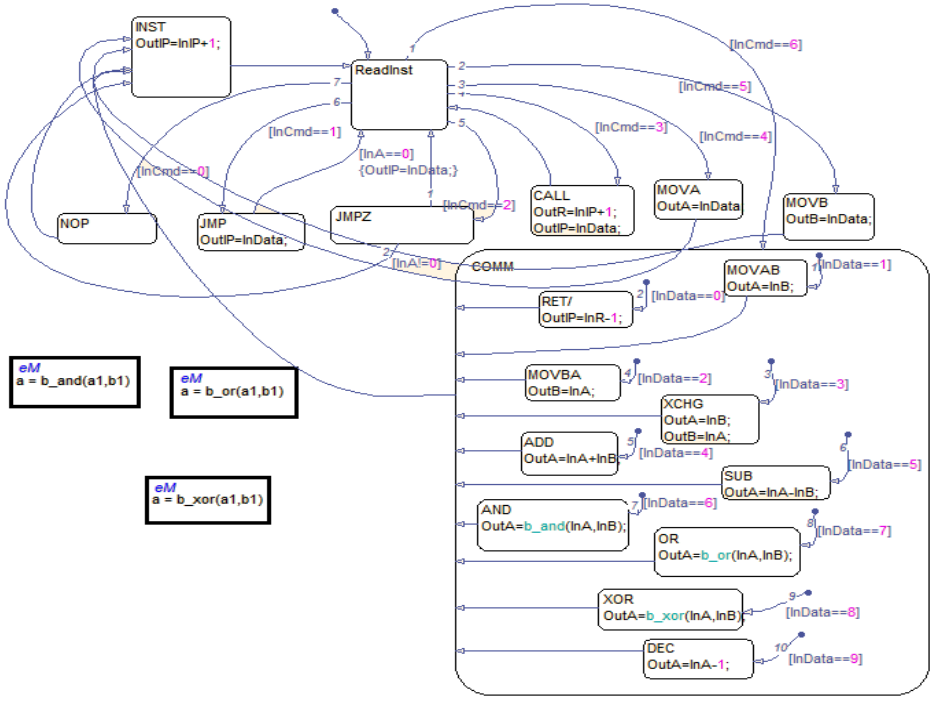


Рис.4.41. Управляющий автомат (блок АЛУ) созданный с помощью приложения StateFlow системы Matlab/Simulink в режиме отладки

