УДК 004.3
ББК 32.973-02

Рецензенты:
Стенин В.Я., доктор технических наук, профессор,
Лазутин Ю.М., доктор физико-математических наук,

С.Г. Бобков *Высокопроизводительные вычислительные системы* / *под ред. академика РАН В.Б. Бетелина.* –
М.: НИИСИ РАН, 2014. – 296 с.
ISBN

В представленном учебном пособии для студентов старших курсов и аспирантов специальности автоматика и электроника расмотрены основные проблемы создания современных высокопроизводительных микропроцессорных систем и пути их решения. Рассмотрены архитектуры микропроцессоров и коммуникационных систем, ориентированных на создание высокопроизводительных вычислительных систем вплоть до суперЭВМ. Основными ограничивающими факторами развития высокопроизводительных микропроцессоров и вычислительных систем в целом является повышение потребления питания СБИС и снижение их надёжности при переходе на суб-100 нм технологические нормы изготовления и увеличении числа транзисторов в СБИС до сотен миллионов. Приводятся основные подходы решения этих задач. Рассмотрены маршрут и методики проектирования высокопроизводительных цифровых микросхем.

УДК 004.3
ББК 32.973-02

ISBN

Оглавление

Список основных сокращений и условных обозначений, словарь терминов

№	Сокращение на русском яз.	Сокращение на английском яз.	Расшифровка
1.		*ATPG*	*Automatic Test Pattern Generation* – метод/технология *EDA*, используемая для поиска входной (или тестовой) последовательности, которая при воздействии на цифровые схемы позволяет испытательному устройству отличать корректную работу схемы от некорректной, вызванной дефектами.
2.		*COTS*	*Commercial (components) of the Shelf* – коммерческие (приборы) со склада
3.		*DARPA*	*Defense Advanced Researched Projects Agency* - Управление перспективных исследований МО США.
4.		*DFT*	*Design for Test* – название метода проектирования, который обеспечивает дополнительные функции тестопригодности микроэлектронного изделия.
5.		*DRC*	*Design Rule Check* – область автоматизации проектирования электроники (*EDA*), которая определяет, удовлетворяет ли физическая топология конкретной микросхемы технологическим проектным нормам завода изготовителя.
6.		*Dynamic IP*	Динамический *IP* адрес: *IP* адрес, который назначается каждый раз, когда компьютер входит в сеть.
7.		*EDA*	*Electronic Design Automation* – категория инструментов для проектирования и производства электронных систем от печатных плат до интегральных схем.
8.		*EPx*	*Endpoint(s)* – конечная точка (точки). Понятие коммуникационных стандартов, определяющее источник или приемник потока данных.
9.		*ESA*	*European Space Agency* – Европейское космическое агентство.
10.		*FEC*	Forward Error Correction – система контроля ошибок передачи данных, при которой отправитель добавляет избыточные данные к своим сообщениям.
11.		*QES*	*Quick Engineering Sample* – сокращённый процесс изготовления микросхем по сравнению со стандартным процессом за счет сокращения некоторых этапов (таких как измерения или тестирование пластины).

№	Сокращение на русском яз.	Сокращение на английском яз.	Расшифровка
12.		*GDS*	*Graphic Data System* – формат файлов баз данных, который фактически является отраслевым стандартом для изготовления фотошаблонов микросхем или других микроэлектронных структур.
13.		*IP*	*IP* адрес - уникальный адрес, который присваивается каждому устройству, получающему доступ в сеть.
14.		*ITRS*	*The International Technology Roadmap for Semiconductors* – международный сетевой график развития полупроводниковых приборов.
15.		*LDPC*	*Low-density Parity-check* – линейный код исправления ошибок, метод передачи сообщения по шумному каналу передачи, при его построении используется разреженный двудольный граф.
16.		*LVS*	*Layout vs. Schematic* – класс проверочного программного обеспечения *EDA*, которое определяет, соответствует ли топология конкретной интегральной схемы первоначальной схематике или диаграмме проектируемой схемы.
17.		*MAC*	*Media Access Control* - подуровень канального уровня (уровень 2) в составе семиуровневой *OSI*-модели. Он обеспечивает механизмы контроля адресации и канального доступа, которые позволяют нескольким терминалам или сетевым узлам связываться внутри многоточечной сети, как правило, локальной компьютерной сети (*LAN*) либо общегородской компьютерной сети (*MAN*).
18.		*PHY*	*Physical Layer* - первый, самый нижний уровень в семиуровневой *OSI*-модели системного взаимодействия, состоящий из технологий передачи данных между основными аппаратными средствами сети.
19.		*PLL*	*Phase Lock Loop* – система контроля, которая генерирует сигнал, имеющий фиксированное отношение к фазе "эталонного" сигнала.
20.		*QAM*	*Quadrature Amplitude Modulation* – как аналоговая, так и цифровая схема модуляции. Она передает два аналоговых информативных сигнала либо два цифровых потока двоичных сигналов путем изменения (модуляции) амплитуд двух несущих частот с использованием

№	Сокращение на русском яз.	Сокращение на английском яз.	Расшифровка
			схемы цифровой модуляции амплитудной манипуляции (*ASK*) либо схемы аналоговой модуляции амплитудной модуляции (*AM*).
21.		*RS*	Reed-Solomon coding – код Рид-Соломона - исправления ошибок, который работает на основе избыточности полинома, построенного из данных.
22.		*RTL*	*Register Transfer Level* – способ описания работы синхронной цифровой схемы. При проектировании *RTL*, работа схемы определяется в контексте потока сигналов (или передачи данных) между регистрами аппаратных средств и логических операций, выполняемых по данным сигналам.
23.		*Static IP*	Статический *IP* адрес - фиксированный IP адрес назначаемый компьютеру, необходимый для веб-серверов.
24.		*Hardware*	Аппаратное обеспечение, физическое оборудование.
25.	ДОЗУ	*DRAM*	Динамическое ОЗУ.
26.	ИС	*IC*	Интегральная схема.
27.	КМОП	*CMOS*	Комплементарные транзисторы структуры «металл-окисел-полупроводник».
28.	КНИ	*SOI*	Кремний на изоляторе.
29.	МК	*MC*	Микроконтроллер.
30.	МП	*CPU*	Микропроцессор.
31.	МПР	*MPR*	Микропериферия.
32.	ОЗУ	*RAM*	Оперативное запоминающее устройство.
33.	ПЗУ	*ROM*	Постоянное запоминающее устройство.
34.	ПО	*Software*	Программное обеспечение.
35.	ПЛИС	*FPGA*	Программируемые пользователем логические ИС, содержащие ячейки с рядами несоединённых транзисторов и резисторов. Соединение производится пользователями для выполнения требуемых им функций. без участия завода-производителя.
36.	СвК	*SiP*	Система в корпусе (несколько кристаллов микросхем, находящихся в одном корпусе или в модуле).
37.	СОЗУ	*SRAM*	Статическое ОЗУ.
38.	СИС	*ASIC*	*Application Specific Integrated Circuit* - Специализированные (заказные) ИС, выполняющие определённые пользовательские требования в отличие от таких микросхем как память или микропроцессор, являющимися универсальными микросхемами.

№	Сокращение на русском яз.	Сокращение на английском яз.	Расшифровка
39.	СнК	*SoC*	Система на кристалле (функционально законченный вычислительный узел, включающий помимо ядра микропроцессора комплект системного контроллера, периферийных и сетевых контроллеров).
40.	УФ	*UV*	Ультрафиолетовый диапазон.
41.	ЦПС	*DSP*	Цифровой процессор сигналов.

Bit stuffing – вставка бита по специальному алгоритму в последовательном потоке битов на линии. Применяется для целей синхронизации или уменьшения/устранения постоянной составляющей в сигнале.

Bus powered devices – устройства *USB*, получающие питание +5В от шины *USB* (линия *VBUS*). Питающее напряжение на линии VBUS формирует хост.

Daisy chained – термин, относящийся к линейной топологии сети, когда сигналы последовательно передаются от одного устройство к последующему. Обычно, если текущий обмен относится к устройству, к которому поступил(и) сигнал(ы) по последовательной цепочке, распространение сигналов дальше по цепи прекращается и текущее устройство начинает выполнять требуемую функцию.

Downstream – нисходящее соединение. Такое соединение имеет место для хоста по отношению к устройству в древовидной иерархии.

Errata – исправления, добавления к стандарту или документу.

Frame Work – рабочая среда, рабочее окружение.

Handshaking – процедура установления связи.

Host – хост, главное устройство на шине. Обычно это компьютер.

Latency –время задержки на передачу пакетов.

Padding – дополнение байтами до нужного количества (обычно нулевыми байтами).

Payload – полезные данные, передаваемые в пакете.

PCB – печатная плата.

Peer-to-peer – возможность обмена данными между равноправными устройствами.

Pipe(s) – канал. Означает специальным образом сформированные частные потоки данных через коммуникационный интерфейс.

Pull down – нагрузочный для сигнальной линии резистор, подключенный между линией сигнала и землей.

Pull up – нагрузочный для сигнальной линии резистор, подключенный между линией сигнала и плюсом питания.

Self powered devices - устройства *USB*, питающиеся от отдельного источника (не от шины USB).

Status reporting – получение информации о статусе устройства.

Token – символ.

Token Packet – пакет, показывающий, какие дальше идут данные.

Upstream – восходящее соединение. Такое соединение имеет место для устройства в древовидной иерархии по отношению к хосту.

Хаб: в общем смысле узел какой-то сети. В компьютерной технике сетевой концентратор или сетевое устройство, предназначенное для объединения нескольких устройств в общий сегмент сети и устанавливающее в каждый момент времени только одно соединение. Устройства подключаются при помощи витой пары, коаксиального кабеля или оптоволокна. В настоящее время хабы почти не выпускаются — им на смену пришли сетевые коммутаторы, выделяющие каждое подключённое устройство в отдельный сегмент.

Шлюз (*Gateway*): устройство сопряжения, соединяющее два разных типа сетей.

Введение

Вычислительные системы в современном мире во многом определяют прогресс общества, удобство и комфорт жизни людей. От того, насколько надёжными, быстродействующими и удобными в пользовании являются вычислительные системы, зависит комфорт и качество нашей жизни. Производительность является основной характеристикой вычислительных систем. Имея высокопроизводительный компьютер, ребёнок скажет, что он сможет играть в интересные игры, научные работники получат быстрее результаты работы (в некоторых случаях эти результаты можно получить только при наличии самых мощных компьютеров), военные разработчики получат возможность создавать высокоточное оружие, определять на больших расстояниях наличие противника и т.д. Однако для того чтобы получить требуемые высокопроизводительные системы и для профессиональной деятельности, и для бытовых задач требуется решения комплекса мер по поднятию производительности с одновременным учётом таких факторов как ограничения по потребляемой мощности, стоимости, размеру кристалла, надёжности функционирования.

Основным фактором повышения производительности является создание высокопроизводительного микропроцессора. Производительность микропроцессора определяется рядом факторов: частотой его функционирования; архитектурой: возможностью одновременного выполнения нескольких инструкций, обрабатывать несколько потоков данных, наличием нескольких ядер, архитектурными особенностями (например, возможностью аппаратного выполнения сложных функций, таких как $\sin x$, бабочка и пр.); наличием большого объёма встроенных кэш памятей различного уровня (встроенная подсистема памяти) и системой команд.