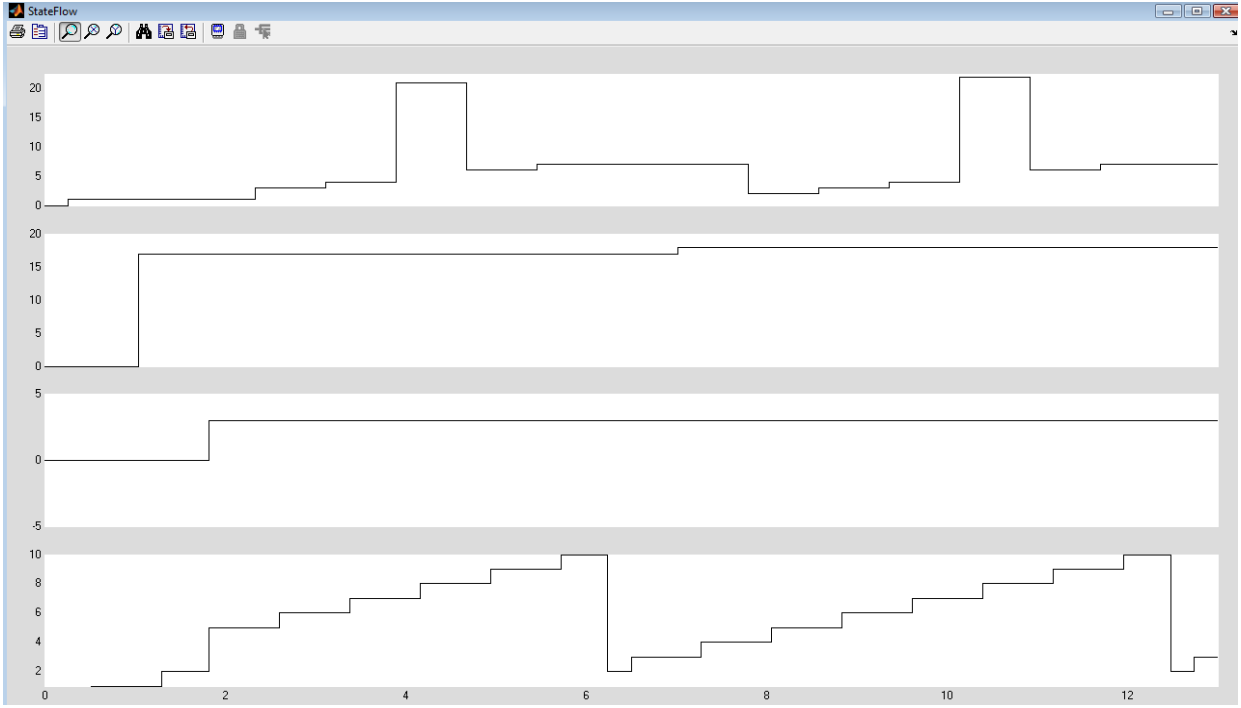


Рис.4.42. Временные диаграммы работы модели микропроцессорного ядра в системе Matlab/Simulink

На рис.4.43 показано микропроцессорное ядро с асинхронным ПЗУ в базисе ПЛИС Stratix III в САПР Quartus II, код языка которого получен с использованием Simulink HDL Coder системы Matlab/Simulink. ПЗУ, дешифратор команд – комбинационные устройства, регистры RON, RSN и ALU – синхронные последовательностные устройства. Для описания управляющего автомата микропроцессора (пример 1) используются перечислимые типы. Перечислимый тип часто используется для обозначения состояний конечных автоматов.

Анализируя код VHDL можно сделать вывод, что сгенерирован синхронный автомат с асинхронным входом reset (активный – высокий уровень) и с синхронным сигналом разрешения тактирования clk_enable. Функциональное моделирование работы микропроцессорного ядра с асинхронным ПЗУ и синхронным управляющим автоматом в базисе ПЛИС Stratix III показано на рис.4.44.

Для сравнения, на рис.4.45 покзано функциональное моделирование работы микропроцессорного ядра с асинхронным ПЗУ и асинхронным управляющим автоматом в базисе ПЛИС Stratix III. Видно, что микропроцессорные ядра работают одинаково, но для микропроцессора с синхронным автоматом требуется большее число тактов на отработку команд и по разному выполняется команда с кодом 0600H (RET). На StateFlow- диаграмме можно по иному организовать выполнение команды вызова и возврата из подпрограммы. Для команды CALL: OutR=InIP и команды RET: OutIP=InR. Сведения по используемым ресурсам ПЛИС представлены в табл.4.8.