Следующим фактором является организация подсистемы памяти в целом. Особенно остро возникает задача повышения скорости обмена данными с ОЗУ для задач с большими объёмами данных, не позволяющих эффективно использовать кэш память. В таких случаях ускорение возможно, прежде всего, за счёт введения режимов прямого доступа к памяти (*DMA*), предвыборок, увеличение частоты и ширины памяти.

Для систем реального времени и многопроцессорных систем важно аппаратное выполнение функций синхронизации процессов, организация прерываний и контроля функционирования вычислительной системы и её отдельных узлов.

Большие потери производительности (до нескольких раз для отдельных задач) вычислительных систем происходят при обмене между отдельными микросхемами в силу ограничений по скорости передач данных по плате. Поэтому интеграция функций на 1 кристалле и создание систем на кристалле приводит не только к повышению надёжности системы и уменьшению её габаритов в силу уменьшения числа компонент, но и к повышению производительности системы в целом.

При создании микросборок *SiP* возможна оптимизация выводов кристаллов с учётом их характеристик и числа в микросборке и соответствующее снижение ёмкости контактов. Это приводит к уменьшению потребления питания и увеличению скорости обмена данными.

Уменьшение числа мостовых схем приводит к уменьшению задержки на передаваемые данные.

Создание комплектов СБИС под вычислительные системы позволяет оптимизировать потребляемую мощность, габариты и повысить производительность системы.

Введение сопроцессоров под выделенные задачи позволяет в несколько раз поднять производительность систем на таких задачах. Для примера, в современных коммуникационных микропроцессорах имеются десятки встроенных сопроцессоров, аппаратно выполняющих функции сжатия данных, криптографии, контроля, *TCP* и пр. [1,2].

Следующими шагами повышения производительности является создание многопроцессорных модулей и систем. Эффективность многопроцессорной системы во много зависит от решаемых задач (возможностью параллельно выполнения отдельных частей программы) и программного обеспечения, обеспечивающего параллельное выполнение задач. Особая роль в таких системах с точки зрения достижения требуемых параметров производительности отводится коммуникационным системам. Именно поэтом в дальнейшем им будет уделено значительное внимание.

Глава 1. Создание высокопроизводительных микропроцессоров

1.1. Ограничения роста производительности микропроцессоров и пути повышения производительности

Главными характеристиками микропроцессоров являются тактовая частота, производительность, энергопотребление и архитектура. Ключевым показателем микропроцессора является его производительность. Определим, от чего зависит производительность микропроцессора (CPU). Время выполнения программы $T_{прог}$ можно определить как:

$$T_{прог} = \frac{N_{цик}}{F_{пр}} \quad \text{или} \quad T_{прог} = N_{цик} \times t_{цик} \tag{1.1}$$

где $N_{цик}$- число циклов микропроцессора при выполнении программы, $F_{пр}$- частота микропроцессора, $t_{цик}$ – длительность такта микропроцессора – обратная величина $F_{пр}$.

Помимо числа циклов нужно учесть число выполненных инструкций ($N_{инст}$). Если известно число циклов и число инструкций, можно расчитать среднее число тактов на инструкцию ($CPI_{инст}$):

$$CPI_{инст} = \frac{N_{цик}}{N_{инст}} \tag{1.2}$$

Подставляя значение $N_{цик}$из 1.2 в формулу 1.1 получаем:

$$T_{прог} = N_{инст} \times CPI_{инст} \times t_{цик} \tag{1.3}$$

David Patterson и John Hennessy представили эту формулу как [3]:

$$\frac{N_{инст}}{Программа} \times \frac{N_{цик}}{N_{инст}} \times \frac{T_{прог}}{N_{цик}} = \frac{T_{прог}}{Программа} = время\ работы\ CPU$$

Как видно, производительность зависит от трёх характеристик - частоты, частоты на инструкцию, и количества инструкций. Более того, время работы CPU равно зависит от каждой из них - увеличение на 10% одной даёт общий прирост так же в 10%. К сожалению, сложно изменить один из этих параметров в изоляции от остальных, поскольку основные технологии, определяющие каждую характеристику, являются взаимозависимыми:

частота - технология изготовления микропроцессора,

частота на инструкцию - архитектура микропроцессора,

количество инструкций - система команд и технологии компиляции.

В соответствие с эмпирическим наблюдением, сделанным Гордоном Муром в 1965 году, названным в дальнейшем законом Мура, число транзисторов на кристалле удваивается каждые 24 месяца. Существует масса схожих утверждений, которые характеризуют процессы экспоненциального роста, также именуемых «законами Мура». К примеру, менее известный «второй закон Мура», введённый в 1998 году Юджином Мейераном, который гласит, что стоимость фабрик по производству микросхем экспоненциально возрастает с усложнением производимых микросхем. В 2007 году Мур заявил, что закон, очевидно, скоро перестанет действовать из-за атомарной природы вещества и ограничения скорости света.

Одним из физических ограничений на миниатюризацию электронных схем является также Принцип Ландауэра, согласно которому логические схемы, не являющиеся обратимыми, должны выделять тепло в количестве, пропорциональном количеству стираемых (безвозвратно потерянных) данных. Возможности же по отводу тепла физически ограничены.

Улучшение технологических норм изготовления кристаллов микросхем и увеличение числа транзисторов в микросхемах примерно до 2003 г. приводило также к постоянному росту частоты микропроцессоров вплоть до частот 3-4 ГГц. Однако дальнейшее повышение частоты стало приводить к существенному повышению потребления питания и соответственно к повышению тепловыделения, которое невозможно отвести от кристаллов стандартными средствами. Рекордная частота принадлежит микропроцессору *AMD*, тактовая частота которого составила чуть больше 8,4 ГГц. Для его охлаждения требуется жидкий азот. Все процессоры, которые в настоящее время производятся в промышленных объёмах, работают на частоте менее 4 ГГц. Следующее ограничение, повышение частоты достигается в основном за счёт увеличения количества ступеней в конвейере микропроцессора, что приводит к большим потерям времени при необходимости перезагрузки конвейера при конфликте

по управлению и переключении на новую задачу. Повышение производительности стало достигаться за счёт усложнения ядра микропроцессора, включением дополнительных функций, сопроцессоров и, главное, за счёт создания многоядерных микропроцессоров.

Переход на многоядерность связан также и с проблемами разброса параметров транзисторов и существенном возрастании токов утечки при использовании топологических норм 45 нм и ниже. В результате для достижения предельных параметров становится необходимо оптимизировать каждый проект под конкретный технологический маршрут. Диапазон вариации электрических параметров внутри одного и того же кристалла составляет 1-3 мм. Если размер модуля СБИС не превышает 2 мм, внутри модуля сохраняются корреляции статических параметров элементов и возможна схемотехническая компенсация технологических вариаций.

Оценки основных параметров локального вычислительного модуля, сделанные фирмой *Intel* для КМОП технологии с технологическими нормами 45 нм, показывают, что при тактовой частоте 500 МГц и разбросе фронтов сигналов не более 30 % от периода, геометрическая область, в которой сохраняются условия синхронизации составляет 2×2 мм^2 [4]. Этой области вполне достаточно для создания законченного вычислительного узла.

Серьезные проблемы при технологических нормах ниже 45 нм возникают с перекрестными наводками и импульсными помехами.

Таким образом, ограничения технологии производства кристаллов микросхем дают дополнительный толчок к созданию многоядерных процессоров с ограниченными размерами ядер. Сдерживающим фактором развития этого направления являются ограничения современного программного обеспечения, не позволяющего в полной мере эффективно использовать многоядерность. Однако тенденция в создании многоядерных и многопроцессорных систем очевидна. Эффективность функционирования многоядерных систем во многом определяется скоростями обмена между ядрами и архитектурой связи ядер - «сети на кристалле». Архитектура микросхем «сеть на кристалле» ориентирована на использование физически и программно совместимых модулей. Каждый модуль включает микропроцессор и средства коммутации. Модули образуют вычисли-

тельную сеть с множествами каналами связи. Такая архитектура позволяет решать задачи энергосбережения и синхронизации (при большом размере кристалла из-за разброса параметров невозможно синхронизовать все устройства стандартным способом). Один из главных способов уменьшения потребления питания - оптимизация информационных потоков, так как энергопотребление микросхемы в активном режиме определяется частотой переключения элементов, нагруженных на длинные линии. Распределенная память под каждый вычислительный узел, а не глобальная, как это делается в микропроцессорах с технологическими нормами выше 100 нм, позволяет существенно снизить энергопотребление.

Существенное снижение энергопотребления можно достичь при использовании асинхронного протокола или строго самосинхронного обмена данными (как части асинхронной схемотехники). Это один из возможных путей дальнейшего развития цифровой схемотехники. Однако в настоящее время системы проектирования ведущих мировых компаний позволяют полноценно проектировать только синхронные схемы.

При технологических нормах ниже 45 нм и числом транзисторов больше 100 млн. необходимо также вводить средства, повышающие надежность микросхемы, в противном случае микросхема будет непрерывно сбиваться. Архитектура «сеть на кристалле» с возможностью изменения конфигурации и перераспределения ресурсов - один из значимых механизмов повышения надежности.

Итак, несмотря на экспоненциальный рост числа транзисторов в микропроцессорах, имеются значительные ограничения в росте производительности микропроцессора:

1. значительное повышение потребление питания при повышении частоты микропроцессора свыше 3-4 ГГц и невозможность отвода тепла от него существующими технологиями,

2. существенный разброс технологических параметров транзисторов на кристаллах микросхем с технологическими нормами 45 нм и ниже на расстояниях больших 1-3 см (для 65 нм технологии эти расстояния уже больше 5 мм),

3. значительное снижение надёжности функционирования микропроцессоров при создании кристаллов с числом транзисторов

большим нескольких сот миллионов и использования технологических норм ниже 45 нм.

Одним из наиболее значимых методов повышения производительности современных микропроцессоров является создание многоядерных микропроцессоров, с архитектурами типа *CMP (Chip Multi Processing)* и «сети на кристалле». Широкое распространение получила архитектура *SMT (Simultaneous Multithreading)* – многотредовая или многонитевая архитектура, позволяющая организовать параллельное вычисления для различных процессов. Как правило, современные микропроцессоры совмещают эти 2 архитектуры, то есть это многоядерные многотредовые процессоры. Дальнейшее повышение производительности вычислительных систем обеспечивается за счёт создания многопроцессорных систем вплоть до суперЭВМ.

1.2. Методы снижения энергопотребления цифровых КМОП СБИС

Как было отмечено выше, основной сдерживающий фактор роста производительности микросхем является рост потребляемой мощности. С точки зрения подходов к минимизации энергии, потребляемой при функционировании логических элементов, можно ввести следующую классификацию составляющих частей потребляемой энергии:

- рассеивание энергии из-за токов утечек в статическом режиме (1);
- рассеивание энергии из-за протекания токов в динамическом режиме (2);
- энергия формирования логического состояния элемента (3).