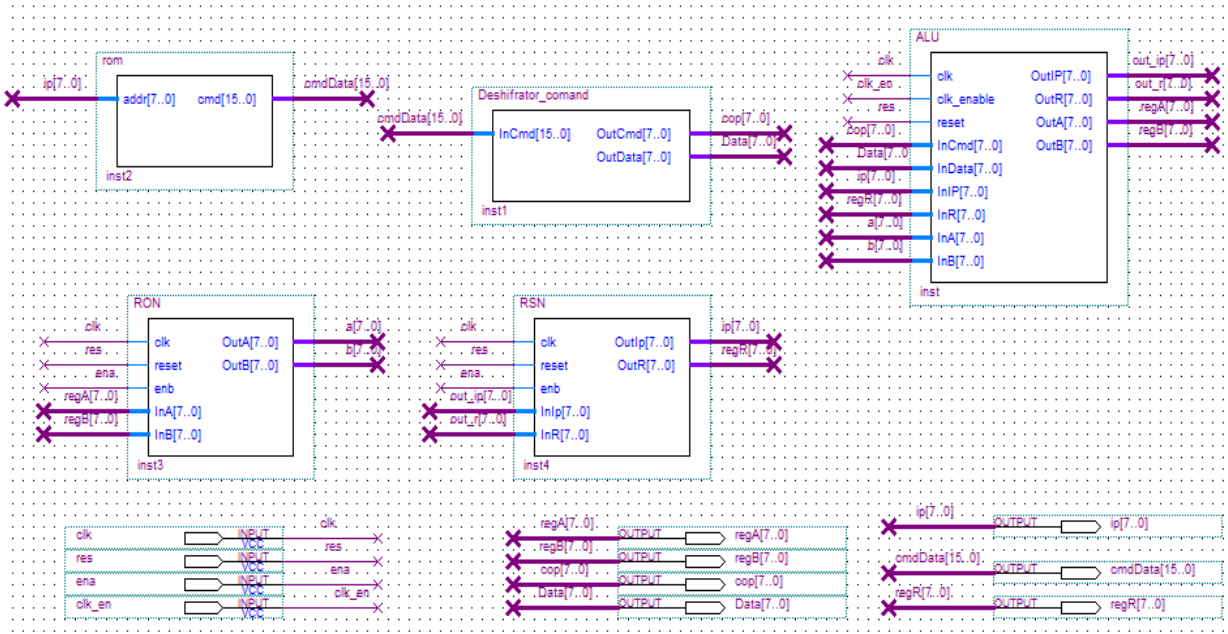


Рис.4.43. Микропроцессорное ядро с асинхронным ПЗУ и синхронным управляющим автоматом в базисе ПЛИС Stratix III в САПР ПЛИС Quartus II, код языка которого получен с использованием Simulink HDL Coder системы Matlab/Simulink

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY ALU IS
   PORT (
      clk : IN std_logic;
      clk_enable : IN std_logic;
      reset : IN std_logic;
      InCmd : IN std_logic_vector(7 DOWNTO 0);
      InData : IN std_logic_vector(7 DOWNTO 0);
      InIP : IN std_logic_vector(7 DOWNTO 0);
      InR : IN std_logic_vector(7 DOWNTO 0);
      InA : IN std_logic_vector(7 DOWNTO 0);
      InB : IN std_logic_vector(7 DOWNTO 0);
      OutIP : OUT std_logic_vector(7 DOWNTO 0);
      OutR : OUT std_logic_vector(7 DOWNTO 0);
      OutA : OUT std_logic_vector(7 DOWNTO 0);
      OutB : OUT std_logic_vector(7 DOWNTO 0));
END ALU;

ARCHITECTURE fsm_SFHDL OF ALU IS

   TYPE T_state_type_is_ALU is (IN_NO_ACTIVE_CHILD, IN_CALL, IN_COMM, IN_INST, IN_JMP, IN_JMPZ, IN_MOVA, IN_MOVB, IN_NOP, IN_ReadInst);
   TYPE T_state_type_is_COMM is (IN_NO_ACTIVE_CHILD, IN_ADD, IN_AND, IN_DEC, IN_MOVAB, IN_MOVBA, IN_OR, IN_RET, IN_SUB, IN_XCHG, IN_XOR);

   SIGNAL is_ALU : T_state_type_is_ALU;
   SIGNAL is_COMM : T_state_type_is_COMM;
   SIGNAL OutIP_reg : unsigned(7 DOWNTO 0);
   SIGNAL OutR_reg : unsigned(7 DOWNTO 0);
   SIGNAL OutA_reg : unsigned(7 DOWNTO 0);
   SIGNAL OutB_reg : unsigned(7 DOWNTO 0);
   SIGNAL is_ALU_next : T_state_type_is_ALU;
```