Потребляемую элементом мощность можно рассчитать исходя из следующей формулы:

$$P = P_{in} + P_{st} + P_l, \text{ где} \quad (1.1)$$

$$P_l = \frac{C \times U^2}{2} \times K \times f \quad (1.2)$$

$$P_{in} = E(t_f C) \times K \times f \quad (1.3)$$

P_{in} - внутренняя мощность элемента, определяемая сквозными токами и определяемая как энергия переключения элемента $E(t_f, C)$, которая зависит от фронта сигнала на входе и от нагрузки, умноженная на коэффициент переключений (K) и на частоту (f);

P_{st} - мощность, потребляемая в статическом режиме;

P_l – мощность, затрачиваемая на перезаряд нагрузки, определяемая энергией перезаряда емкости нагрузки, умноженная на коэффициент переключений (K) и на частоту (f);

Правильный выбор коэффициента K во многом определяет правильность расчета потребляемой мощности. Он может быть выбран следующими способами:

- $K = const$ и определяется из статистики;
- $K = var$ и определяется из конкретного теста конкретной микросхемы
- $K = var$ и определяется эвристическими алгоритмами вычисления вероятности переключения, определяемыми САПР [8];

1. Статическое рассеивание определяется суммарными токами утечки закрытых n- и p-канальных транзисторов при отсутствии тактирующих сигналов, когда схема находится в определенном, стабильном состоянии. Для норм выше 65 нм цифровых КМОП схем влияние токов в статическом состоянии можно считать пренебрежимо малым и не оказывающим значительного влияния на общую потребляемую мощность [5]. Для норм 65 нм токи в статическом состоянии становятся уже заметными, а для норм 45 нм и ниже значения токов утечки становятся значительными.

Токи утечки эффективно снижаются созданием специальных технологических процессов *Low Power*, основанных на изменении режимов работы транзисторов и повышении порога его переключения, соответственно используется специальная библиотека логических элементов на основе транзисторов с разными порогами. МОП транзисторы с пороговым напряжением, увеличенным на 120 - 150 мВ, имеют ток утечки в 10 - 20 раз меньше тока транзисторов с номинальным пороговым напряжением 0,25 - 0,3 В. Увеличение порогового напряжения достигается с помощью увеличения толщины подзатворного диэлектрика и изменения профиля легирования кармана.

Однако это приводит к снижению быстродействия и нагрузочной способности транзисторов.

Схемотехнические методы снижения энергии статического потребления направлены на снижение токов утечек, главным образом, за счёт последовательного включения в стоковую цепь элементов дополнительных транзисторов, которые ограничивают величину тока стока. Подобные меры по ограничению тока стоковой цепи способствуют увеличению времени переключения элемента из одного логического состояния в другое. Для решения этой проблемы применяются различные схемы управления дополнительными транзисторами, которые в зависимости от режима работы элементов (частоты изменения его выходных логических состояний, продолжительность нахождения в каком-либо одном логическом состоянии) задают повышенный или пониженный ток стока элемента. Подобные подходы имеют наибольшую эффективность снижения энергии потребления в статическом режиме (в 2-5 раз), если частота переключения логических элементов не превышает нескольких сотен МГц [6];

2. Динамическое рассеивание происходит в моменты переключения в логических элементах схемы. Основная причина наличия динамического тока в КМОП схемах – это перезаряд внутренних емкостей СБИС, величина которых определяется размерами топологических элементов, формируемых в процессе создания транзисторов и межсоединений [5, 7]. Заряд и разряд емкостей происходит через коммутируемые n- и p-канальные транзисторы, в которых происходит рассеяние потребляемой динамической энергии.

На рис 1.1. показана эквивалентная схема заряда и разряда выходной емкости C_L при переключении КМОП элемента. Ключ моделирует циклы заряда и разряда, V – идеальный источник напряжения питания, Rs и Rd – эквивалентные сопротивления открытых n- и p-канальных транзисторов.

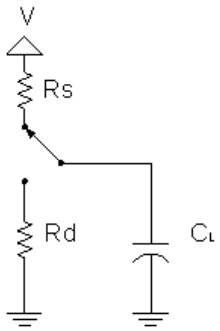


Рис. 1.1. Эквивалентная схема заряда и разряда конденсатора.

Если заряжать и разряжать конденсатор C_L с частотой f циклов в секунду, общая мощность рассеивания системы составит:

$$P_l \sim C_L \times V^2 \times f \quad (1.4)$$

Уравнение 1.4 служит для оценки мощности рассеивания цифровых КМОП схем.

Необходимо отметить, что при этом были сделаны следующие допущения: ёмкость нагрузки C_L и напряжение питания V постоянны, ёмкость успевает полностью зарядиться и разрядиться.

Если частота переключения f и напряжение питания одинаковы для всех этих узлов, общую мощность потребления P можно представить как

$$P_l \sim C_{total} \times V^2 \times f \quad (1.5)$$

где C_{total} – эквивалентная емкость, определяемая суммарной величиной емкостей всех узлов СБИС, заряжаемая и разряжаемая в процессе функционирования микросхемы.

Все существующие методы снижения энергопотребления цифровых схем направлены на уменьшение величины параметров, входящих в формулу (1.5).

По каждому из варьируемых параметров необходим поиск его оптимального значения, поскольку выигрыш по рассеиваемой мощности сопровождается определёнными потерями по другим параметрам микросхем или системным характеристикам.

При уменьшении величины емкостей, например, при увеличении степени интеграции и уменьшении топологических размеров элементов структур, уменьшаются коммутируемые заряды и потребляемая динамическая мощность СБИС. Однако уменьшение величины заряда в ёмкости элемента хранения информации приводит к уменьшению устойчивости данного элемента, так как меньший по величине паразитный заряд вызовет сбой этого элемента. Определение оптимальных для конкретных применений параметров библиотечных элементов, в том числе емкостей элементов памяти, требует проведения исследований условий работы с учётом сбое- и отказоустойчивости СБИС.

Квадратичная зависимость от напряжения питания V определяет высокую эффективность уменьшения V для снижения потребляемой мощности. Это определяет тенденцию уменьшения напряжения питания до значений ниже 2,0-1,5 вольт. Но такое решение возможно только при условии уменьшения величины пороговых напряжений транзисторов, что вызывает снижение помехоустойчивости и, следовательно, ужесточает требования к условиям работы системы.

Величина напряжения питания, необходимая для выполнения операций с минимальной или максимальной скоростью, может отличаться в несколько раз. Это позволяет использовать динамическое управление величиной напряжения питания СБИС или даже её отдельных узлов в разных режимах работы для уменьшения энергопотребления. Для повышения быстродействия требуемых блоков в ряде случаях используют:

- локальное повышение напряжения питания,
- уменьшение пороговых напряжений транзисторов до предельно допустимых значений, ограниченных возрастанием токов утечек, соизмеримых со значением динамических токов.

Иногда повышение быстродействия СБИС с одновременным снижением величины коммутируемых зарядов, а значит, и уменьшением потребления, достигается уменьшением перепада напряжения заряженной и разряженной ёмкости (неполного перезаряда).

Отдельное направление занимает снижение энергопотребления с помощью программных средств [9,10], а также с помощью техноло-

гических приемов, например, за счет применения технологии «кремний-на-изоляторе» [11], которая характеризуется меньшими значениями емкостей элементов топологических структур, изолированных слоем диэлектрика.

Учитывая вышесказанное, можно провести классификацию методов снижения энергопотребления следующим образом:

- Методы, направленные на снижение напряжения питания.
- Методы, направленные на снижение суммарной емкости схемы.
- Методы, направленные на снижение частоты переключения.

Имеются также методы снижения потребляемой энергии в динамическом режиме, рассеиваемой транзисторами в тепло, выделенные в отдельное направление проектирования адиабатических элементов. Разработка адиабатических элементов включает в себя специальную схемотехнику и организацию функционирования, благодаря которым исключается протекание сквозных токов в динамическом режиме и снижается количество рассеиваемой энергии при смене логического состояния. Причём, на пониженных частотах функционирования (десятки-сотни МГц) эффективность снижения рассеиваемой энергии при изменении логического состояния у адиабатических элементов значительно выше, чем у обыкновенных статических КМОП элементов.

3. Энергия логического состояния определяется величиной ёмкости выходного узла, на котором электрический заряд создаёт разность потенциалов между выходом и шиной питания или общей шиной. Смена логического состояния приводит к рассеиванию этой энергии в тепло на транзисторах элемента. Процесс переключения является необратимым и для формирования очередного логического состояния требуется такое же количество энергии. Задача по сохранению энергии логического состояния заключается в организации такой работы логической схемы, при которой информация о входных сигналах не теряется в процессе совершения логических операций и для определения значения входных сигналов не требуется затрачивать дополнительную энергию на формирование новых логи-

ческих уровней. Логический элемент, обладающий такими функциональными свойствами, называется обратимым. В отличие от обратимых элементов, методы понижения динамического энергопотребления, применяемые в адиабатических элементах, направлены на минимизацию рассеивания энергии логического состояния в тепло, но не сохранении логического состояния. Это означает, что основная проблема обратимых вычислений заключатся в исследовании причин потери логических состояний при совершении логических операций (смена логического состояния с потерей информацией об этом логическом состоянии), а основная проблема адиабатических вычислений заключается в минимизации последствий смены логического состояния (рассеивание энергии логического состояния в тепло).

1.2.1. Снижение напряжения питания и уменьшение проектных норм

В соответствии с формулой (1.2) мощность потребления имеет квадратичную зависимость от напряжения питания, поэтому наибольшую популярность получили методы, ориентированные на снижение напряжения. Однако простое снижение приводит к уменьшению производительности схемы. Так, по данным [12] снижение напряжения питания на 10 % приводит к снижению быстродействия в среднем на 10 %. Поэтому, для высокопроизводительных схем всегда необходимо использование комплекса мер по сохранению быстродействия. Самым распространенным методом снижения напряжения при одновременном сохранении и даже повышении быстродействия является переход на улучшенные топологические проектные нормы. Применение новой технологии позволяет не только компенсировать снижение быстродействия, связанное с уменьшением напряжения, но и приводит к существенному повышению производительности СБИС. При переходе на новый уровень технологии уменьшается еще один параметр формулы (1.2) – эквивалентная емкость нагрузки. Это происходит как за счет того, что базовые элементы имеет меньшие размеры и меньшую емкость, так и за счет

того, что общий размер кристалла становиться меньше, при сохранении выполняемых функций СБИС, и соответственно, уменьшается длина и емкость межсоединений.

Тем не менее, данный метод имеет следующие недостатки:

1. Стоимость изготовления микросхем значительно увеличивается (примерно в 2 раза при уменьшении технологических норм в 1.4 раза). Особенно это заметно для малых партий, когда изготавливается несколько пластин и стоимость накладных расходов (стоимость масок, подготовка к производству и пр.) получается очень значительной.
2. Изменение напряжения питания требует соответствующего изменения схемотехники вычислительных систем, в которых установлена исходная микросхема.
3. Если оптимизируется микропроцессор, то во многих случаях требуется переделка контроллеров, дополняющих этот микропроцессор в вычислительных системах.
4. При переходе на более низкие нормы уменьшается сбоеустойчивость микросхем. Тиристорный эффект для КМОП СБИС начинает сильно сказываться для авиационной и космической техники уже с норм 0,18 мкм и приводит к недопустимому снижению сбоеустойчивости начиная с норм 0,13 мкм при отсутствии специальных схемотехнических и технологических решений.
5. Статический ток заметно возрастает, начиная с норм 65 нм.

1.2.2. Динамическое управление частотой и напряжением питания

Динамическое управление частотой и напряжением питания для снижения энергопотребления широко используется при проектировании вычислительных систем на базе микропроцессоров [13]. Метод заключается в том, что специальная программа анализирует текущую загрузку микропроцессора и, если текущая загрузка не требует максимальной производительности, снижает напряжение питания и рабочую частоту. Все наиболее известные производители микропроцессоров для персональных компьютеров используют данный метод в своих системах. Для микропроцессоров компании *Intel* данный метод известен как *SpeedStep* [14], для компании *AMD* – как

AMD PowerNow!™ Technology [15], для *Transmeta* – как *Transmeta LongRun Power Management* [16]. Эффект, достигаемый применением данного метода, зависит от выполняемой задачи.

Преимущества метода заключается в том, что снижение энергопотребления происходит без потери реальной производительность для пользователя. Данный метод имеет следующие недостатки:

1. Конструкция микропроцессора должна обеспечить стабильную работу в условиях переменного напряжения питания и тактовой частоты.
2. Требуются специальные аппаратные ресурсы на уровне вычислительной системы, такие как программно-управляемый источник напряжения. Контроллеры, входящие в состав микропроцессорной системы должны обеспечивать работу с изменяемой временной диаграммой со стороны микропроцессора и, при этом, обеспечивать требуемые временные интервалы для внешних устройств: таймеры, контроллеры последовательных портов и другие.
3. Требуется специальное программное обеспечение, которое должно принимать решение о необходимой достаточности текущих значений напряжения и частоты и, в случае необходимости, управлять их изменением.
4. Сложно прогнозировать время выполнения программ, что ограничивает применение метода в системах реального времени.

1.2.3. Включение дополнительных тактов останова в конвейер

Оригинальный способ контроля за энергопотреблением применен в микропроцессорах *G*3 и *G*4 семейства *PowerPC* [17,18]. Данные микропроцессоры содержат блок термического сопровождения (*Thermal Assist Unit*) в функцию которого входит контроль за температурой кристалла. В случае превышения допустимой нормы блок дает специальное прерывание в устройство управления, реакцией на которое является формирование тактов останова, что приводит к снижение частоты доступа к кэш-памяти команд. При этом, микропроцессор потребляет меньше энергии и, соответственно, начинает охлаждаться. Метод относиться к группе методов, направленных на

снижение частоты переключения, и имеет следующие преимущества по сравнению с понижением частоты или снижением напряжения питания:

1. Поскольку рабочая частота не меняется, возможно использование традиционной схемы автоподстройки частоты.

2. Включение тактов останова происходит без участия программного обеспечения.

3. Не требуется дополнительных аппаратных ресурсов на уровне вычислительной системы.

Данный метод имеет следующие недостатки:

1. Требуется включение в конструкцию микропроцессора дополнительных элементов, в том числе аналоговых.

2. Снижение энергопотребления происходит за счет снижения быстродействия.

3. Сложно прогнозировать время выполнения программ, что ограничивает применение метода в системах реального времени.

1.2.4. Использование нескольких напряжений в одном кристалле

Применение нескольких источников напряжения в одном кристалле, описанное в [19], основано на том, что для любой схемы имеются блоки, которые не являются критичными с точки зрения быстродействия. Соответственно, требования к быстродействию таких блоков ниже, чем для блоков, определяющих производительность СБИС, и для них возможно снижение напряжения питания. В [20] приведен пример СБИС в которой реализован данный метод. Наиболее критичные блоки памяти команд и регистровый файл используют напряжение 2.5 В, тогда как остальные блоки питаются от пониженного источника 1.75 В.

Для данной СБИС применение раздельных источников позволило снизить энергопотребление на 30 % без снижения производительности.

Метод имеет следующие недостатки:

1. Значительно повышается трудоемкость разработки топологии микросхемы за счет того, что требуется реализация двух или более шин питания.

2. Необходима установка дополнительных преобразователей уровней между блоками с разным напряжением питания, что вносит дополнительную задержку.
3. Предъявляются дополнительные требования к конструкции печатной платы и к источникам питания.
4. В процессе проектирования требуется использование нескольких библиотек стандартных ячеек.
5. Необходимо наличие специального технологического процесса.

1.2.5. Применение технологии «кремний на изоляторе»

Технология «кремний на изоляторе» (КНИ) обладает рядом преимуществ по сравнению с традиционной технологией объемного кремния и получила широкое распространение в конце 90-х годов. В частности, компании *IBM* и *AMD* выбрали эту технологию как основную для линии своих микропроцессоров. Одним из преимуществ данной технологии является пониженное энергопотребление. На рис. 1.2 показана конструкция транзистора, выполненного по технологии КНИ.

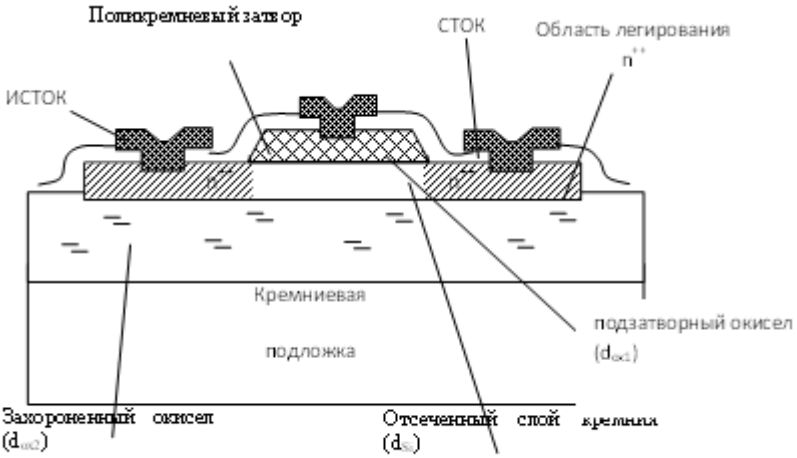


Рис. 1.2. Пример транзистора на структуре КНИ.

Площадь изолирующего *p-n* перехода для КНИ транзистора значительно меньше, чем для традиционного транзистора. Соответственно, меньшую емкость требуется заряжать при каждом переключении.

Из литературы следует, что применение КНИ дает до 50 % снижения энергопотребления [21]. Микросхемы, созданные в России по технологии КНИ, потребляют ниже аналогичных микросхем, изготовленных по технологии КМОП, на 20-25 % при увеличении быстродействия до 40 % [22].

Данный метод имеет следующие недостатки:

Высокая стоимость исходного материала, КНИ пластина стоит на порядок больше пластины на объемном кремнии.

Отсутствие широкодоступных библиотек стандартных ячеек и IP-блоков для проектирования цифровых схем для КНИ.

Сложность с размещением заказа на изготовление на зарубежной фабрике.

Следует также отметить, что существует 2 направления развития КНИ технологии: создание радиационно-стойких микросхем и создание быстродействующих и низкопотребляющих микросхем. Для радиационно-стойкой технологии необходимо обеспечить устойчивую работу микросхемы при прохождении высокоэнергетической заряженной частицы (ВЧ) (см. рис. 1.3а) и образовании большого количества электрон/дырочных пар (см. рис. 1.3б).

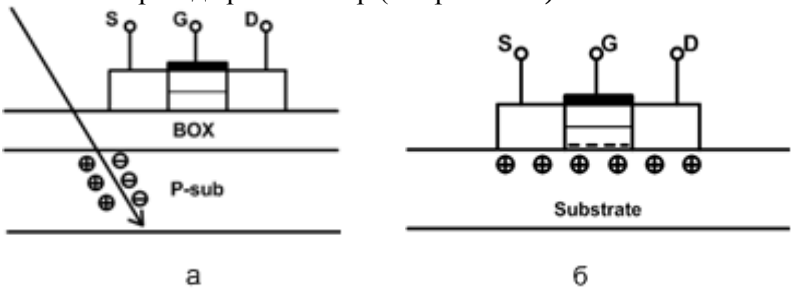


Рис. 1.3. Образование электрон/дырочных пар КНИ СБИС (а) и положительного заряда в подложке КНС СБИС (б) под воздействием ВЧ.

Под воздействием положительного заряда на «тело» транзистора снижается пороговое напряжение. Эффект снижения порогового

напряжения *N*-канального транзистора наблюдается и при воздействии ионизирующего излучения (ИИ) на КНИ СБИС. Это вызывает снижение напряжения пробоя сток-истока (*Vds*) *N*-канального транзистора, что в свою очередь приводит к росту статического тока потребления КНИ СБИС и к функциональным сбоям.

Для снижения этого эффекта, в частности, устанавливается фиксированный потенциал подложки (*substrate*) КНИ СБИС, что не делается в случае создания технологии высокоскоростных схем. Улучшение стойкости к ИИ достигается за счет минимизации накопления заряда в захороненном окисле под действием отрицательного потенциала подложки.

1.2.6. Метод селективного отключения синхросигнала

Современные высокопроизводительные микросхемы являются, как правило, синхронными схемами, то есть изменение состояний происходит по тактовому сигналу (синхроимпульсу). Для использования внешних асинхронных сигналов в таких схемах необходимо их сначала синхронизовать (удовлетворить требованиям предустановки и сохранения сигнала относительно фронта синхроимпульса, по которому происходит изменение состояний). На рис. 1.4 приведён пример синхронизации сигнала *DATA_IN* двумя *D*-триггерами.

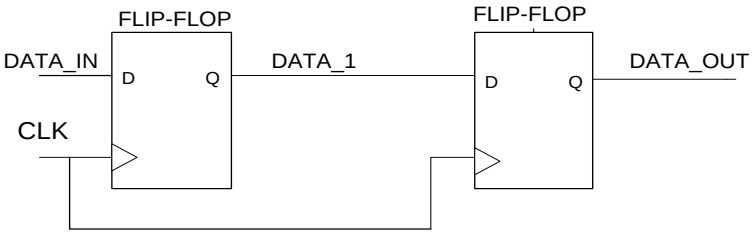


Рис. 1.4. Синхронизация асинхронных сигналов в синхронных системах.

На первый взгляд может показаться, что достаточно только одного *D*-триггера, однако сигнал *DATA_OUT* в этом случае может не просто принять неопределённое состояние «0» или «1», а стать последовательностью «глитчей» (коротких импульсов длительностью, сопоставимых величине задержки внутренних элементов) в течение первого полупериода синхроимпульса. Это объясняется тем, что триггер в этом случае может заработать как генератор. Чаще всего на выходе появляется просто короткий импульс. Характер выходного сигнала зависит от реализации и параметров *D*-триггера и времени появления сигнала *DATA_IN* относительно синхроимпульса. Сгенерированные «глитчи» могут проходить все логические схемы, поскольку их длительность мала, и в конечном итоге вызвать сбой работы схемы (возможно также затухание). Одиночный короткий импульс также может вызвать сбой схемы по самым различным причинам. Такие сбои обычно являются редкими событиями, и поиск причины сбоя в готовой микросхеме может занять длительное время. Поэтому наличие двух триггеров или других подобных решений является обязательным. Для минимизации затрат сигнал *DATA*_1 после выхода первого триггера можно замешать с другими синхронными сигналами с целью выполнения требуемой логической функции.