```
   SIGNAL is_COMM_next : T_state_type_is_COMM;
   SIGNAL OutIP_reg_next : unsigned(7 DOWNTO 0);
   SIGNAL OutR_reg_next : unsigned(7 DOWNTO 0);
   SIGNAL OutA_reg_next : unsigned(7 DOWNTO 0);
   SIGNAL OutB_reg_next : unsigned(7 DOWNTO 0);
BEGIN
   initialize_ALU : PROCESS (reset, clk)
      -- local variables
   BEGIN
      IF reset = '1' THEN
         is_COMM <= IN_NO_ACTIVE_CHILD;
         OutIP_reg <= to_unsigned(0, 8);
         OutR_reg <= to_unsigned(0, 8);
         OutA_reg <= to_unsigned(0, 8);
         OutB_reg <= to_unsigned(0, 8);
         is_ALU <= IN_ReadInst;
      ELSIF clk'EVENT AND clk= '1' THEN
         IF clk_enable= '1' THEN
            is_ALU <= is_ALU_next;
            is_COMM <= is_COMM_next;
            OutIP_reg <= OutIP_reg_next;
            OutR_reg <= OutR_reg_next;
            OutA_reg <= OutA_reg_next;
            OutB_reg <= OutB_reg_next;
         END IF;
      END IF;
   END PROCESS initialize_ALU;
   ALU : PROCESS (is_ALU, is_COMM, OutIP_reg, OutR_reg,
OutA_reg, OutB_reg, InCmd, InData, InIP, InR, InA, InB)
      -- local variables
   BEGIN
      is_ALU_next <= is_ALU;
      is_COMM_next <= is_COMM;
      OutIP_reg_next <= OutIP_reg;
      OutR_reg_next <= OutR_reg;
      OutA_reg_next <= OutA_reg;
      OutB_reg_next <= OutB_reg;
```

```
CASE is_ALU IS
   WHEN IN_CALL =>
      is_ALU_next <= IN_ReadInst;
   WHEN IN_COMM =>
      is_COMM_next <= IN_NO_ACTIVE_CHILD;
      is_ALU_next <= IN_INST;
      OutIP_reg_next <= unsigned(InIP) + 1;
   WHEN IN_INST =>
      is_ALU_next <= IN_ReadInst;
   WHEN IN_JMP =>
      is_ALU_next <= IN_ReadInst;
   WHEN IN_JMPZ =>
      IF unsigned(InA) = 0 THEN
         OutIP_reg_next <= unsigned(InData);
         is_ALU_next <= IN_ReadInst;
      ELSIF unsigned(InA) /= 0 THEN
         is_ALU_next <= IN_INST;
         OutIP_reg_next <= unsigned(InIP) + 1;
      END IF;
   WHEN IN_MOVA =>
      is_ALU_next <= IN_INST;
      OutIP_reg_next <= unsigned(InIP) + 1;
   WHEN IN_MOVB =>
      is_ALU_next <= IN_INST;
      OutIP_reg_next <= unsigned(InIP) + 1;
   WHEN IN_NOP =>
      is_ALU_next <= IN_INST;
      OutIP_reg_next <= unsigned(InIP) + 1;
   WHEN IN_ReadInst =>
      IF unsigned(InCmd) = 6 THEN
         is_ALU_next <= IN_COMM;
         IF unsigned(InData) = 1 THEN
            is_COMM_next <= IN_MOVAB;
            OutA_reg_next <= unsigned(InB);
         ELSIF unsigned(InData) = 0 THEN
            is_COMM_next <= IN_RET;
            OutIP_reg_next <= unsigned(InR) - 1;
```

```
    ELSIF unsigned(InData) = 3 THEN
      is_COMM_next <= IN_XCHG;
      OutA_reg_next <= unsigned(InB);
      OutB_reg_next <= unsigned(InA);
    ELSIF unsigned(InData) = 2 THEN
      is_COMM_next <= IN_MOVBA;
      OutB_reg_next <= unsigned(InA);
    ELSIF unsigned(InData) = 4 THEN
      is_COMM_next <= IN_ADD;
      OutA_reg_next <= unsigned(InA) + unsigned(InB);
    ELSIF unsigned(InData) = 5 THEN
      is_COMM_next <= IN_SUB;
      OutA_reg_next <= unsigned(InA) - unsigned(InB);
    ELSIF unsigned(InData) = 6 THEN
      is_COMM_next <= IN_AND;
      OutA_reg_next <= unsigned(InA AND InB);
    ELSIF unsigned(InData) = 7 THEN
      is_COMM_next <= IN_OR;
      OutA_reg_next <= unsigned(InA OR InB);
    ELSIF unsigned(InData) = 8 THEN
      is_COMM_next <= IN_XOR;
      OutA_reg_next <= unsigned(InA XOR InB);
    ELSIF unsigned(InData) = 9 THEN
      is_COMM_next <= IN_DEC;
      OutA_reg_next <= unsigned(InA) - 1;
    END IF;
  ELSIF unsigned(InCmd) = 5 THEN
    is_ALU_next <= IN_MOVB;
    OutB_reg_next <= unsigned(InData);
  ELSIF unsigned(InCmd) = 4 THEN
    is_ALU_next <= IN_MOVA;
    OutA_reg_next <= unsigned(InData);
  ELSIF unsigned(InCmd) = 3 THEN
    is_ALU_next <= IN_CALL;
    OutR_reg_next <= unsigned(InIP) + 1;
    OutIP_reg_next <= unsigned(InData);
  ELSIF unsigned(InCmd) = 2 THEN
```