Однако при каждом переключении синхронных схем по тактовому сигналу возникает соответствующее возрастание потребления питания. Исключение тактового сигнала для схем, которые не выполняют полезной функции в данный момент времени, приведёт к аналогичному снижению потребления питания.

Метод селективного отключения синхросигнала (*clock gating*) заключается в том, чтобы отключать синхросигнал на более ранней ступени дерева синхросигналов для блоков, которые в данный момент не выполняют полезную функцию. При этом перестают переключаться и рассеивать энергию все узлы отключаемого блока, включая узлы, формирующие сами синхросигналы, расположенные по дереву ниже вентиля отключения. Это дает существенную экономию мощности, поскольку в синхронных схемах дерево синхросигналов работает с наибольшей частотой и имеет большую суммарную емкость. Для микропроцессоров дерево синхросигналов может по-

треблять свыше трети общей мощности микросхемы [23]. В настоящий момент метод отключения синхросигналов широко используется при проектировании микропроцессоров и достаточно подробно описан в литературе [24]. Имеются средства САПР, позволяющие автоматически встраивать схемы *clock gating* в исходную схему, однако, как правило, это не позволяет добиться максимальной эффективности, то есть не все схемы, которые могли бы отключаться в данный момент времени отключаются в синтезированной схеме. Для максимального использования данного метода разработчики кода *RTL* (исходный код для синтеза микросхемы) вставляют конструкции, позволяющие при синтезе однозначно вставлять схемы, отключающие синхросигналы.

Вариант перехода от традиционной синхронной схемы к схеме с использованием метода отключения синхросигнала приведён на рис. 1.5. и рис. 1.6.

На схеме, показанной на рис. 1.4, одно и то же значение входной информации перезаписывается в регистр каждый такт, когда управляющий сигнал не активен (*EN*=0). Заменив мультиплексор схемой, позволяющей отключать поступающий на регистр синхросигнал при *EN*=0, можно значительно минимизировать энергопотребление данной схемы.

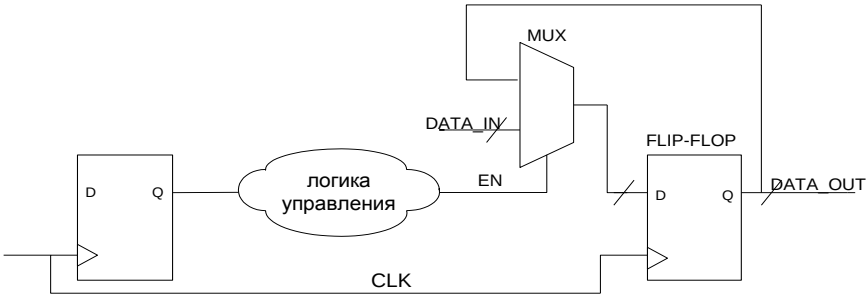


Рис. 1.5. Традиционный способ построения синхронной схемы.

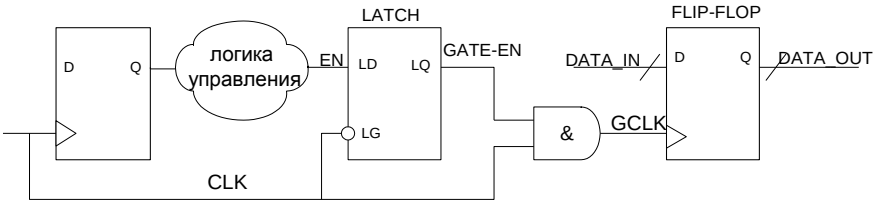


Рис. 1.6. Построение синхронной схемы с отключаемым синхросигналом.

На схеме, показанной на рис. 1.6, сигнал *EN* управляет подачей синхросигнала на элемент типа регистр. В каждом такте, когда уровень сигнала *EN* низкий, синхросигнал *GCLK* не подается на регистр. Элемент «регистр-защелка» предотвращает возможные выбросы (*glitch*) по сигналу управления *GATE-EN* и обеспечивает стабильный gate-сигнал (*GCLK*) во время положительного импульса *CLK*.

Временная диаграмма, характеризующая работу данной схемы, приведена на рис. 1.7.

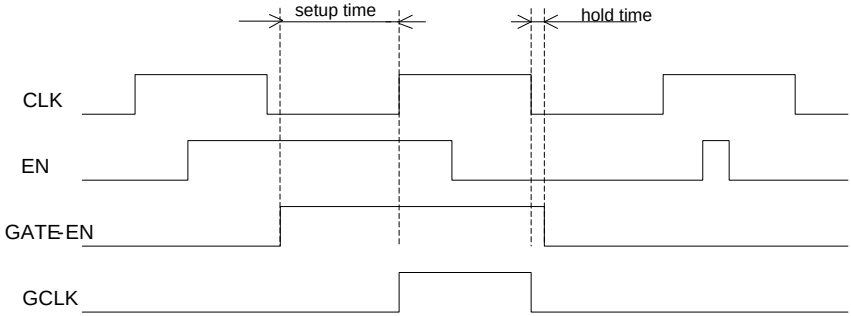


Рис. 1.7. Временная диаграмма работы схемы с отключаемым синхросигналом.

Следует отметить, что элементы, используемые для построения дерева синхросигналов, обладают рядом характеристик, которые отсутствуют у остальных элементов библиотеки. Например, в то время

как обычные элементы имеют разное время переключения из состояния логической единицы в ноль и, наоборот, для элементов, формирующих дерево синхросигналов это недопустимо. Поэтому необходимо обеспечить равенство времени задержек для переднего и заднего фронта во всем диапазоне изменений напряжения питания, температуры и параметров технологического процесса. Особое внимание требуется уделять введению дополнительных буферных элементов в цепь управляющего сигнал (*GATE_EN*), поступающего на объединяющий вентиль, для обеспечения требуемой скважности синхросигнала. Разработчик должен гарантировать, что включение дополнительной управляющей логики в дерево синхросигналов не внесет искажений в длительность фронтов синхросигналов и в их временные характеристики [25].

Таким образом, несмотря на простоту принципа использования механизма отключения синхросигнала, реализация этого метода на практике сопряжена с рядом сложностей. Сама схема управления синхросигналом занимает площадь кристалла, в некоторых случаях значительную и увеличивает энергопотребление. Это означает, что в каждом конкретном случае следует решать вопрос о поиске наиболее оптимального варианта реализации данного механизма.

1.2.7. Использование средств САПР

В связи с тем, что требование снижения энергопотребления современных микропроцессорных систем стало жизненно необходимым, в последние годы появилось достаточное количество САПР, позволяющих автоматизировать процесс построения схем со сниженным энергопотреблением. Наиболее распространенными являются продукты *Power Compiler* (в составе *Design Compiler*) компании *Synopsys* и *RTL Compiler* компании *Cadence*. Оба продукта, в качестве исходных данных требуют библиотеку элементов, содержащую данные о мощности потребления каждого элемента в зависимости от емкости нагрузки и набор временных ограничений, налагаемых на проектируемую схему. Для анализа энергопотребления требуется также информация о входных воздействиях.

Для снижения энергопотребления используются следующие методы:

• анализ схемы на предмет выявления мест, позволяющих применить метод отключения неиспользуемых блоков. На основе анализа автоматически модифицируется схема посредством включения элементов отключения синхросигнала, если это не нарушает временных ограничений;

• оптимизация размещения логических элементов на кристалле с целью уменьшения длин и, соответственно, емкости межсоединений;

• оптимизация схемы на уровне библиотечных ячеек, а именно:

○ уменьшение нагрузочной способности элементов;

○ перестановка логически эквивалентных, но электрически разных входов библиотечных ячеек;

○ ограничение длительности фронтов сигналов посредством добавления буферных элементов;

○ замена нескольких библиотечных ячеек одной, более сложной, логически эквивалентной ячейкой.

Все вышеперечисленные действия могут быть реализованы только, если соблюдаются заданные временные ограничения.

Использование средств САПР для снижения энергопотребления может быть рекомендовано к включению в маршрут проектирования цифровых схем. Основным ограничением может быть высокая стоимость САПР, а также отсутствие соответствующих библиотек стандартных ячеек, удовлетворяющих требованиям САПР.

Однако возможности САПР не могут заменить возможности человека в тех случаях, когда для снижения показателя энергопотребление/производительность требуется изменение алгоритма функционирования микросхемы.

1.2.8. Принцип понижения энергопотребления адиабатических логических элементов.

Переключение статического КМОП цифрового элемента из одного логического состояния в другое сопровождается рассеиванием некоторого количества энергии, которая складывается из энергии, потраченной на протекание сквозных токов через транзисторы при переключении, и энергии, необходимой для изменения потенциала

выходного узла схемы. Величина первой составляющей рассеиваемой энергии зависит от времени нарастания или спада входного сигнала, напряжения питания, порогового напряжения КМОП транзисторов. Величина второй составляющей рассеиваемой энергии рассчитывается по формуле:

$$E_{diss} = I^2RT = \left(\frac{CV_{dd}}{T}\right)^2 RT = \left(\frac{RC}{T}\right) CV_{dd}^2$$

где I – ток, которым осуществляется перезаряд ёмкости выходного узла элемента, R – сопротивление элемента, через который проходит ток перезаряда выходной ёмкости, C – величина эквивалентной ёмкости выходного узла, V_{dd} – напряжение питания (величина разности уровней двух различных логических состояний), T – время, в течение которого происходит перезаряд выходной ёмкости (изменение выходного логического состояния). Эта формула справедлива только для случаев $T >> RC$.

Задача по снижению динамического энергопотребления, с точки зрения принципа адиабатического вычисления, заключается в поиске схемотехнических решений и организации функционирования элемента, при которых, по отношению к режимам работы транзисторов в элементе, должны выполняться два правила:

1) транзистор не может переходить из открытого состояния в закрытое, если через него протекает ток;

2) транзистор не может переходить из закрытого состояния в открытое, если между стоком и истоком есть разность потенциалов;

Формулировка этих двух условий накладывает ограничения на режим работы транзисторов, при которых ток, проходящий через канал транзистора, вызывает падение напряжения между стоком и истоком, приводя к рассеиванию части энергии носителей заряда в тепло. В обыкновенных КМОП элементах применяются шины питания с фиксированным потенциалом и при изменении логического состояния элемента на проводящих транзисторах, формирующих выходной уровень, неизбежно падает часть напряжения и рассеивается энергия. Характерной особенностью адиабатического вычисления является использование на общих шинах элементов импульсов напряжения с линейной зависимостью нарастания и спада от времени. Переключение элемента из одного логического состояния в другое осуществляется в период изменения напряжения на общей

шине за счёт коммутации выходного узла с общей шиной. Ток перезаряда в период переключения, протекающий через транзисторы, задаётся разностью потенциалов между общей шиной и выходным узлом. Если время нарастания Т линейно изменяющегося напряжения на общей шине значительно превосходит постоянную времени $\tau \approx RC$ данного элемента, то разность потенциалов между выходным узлом и общей шиной в период переключения будет незначительной.