```
            is_ALU_next <= IN_JMPZ;
          ELSIF unsigned(InCmd) = 1 THEN
            is_ALU_next <= IN_JMP;
            OutIP_reg_next <= unsigned(InData);
          ELSIF unsigned(InCmd) = 0 THEN
            is_ALU_next <= IN_NOP;
          END IF;
        WHEN OTHERS =>
          is_ALU_next <= IN_ReadInst;
      END CASE;
    END PROCESS ALU;
    OutIP <= std_logic_vector(OutIP_reg_next);
    OutR <= std_logic_vector(OutR_reg_next);
    OutA <= std_logic_vector(OutA_reg_next);
    OutB <= std_logic_vector(OutB_reg_next);
END fsm_SFHDL;
```

Пример.1. Код языка VHDL управляющего автомата модели микропроцессорного ядра на StateFlow, полученный с использованием Simulink HDL Coder системы Matlab/Simulink

Приложение Simulink HDL coder для разработанного управляющего автомата микропроцессора, построенного с помощью StateFlow системы Matlab/Simulink, позволяет сгенерировать код языка VHDL синхронного автомата.

Система визуально-имитационного моделирования Matlab/Simulink с приложениями StateFlow и Simulink HDL Coder может быть эффективно использована для ускорения процесса разработки моделей микропроцессорных ядер.

Проект микропроцессора с асинхронным ПЗУ на языке VHDL может быть успешно размещен в ПЛИС Stratix III EP3SL50F484C2, при этом общее число задействованных логических ресурсов составляет менее 1 %.

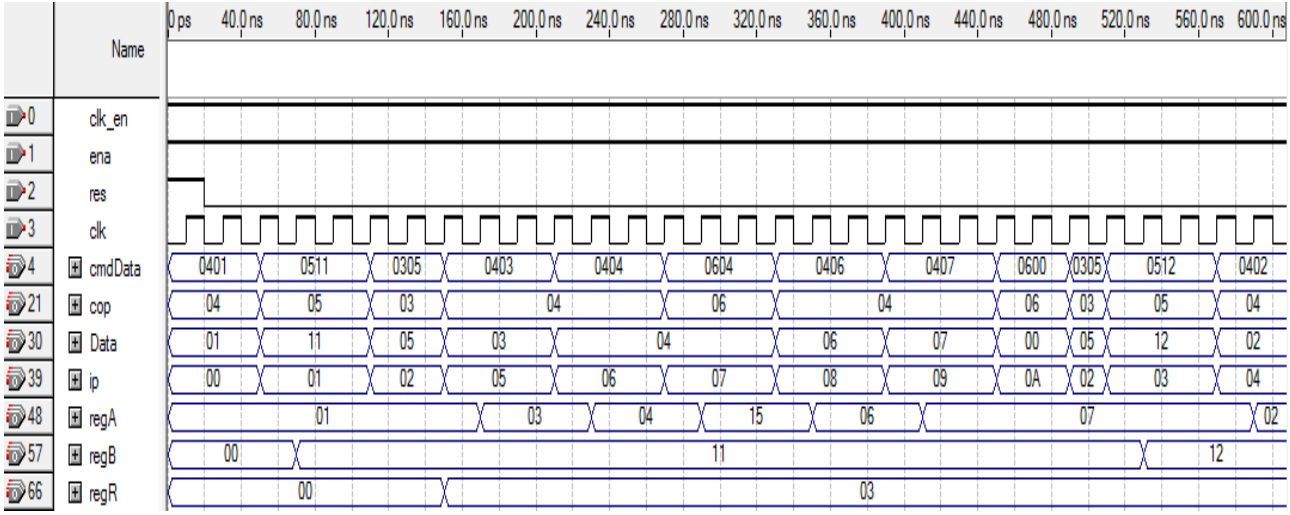


Рис.4.44. Функциональное моделирование работы микропроцессорного ядра с асинхронным ПЗУ и синхронным управляющим автоматом в базисе ПЛИС Stratix III

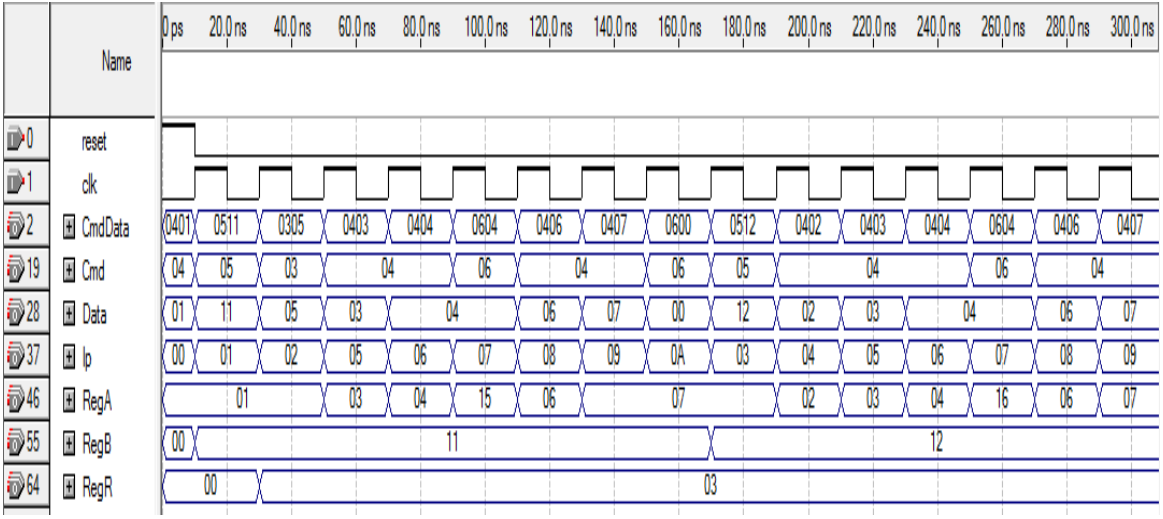


Рис.4.45. Функциональное моделирование работы микропроцессорного ядра с асинхронным ПЗУ и асинхронным управляющим автоматом в базисе ПЛИС Stratix III

Таблица 4.8

Используемы ресурсы ПЛИС Stratix III EP3SL50F484C2

Проект	Логические ресурсы		Максимальная тактовая частота синхро-сигнала, $f_{\max CLK}$, МГц
	Адаптивные таблицы перекодировок ALUTs, для реализации комбинационных функций	Выделенные триггеры логических элементов	
Асинхронное ПЗУ, асинхронный управляющий автомат (М-файл)	82	26	360
Асинхронное ПЗУ, синхронный управляющий автомат (StateFlow)	103	59	275

В данной главе показаны основы проектирования микропроцессорных ядер для реализации в базисе ПЛИС с использованием системного уровня проектирования с привлечением системы визуально-иммитационного моделирования аналоговых и дискретных систем Matlab/Simulink. Рассматриваются различные подходы в проектировании управляющего автомата микропроцессорного ядра: на языке VHDL, с использованием графического представления конечного автомата с помощью приложений StateFlow и Simulink HDL coder, с использованием М-файлов системы Matlab/Simulink.

ЗАКЛЮЧЕНИЕ