Транзисторы (или диоды), коммутирующие общую шину с выходным узлом в период изменения напряжения на общей шине, пропускают через себя ток, который перезаряжает эквивалентную ёмкость выходного узла. Элемент переключается в другое логическое состояние, если ток стока проводящих транзисторов I переносит заряд Q, необходимый для изменения выходного напряжения на величину напряжения питания. Связь тока I и времени изменения напряжения T на общей шине описывается выражением:

$I=Q/T \sim Qf$, где f – частота изменения напряжения на общей шине.

Таким образом, ток стока транзисторов, формирующих выходной сигнал, пропорционален частоте импульсов напряжения на общей шине. При повышении быстродействия схемы за счёт роста частоты смены импульсов, уменьшается время перезаряда выходного узла. Сокращение времени перезаряда вызывает рост тока стока транзисторов, которые осуществляют перезаряд. Повышение тока стока способствует падению напряжения между стоком и истоком и, следовательно, повышению рассеиваемой энергии.

1.2.9. Использование обратимой логики

Для формирования логического состояния требуется потратить некоторое количество энергии. Установлено, что минимальное количество энергии, необходимое для формирования логического уровня имеет порядок энергии теплового шума [26]:

$E = K \times T \times ln2$,

K - константа Больцмана, T - абсолютная температура вычислительного устройства.

Именно это минимальное количество энергии рассеивается при потере одного логического состояния. Прогнозируется, что традиционная статическая КМОП схемотехника приблизится к этому

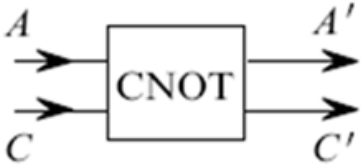
фундаментальному пределу к 2020 году. Преодоление этого фундаментального предела осуществимо, если при совершении любой логической операции не происходит потеря логических значений входных сигналов и возможно их восстановление на основе выходных. То есть для снижения рассеивания энергии при вычислениях необходимо избегать потери логических состояний сигналов.

Обратимая архитектура включает в себя совокупность методов по снижению потери информации, которые могут применяться на разных уровнях организации вычислений. Например, в программировании некоторые операции являются логически обратимыми (*R1 = R1 + R2*), а некоторые являются логически необратимыми (*R1* = 0). Логически необратимая операция приводит к потере бита информации в регистре и, соответственно, к дополнительному рассеиванию энергии. Эта проблема решается оптимизацией алгоритма работы компилятора, который по возможности старается избегать в машинном коде таких операций.

Необратимым является процесс рассеивания энергии при потере битов информации в ходе совершения логических операции или записи в элементы памяти. Можно преодолеть этот предел, если в вычислительном устройстве использовать обратимые логические элементы (т.е. логические элементы, в которых происходящие процессы должны быть обратимыми). Эти элементы могут реализовать обратимую булеву функцию и сохранять полное количество битов информации, поэтому энтропия при работе таких элементов остаётся неизменной. При реализации прямой булевой функции, на выходе обратимого элемента кроме информационных битов формируются так называемые «мусорные» (*garbage*) биты, с помощью которых можно реализовать обратную булеву функцию и восстановить значения входных сигналов.

Примеры обратимых логических элементов. Особенностью всех обратимых логических элементов является возможность однозначно определить состояние входных сигналов по состоянию выходных. Простейшим примером обратимого логического элемента является вентиль НЕ. По состоянию выхода этого элемента можно однозначно определить состояние входа. Вентиль НЕ реализует обратимую логическую функцию, и информация при его работе не теряется.

Другой пример обратимого логического элемента это УПРАВЛЯЕМОЕ НЕ (см. рис. 1.8) [27].



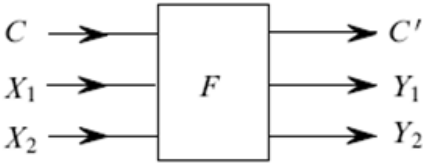
a	c	a'	c'
0	0	0	0
1	0	1	0
0	1	1	1
1	1	0	1

Рис. 1.8. Схематическое изображение вентиля, УПРАВЛЯЕМОЕ НЕ и его таблица истинности.

Состояние выхода *C'* всегда совпадает с входом *C*, а состояние выхода A' инвертируется при C = 1. Как видно из таблицы истинности, этот элемент является обратимым, поскольку каждому выходному состоянию соответствует определённое состояние входов. Логическая функция этого элемента может быть записана следующим образом:$a' = a \times \bar{c} + \bar{a} \times c$. Стоит отметить, что под свойством консервативности элемента понимается сохранение количества информации при её прохождении через обратимый логический элемент или цепочки из таких вентилей. Под свойством обратимости элемента также понимается возможность замены его входов на выходы вместе с заменой его логической функции на обратную. При этом количество входов у обратимого элемента должно совпадать с количеством выходов.

Для построения вычислительной системы необходимо, чтобы элемент мог выполнять любую логическую функцию. При этом, логический элемент должен реализовывать по крайней мере функции И и ИЛИ, или И и НЕ, или НЕ и ИЛИ и допускать каскадирование. Из одних только элементов УПРАВЛЯЕМОЕ НЕ и НЕ нельзя создать вычислительно-универсальный вентиль, на основе которого можно построить любую булеву функцию. Вычислительно-универ-

сальным обратимым элементом является вентиль Фредкина [28], который можно понимать как вентиль УПРАВЛЯЕМЫЙ ОБМЕН, реализующий следующую функцию:$c' = c$; $y_1 = c \times x_1 + \bar{c} \times x_2$; $y_2 = \bar{c} \times x_1 + c \times x_2$, (см. рис. 1.9).



c	x_1	x_2	c'	y_1	y_2
0	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	1	0	1	1	0
1	0	1	1	0	1
1	1	1	1	1	1

Рис. 1.9. Схематическое изображение вентиля Фредкина и его таблица истинности.

На рис. 1.10 показаны различные варианты включения вентиля Фредкина, показывающие его универсальность.

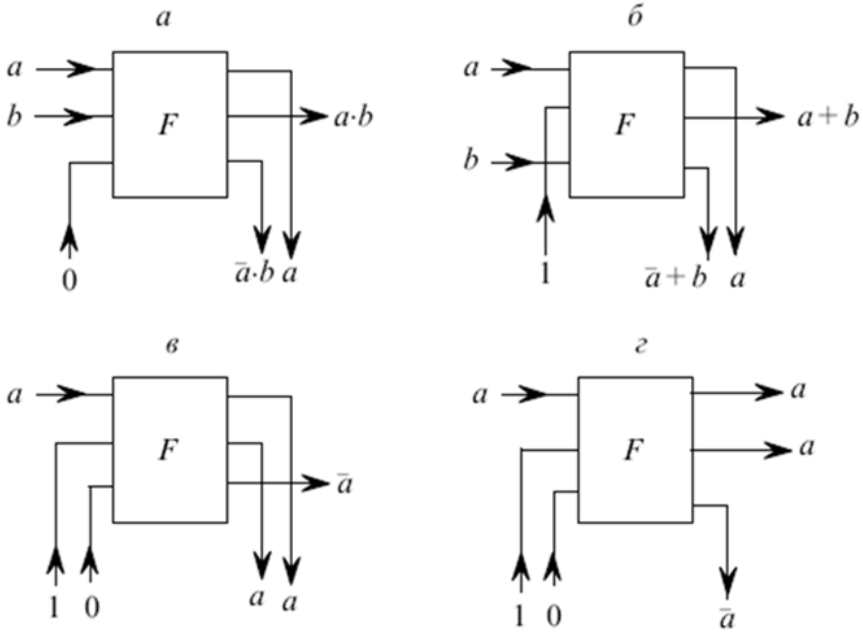
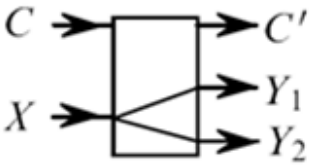


Рис. 1.10. Различные конфигурации обратимых логических вентилей на основе вентиля Фредкина: И (а), ИЛИ (б), НЕ (в) и ВЕТВЛЕНИЕ (г).

С точки зрения оптической реализации можно выделить два логических обратимых и вычислительно-универсальных логических элемента: переключатель Прайса [29] и вентиль ВЗАИМОДЕЙСТВИЕ (*Interaction Gate*) [30]. Переключатель Прайса (см. рис. 1.11) это управляемый переключатель, представляющий логическую функцию: $c' = c;\ y_1 = c \times x;\ y_2 = \bar{c} \times x.$

Вход *C* играет роль адресного.

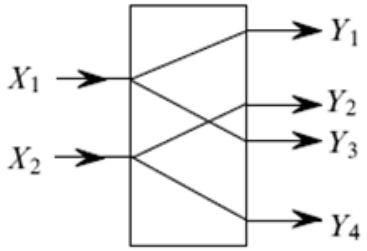


c	x	c'	y_1	y_2
0	0	0	0	0
0	1	0	0	1
1	0	1	0	0
1	1	1	1	0

Рис. 1.11. Схематическое изображение и таблица истинности переключателя Прайса.

Стоит обратить внимание, что конфигурация включения вентиля Фредкина, показанная на рис. 1.8,а, реализует тот же самый переключатель, если переменную *a* понимать как адресный сигнал, а переменную *b* как переключаемый сигнал. Это дополнительный пример универсальности вентиля Фредкина. Вентиль ВЗАИМОДЕЙСТВИЕ (см. рис. 1.12) представляет собой четырёхкомпонентную булеву функцию двух аргументов:

$y_1 = x_1 \times x_2;\ y_2 = \bar{x}_1 \times x_2; y_3 = x_1 \times \bar{x}_2; y_4 = \bar{x}_1 \times \bar{x}_2;.$



x_1	x_2	y_1	y_2	y_3	y_4
0	0	0	0	0	0
1	0	0	0	1	0
0	1	0	1	0	0
1	1	1	0	0	1

Рис. 1.12. Схематическое изображение и таблица истинности вентиля ВЗАИМОДЕЙСТВИЕ.

Для вентиля взаимодействие была предложена механическая реализация – модель бильярдных шаров. Суть данной модели можно

понять из рис. 1.13. Консервативность этой логики обусловлена обратимостью механики Ньютона, описывающей движение и столкновение абсолютно упругих шаров. Бильярдная реализация переключателя Прайса и вентиля ВЗАИМОДЕЙСТВИЕ показана на рис. 1.14.

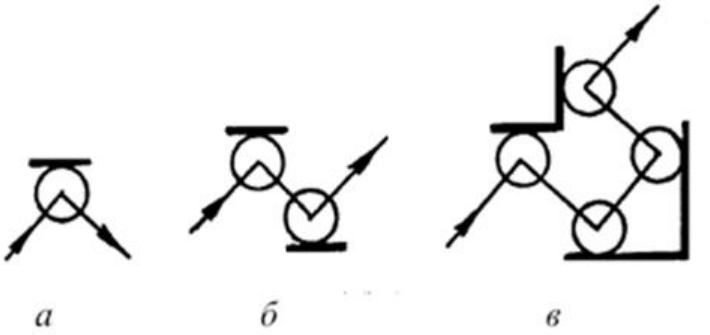


Рис. 1.13. Бильярдная логика: отклонение пути шара (а), смещение пути шара (б) и задержка шара (в).

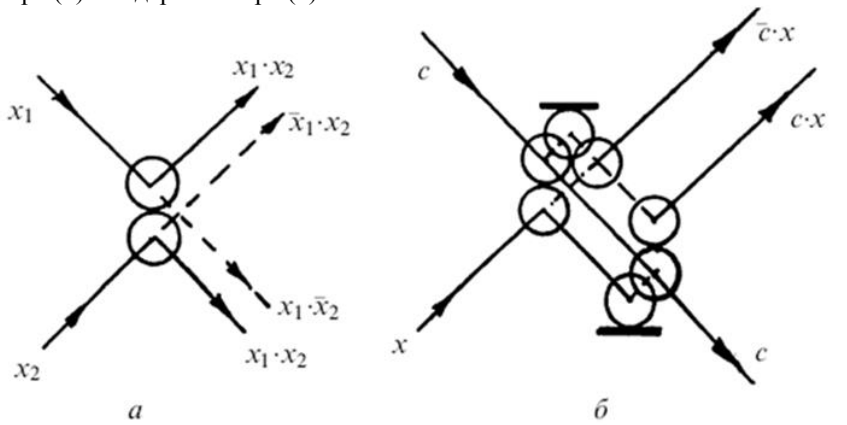


Рис. 1.14. Бильярдная реализация вентиля ВЗАИМОДЕЙСТВИЕ (а) и переключателя Прайса (б).

Предположение, что шары взаимодействуют (сталкиваются) упруго друг с другом и с отражающей стенкой и все столкновения

являются центральными, является жёстким ограничением, не существенным для теоретического анализа, но выступающее веским препятствием при реальном воплощении.

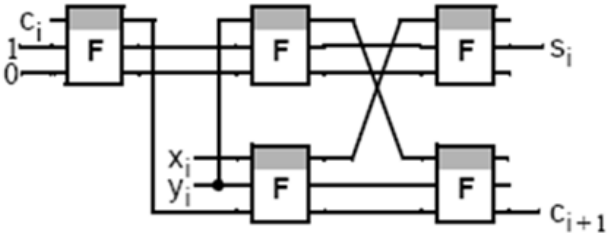
В зависимости от технологического базиса, возможен обоснованный выбор той или иной схемы обратимого вентиля. Перечень схем обратимых элементов не ограничивается элементами, представленными выше. Были предложены и другие схемы обратимых элементов: вентиль Тоффоли [31], вентиль Кернтопфа [32], вентиль Переса [33], вентиль Магроласа [34], и несколько вентилей Де Воса [35].

Схемы цифровых устройств на обратимых элементах.

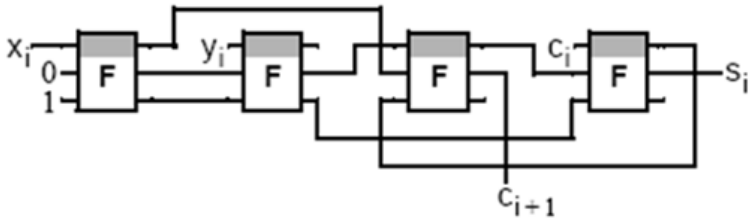
Классические методы синтеза логики не могут быть напрямую применены в схемах, построенных на обратимых логических элементах. В статье [36], говорится, что схемы на обратимых логических элементах проектируются для каждого конкретного устройства с заданной булевой функцией. При этом не существует единых правил, с помощью которых можно было бы спроектировать устройство с произвольной булевой функцией.

Триггеры. В работе [37] была предложена методика построения основных последовательных элементов на обратимых вентилях Фредкина. Согласно описанию, триггеры, спроектированные по предложенной методике, имеют лучшие показатели, чем триггеры, предложенные в работе [38]. Триггеры сравнивались по количеству «мусорных» выходов и по количеству используемых вентилей. При этом были приведены схемы триггеров без привязки к какой-либо элементной базе.

Сумматоры. В статье [39] предложена схема сумматора на обратимых вентилях Фредкина. В этой статье рассмотрены различные реализации схем сумматоров и обоснована схема на рис. 1.15,б и рис. 1.16,б по критерию минимального пути прохождения сигналов.

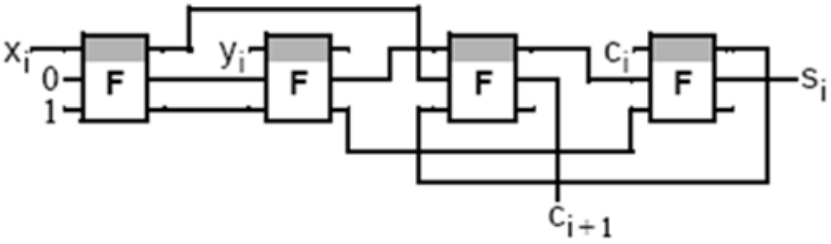


а

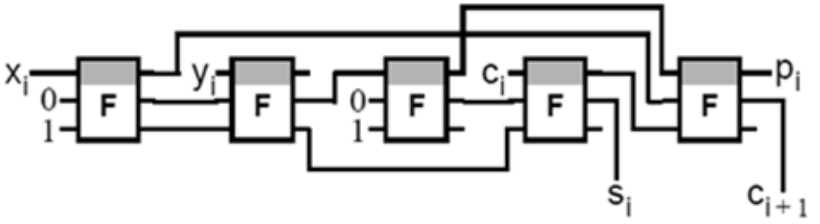


б

Рис.1.15. Возможные варианты реализации полного сумматора на обратимых вентилях Фредкина без возможности каскадирования.



а



б

Рис.1.16. Возможные варианты реализации полного сумматора на обратимых вентилях Фредкина с возможностью каскадирования.

Матрицы программируемых логических вентилей на обратимых элементах. В статье [40] описывается программируемое логическое устройство, построенное на обратимых элементах Фредкина и обратимых элементах ВЗАИМОДЕЙСТАИЕ (вентиль Фейнмана): *Reversible Programmable Logic Array (RPLA)* (см. рис. 1.17).

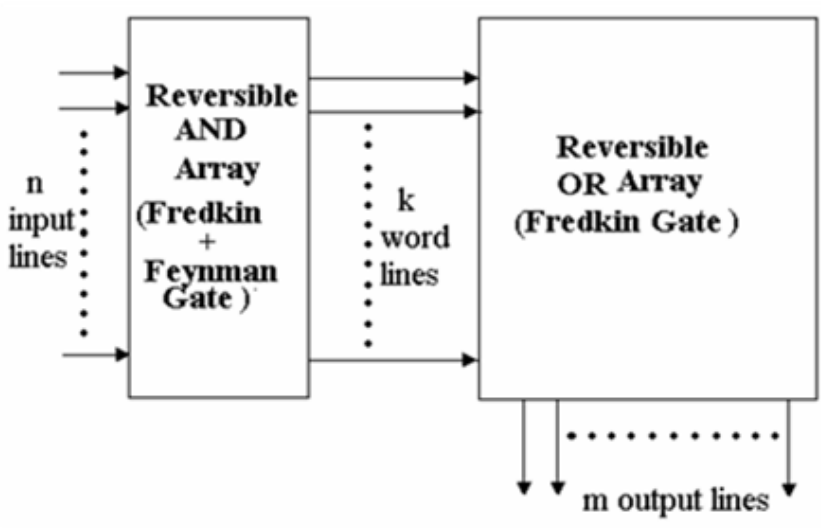


Рис. 1.17. Архитектура *RPLA.*

Технологические базисы для устройств на обратимых логических элементах.

Получены также результаты исследований по реализации обратимых элементов на цилиндрических магнитных доменах [41]. В статьях, опубликованных в период с 1995-2002 года, встречаются описания по реализации обратимых вентилей в следующих технологических базисах:

- КМОП технология с пониженным потреблением энергии [42] (см. рис. 1.18);
- ПЗС структуры [43];
- Оптические вычислительные устройства (оптические солитоны) [29];
- Квантовые вычислительные устройства [44];
- Нанотехнологии [45];

Одной из наиболее перспективных считается технология квантовых вычислений. Кроме возможности реализации обратимых элементов, производительность квантовых вычислительных устройств имеет экспоненциальную зависимость от размерности кубитов. Дан-

ный технологический базис является наиболее подходящим для организации параллельных вычислений, однако в силу принципа неопределённости, точность вычислений с увеличением размерности кубитов падает.

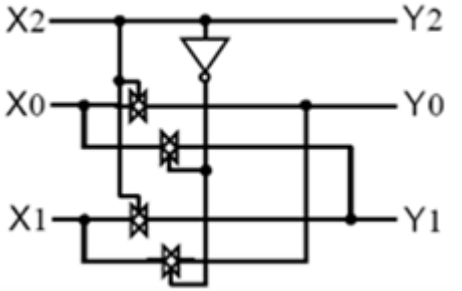


Рис. 1.18. Вентиль Фредкина на КМОП элементах.

Примером обратимого элемента комбинационной логики является элемент Фредкина. На выходе этого элемента формируется выходной сигнал и так называемые «мусорные биты», на основе которых можно восстановить входные значения сигналов. Наиболее наглядной моделью работы обратимого элемента Фредкина является биллиардная модель. Присутствие шара интерпретируется как логическая “1”, отсутствие – как логический “0”. Траектория полёта шара полностью обратима. Совершение логических операций интерпретируется как изменение траектории движения шаров при их соударении между собой. Шары никогда не «слипаются» и не покидают пределы вычислительного устройства. Количество шаров в устройстве остаётся неизменным. Рассеивание энергии шаров при движении может быть минимизировано, если скорость движения шаров относительна невелика. Главная проблема реализации такой модели заключается в синхронном движении шаров. Ведутся работы по реализации модели вентиля Фредкина с использованием особенностей квантово-механических процессов. Например, синхронность поведения шаров может быть реализована на основе дискретных значений спина.

Возможна реализация элементов обратимой логики на ПЗС структурах. Вентиль Фредкина строится по схеме «исток – цель»

(прибор с обратимым переносом заряда *Reversible Charge Transfer*). Между «истоком» (*source*) и «целью» (*destination*) заряд переносится через полупроводниковую область. Отдельное формирование заряда обозначается как пакет. Для минимизации потери энергии в устройствах RCT при переносе заряда необходимо соблюдение трёх правил:

1) никогда не объединять и не делить пакеты зарядов;
2) никогда не переводить пакет зарядов на шину земли;
3) переводить пакет от истока к цели и наоборот только в термодинамически-обратимую фазу работы устройства;

Для выполнения последнего правила используется диффузионный механизм распространения заряда через полупроводник и соответствующее управление потенциалами области истока и цели.