Для современных коммерческих ПЛИС типа ППВМ характерны следующие архитектурные особенности: наличие многоуровневой структуры межсоединений; объединение логических блоков в кластеры; широкое использование коммутаторов-маршрутизаторов; смещение схемотехники в сторону использования технологии соединений single-driver; сегментированные межсоединения в трассировочных каналах различной длины.

Академические ПЛИС являются хорошей основой для разработки новых перспективных видов ПЛИС, таких как 3D ПЛИС и ПЛИС с конфигурационной памятью на нанотрубках. Ведущие мировые дизайн-центры и учебные образовательные центры широко используют программные инструменты T-Vpack и VPR как для проектирования, так и для исследования новых архитектур ПЛИС типа ППВМ.

В настоящее время разработчики как коммерческих, так и академических ПЛИС пришли к выводу о целесообразности использования однонаправленных сегментированных межсоединений различной длины в трассировочных каналах и использования мультиплексорных структур в соединительных блоках и коммутаторах-маршрутизаторах, что позволяет получать существенный выигрыш по быстродействию и по площади кристалла.

Рассмотрены различные подходы в проектировании микропроцессорных ядер для реализации в базисе ПЛИС с использованием системы визуально-имитационного моделирования Matlab/Simulink и САПР ПЛИС Quartus II фирмы Altera.

Микропроцессорные ядра, представленные в виде сложно-функциональных блоков в базисе ПЛИС, позволяют реализовать современную концепцию “система на кристалле”. Использование более высокой степени абстракции в проектировании БИС и сложно-функциональных блоков в виде готовых модулей позволяют создавать конкурентоспособные изделия в кратчайшие сроки.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Уилкинсон Б. Основы проектирования цифровых схем: пер. с англ. / Б. Уилкинсон. М.: Издательский дом Вильямс, 2004. 320 с.

2. Армстронг Дж. Р. Моделирование цифровых систем на языке VHDL: пер. с англ. / Р. Дж. Армстронг. М.: Мир, 1992. 348 с.

3. Максфилд К. Проектирование на ПЛИС: курс молодого бойца: пер. с англ. / К. Максфилд. М.: Издательский дом Додэка XXI, 2007. 408 с.

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

5. Рабаи Ж.М. Цифровые интегральные схемы. Методология проектирования. / Ж.М. Рабаи, А. Чандракасан, Б. Николич. М.: Вильямс, 2007. - 911 с.

6. Угрюмов Е.П. Цифровая схемотехника / Е.П. Угрюмов. СПб.: БХВ, 2004. 528 с.

7. Стешенко В. ПЛИС фирмы ALTERA: проектирование устройств обработки сигналов / В. Стешенко. М.: Додэка, 2000. 457 с.

8. Программируемые логические ИМС на КМОП-структурах и их применение / П.П. Мальцев, Н.И. Гарбузов, А.П. Шарапов, Д.А. Кнышев – М.: Энергоатомиздат. 1998. – 160 с.

9. Строгонов А.В. Проектирование сложно-функциональных блоков в базисе ПЛИС: учеб. пособие / А.В. Строгонов, С.А. Цыбин. Воронеж: ГОУВПО “Воронежский государственный технический университет”, 2010. 333 с.

10. Строгонов А. ПЛИС типа ППВМ с одноуровневой структурой межсоединений / А. Строгонов, М. Мотылев, С. Давыдов, А. Быстрицкий // Компоненты и технологии. 2011. N1. С.14-19.

11. Строгонов А. ПЛИС в ПЛИС или как спроектировать самому / А. Строгонов, М. Мотылев, С. Давыдов, А. Быстрицкий, С. Цыбин // Компоненты и технологии. 2011. N4. С.68-73.

12. Строгонов А. Проектирование академических ПЛИС типа ППВМ с одноуровневой структурой межсоединений / А.

Строгонов, М. Мотылев, С. Давыдов, А. Быстрицкий, С. Цыбин // Компоненты и технологии. 2011. N6. С.64-69.

13. Строгонов А.В. Программируемая коммутация межсоединений в ПЛИС типа программируемые пользователем вентильные матрицы / Строгонов А.В., С.И. Давыдов, М.С. Мотылев, А.В. Быстрицкий // Вестник ВГТУ. 2011. N 8. С.21-24.

14. Строгонов А.В. Проектирование учебного процессора для реализации в базисе ПЛИС / А.В. Строгонов // Компоненты и технологии. 2009. N3. С.6-9.

15. Строгонов А.В. Проектирование учебного процессора для реализации в базисе ПЛИС с использованием системы Matlab/Simulink / А.В. Строгонов, А.И. Буслов // Компоненты и технологии. 2009. N5. С.10-14.

16. Строгонов А.В. Проектирование микропроцессорных ядер с конвейерной архитектурой для реализации в базисе ПЛИС фирмы Altera / А.В. Строгонов, С.И. Давыдов // Компоненты и технологии. 2009. N8. С.76-79.

17. Строгонов А.В. Проектирование учебного процессора с фиксированной запятой в системе Matlab/Simulink / А.В. Строгонов // Компоненты и технологии. 2009. N7. С.22-27.

18. Строгонов А.В. Проектирование учебного процессора с фиксированной запятой в САПР Quartus II компании Altera / А.В. Строгонов, А.И. Буслов, С.И. Давыдов // Компоненты и технологии. 2009. N11. С.20-25.

19. Строгонов А.В. Использование различных типов памяти при проектировании учебного микропроцессорного ядра для реализации в базисе ПЛИС / А.В. Строгонов, С.А. Цыбин // Компоненты и технологии. 2009. N12. С.92-96.

20. Строгонов А.В. Проектирование микропроцессорных ядер с использованием приложения StateFlow системы Matlab/Simulink / А.В. Строгонов, С.А. Цыбин, А.И. Буслов // Компоненты и технологии. 2010. N1. С.66-70.

21. Строгонов А.В. Использование ресурсов ПЛИС Stratix III фирмы Altera при проектировании микропроцессорных ядер / А.В. Строгонов, С.А. Цыбин // Компоненты и технологии. 2010. N2. С.70-73.

ОГЛАВЛЕНИЕ

Учебное издание

Строгонов Андрей Владимирович

СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ ИНТЕГРАЛЬНЫХ СХЕМ

В авторской редакции

Компьютерная верстка А.В. Строгонова

Подписано к изданию 26.03.2012

Объём данных 45 МБ

ФГБОУ ВПО «Воронежский государственный технический университет»
394026 Воронеж, Московский просп., 14