1.2.10. Синронная и асинхронная логики. Самосинхронные схемы, использование самосинронных схем в потоковых процессорах

Одна из важнейших задач, решающая проблему координации событий (сигналов, операций или процессов) в аппаратуре цифровых систем – синхронизация. Она связана, в основном, с обеспечением интерфейса между физическим (естественным) и логическим (искусственным) временем [46]. Координация событий отражает причинно-следственные связи между ними и обычно определяется последовательностью множества событий, происходящих в системе. Это близко к понятию логического времени, течение которого отмечается событиями. При этом любая система функционирует в непрерывном физическом времени.

В середине прошлого века активно исследовались две альтернативные методологии синхронизации элементов в аппаратуре: синхронная (С) и асинхронная.

В синхронной методологии интерфейс между физическим и логическим (системным) временем определяется системными часами, удаляющими физическое время из поведения модели. События во внешних часах отделены от модели системного поведения и не имеют завершенного причинно-следственного отношения к событиям в системе. Все события в синхронной системе инициируются

метками физического времени - синхросигналами (*clk,* см. рис. 1.19). Действительная длительность инициированных событий никак не отслеживается. Чтобы синхронизируемая аппаратура работала корректно, период синхросигналов выбирается из расчета на наихудший случай – максимально возможное время переключения отдельных элементов и распространении сигналов (с учётом паразитных элементов) при одновременном сочетании неблагоприятных условий функционирования (повышенное/пониженное напряжение питания, повышенная/пониженная температура, разброса технологических параметров изготовления микросхемы, значений паразитных элементов, входной ёмкости и т.п.). Таким образом, цена корректной работы синхронной аппаратуры – недоиспользование ее возможностей по быстродействию (до 120 % по сравнению с номинально возможным быстродействием).

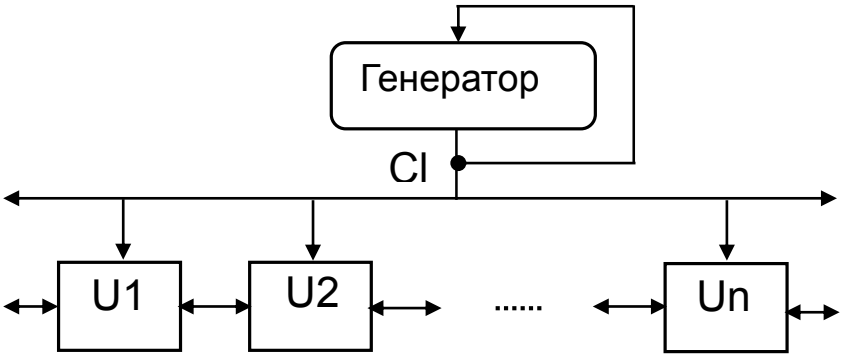


Рис. 1.19. Синхронизация в синхронной методике.

В асинхронной логике периоды, как правило, не постоянны и они определяют разрешённые и запрещённые времена прохождения сигналов через логические схемы и времена сохранения состояния в триггере (то есть время, в течение которого состояние на входе данных триггера запоминается). В отличие от синхронных схем, когда функционально законченные блоки микросхемы работают от одного синхросигнала, в асинхронных схемах блокирующего сигнала либо нет, либо он организуется локально для выполнения минимальных

функций. В традиционной асинхронной логике используется запрос-ответное взаимодействие, то есть сначала от одного вычислительного узла посылается сигнал запроса другому узлу и ждётся получения сигнала ответа, приход которого означает окончание действия. В самом вычислительном узле (как правило, выполняющем минимальную функцию) для формирования сигнала ответа используются схемы задержки, определяющие максимальную длительность прохождения сигналов в узле при наихудших условиях функционирования (максимальные и минимальные температуры и напряжение питания, технологические разбросы).

Одним из вариантов асинхронной логики является самосинхронная логика (СС). В западной литературе имеется также термин *self-timed* (*ST*) – самосинхронные схемы, применяемый к различным типам асинхронных схем с обратной связью. В самосинхронной логике оба периода (разрешённый и запрещённый) определяются по фактическому окончанию переходных процессов, что позволяет избежать гонок (ситуаций, когда из-за нарушения порядка прихода сигнала возникает ложный импульс, могущий привести к сбою в работе схемы), которые принципиально могут быть в традиционной асинхронной схемотехнике. Механизмы, обеспечивающие системное время в СС-подходе, включены в модель системного поведения и должны быть разработаны вместе с созданием начальной поведенческой спецификации. Корректные СС-системы базируются на механизме фиксации действительного окончания переходных процессов, т.е. на определении момента завершения переключений состояний элементов схемы. При этом обеспечивается правильное функционирование таких схем независимо от задержек элементов, их составляющих и условий функционирования. Другое важное свойство самосинхронных схем – отказобезопасность или самопроверяемость, позволяющее зафиксировать отказ схемы. Отсутствие гонок и отказобезопасность – основные свойства СС схем, кроме того, отсутствие синхросигнала и минимизация задержек позволяет понизить потребление питания и увеличить быстродействие в ряде случаев. Главные недостатки такого проектирования – сложность проектирования, отсутствие развитых средств проектирования и необходимость комбинирования с синхронными схемами, что снижает эффективность СС схем.

С момента появления теории Маллера [47] СС-проектирование было областью активных исследований. В последние годы интерес к СС-схемотехнике повышается по мере нарастания проблем в синхронной схемотехнике, отмечается рост числа публикаций по этой тематике. Вопросы проектирования СС-схем обсуждаются на таких известных конференциях, как *ICCD, DAC, EDAC, FTSC, ICCAD, AR_VLSI, ISAS* и т.п. Перечень работ, в которых освещаются научные проблемы и практические результаты разработок в области СС-схем, достигает более полутора тысяч [48].

Однако, несмотря на многочисленные потенциальные преимущества СС-схем, коммерчески выпускаемых СС-схем немного. Основная масса схемотехнических решений относится к классу квази-самосинхронных, а не СС-схем. Такого рода схемы представляют собой комбинации СС-фрагментов, в которых реализован контроль действительного окончания переходных процессов, и асинхронных блоков. В квази-самосинхронных схемах вместо контроля окончания событий действует гипотеза ограниченного времени протекания событий, реализуемая встроенными элементами задержки (в асинхронных реализациях) или настройкой генераторов импульсов (в синхронных реализациях). Это позволяет уменьшить число транзисторов при реализации схем и использовать стандартные средства САПР СБИС для проектирования цифровых устройств. Однако теряется основное преимущество СС-схем – независимость их поведения от задержек, и, как следствие, пропадает возможность бестестовой самодиагностики и локализации неисправностей − определяющих качеств при построении надежностно-ориентированной аппаратуры. В полной мере данными свойствами обладают только независящие от задержек (*Delay Insensitive, DI*) схемы, причем независящие от задержек, как в элементах, так и в цепях.

Основные преимущества *DI*-схем:

● устойчивая работа – отсутствие сбоев при любых возможных условиях эксплуатации;

● безопасная работа – прекращение всех переключений в момент появления отказа любого элемента (константной неисправности, при которой выход элемента "залипает" в одном состоянии).

Практические следствия этих свойств *DI* -схем:

● устойчивость к параметрическим отказам, вызываемым изменением параметров элементов из-за процессов старения и неблагоприятных воздействий окружающей среды;

● естественная стопроцентная самопроверяемость и самодиагностируемость по отношению к множественным константным неисправностям;

● безопасность функционирования на основе бестестовой локализации неисправностей – прекращение работы в момент отказа элемента, исключающее выдачу недостоверной информации, с одновременной индикацией места события;

● максимально возможная область эксплуатации (диапазон работоспособности), определяемая только физическим сохранением переключательных свойств активных элементов.

Перечисленные свойства *DI*-схем обеспечивают высокую эффективность создания надёжных изделий, в том числе и отказоустойчивых.

К *DI*-схемам, имеющим практическую значимость, относятся, например, *NCL*-схемы [49]. Но они обладают слишком большой аппаратной избыточностью. С этой точки зрения, более практичным является подкласс схем, независящих от задержек в элементах (*Element's Delay Insensitive, EDI*). В пределах эквихронной зоны они обладают всеми свойствами *DI*-схем.

Практически целесообразным являются также использование квази-*EDI* схем (КНЗЭ). Основное отличие *EDI* -схем от квази-*EDI* состоит в том, что *EDI*-схемы контролируют с помощью индикаторной подсхемы окончание переключения каждого элемента в схеме, в то время как квази-*EDI* обеспечивают индикацию только элементов, стоящих в критических путях обработки информации. За счет этого квази-*EDI* схемы оказываются более быстродействующими и менее сложными, но не дают стопроцентной гарантии сохранения работоспособности при изменении в широком диапазоне условий эксплуатации (напряжения питания, температуры) и при воздействии экстремальных факторов.

Успех проектирования цифровых схем любого типа во многом определяется составом библиотеки элементов, на основе которой ведется проектирование. Качество *EDI*-схем зависит от этого в боль-

шей степени, чем качество синхронных схем. Необходимость индицировать окончание переходных процессов во всех элементах *EDI*-схемы делает нежелательной сильную функциональную декомпозицию. Она приводит к появлению в схеме множества "мелких" логических элементов, каждый из которых требует дополнительных аппаратных затрат для реализации его индикации. Это делает целесообразной разработку библиотеки с широкой номенклатурой логических, триггерных и индикаторных элементов. При этом функциональный состав библиотеки определяется, в первую очередь, целесообразностью использования тех или иных элементов в *EDI*-схемах, а во вторую очередь – технологическим базисом реализации проектируемой БИС.

Рассмотрим основные подходы при создании СС схем. Маллер ввёл понятие С-элемента, выполняющего логическую функцию и имеющего гистерезис. Выход С-элемента отражает состояние входов после окончания переходных процессов. Выход элемента остаётся в данном состоянии до тех пор, пока новые переходные процессы не окончатся. То есть переходные процессы в схеме не должны влиять на выход схемы и необходимо знать (устанавливать) момент завершения переходных процессов. Сигнал, определяющий момент окончания переходных процессов называется индикатором [50]. Этот сигнал по-разному формируется для логических схем и памяти. Для формирования его в логике используется избыточное кодирование. В частности, можно принять кодирование, когда значение 1 кодируется как 01, а значение 0 как 10. Такой код называется парафазным. В этом случае истинными комбинациями становятся комбинации чисел 01 и 10 и переходными 00 и 11. В этом случае индикатор должен формироваться тогда, когда отсутствуют любая из комбинаций, где имеются 00 или 11. Рассмотрим примеры реализации СС схем и формирование индикаторов. Парофазные сигналы *a* и *b* с выходов логической схемы ЛС (см. рис. 1.20а) поступают на схемы «исключающее ИЛИ», определяющие отсутствие запрещённых состояний. При завершении переходных процессов на обоих выходах (схема И) формируется индикатор *I*. Однако уже эта простая схема требует достаточно большие аппаратные ресурсы для формирования индикатора. Схему можно упростить за счёт замены «исключающего ИЛИ» на схему «ИЛИ» (см. рис. 1.20б). Однако при этом могут появиться ряд

промежуточных состояний, когда реально переходные процессы не окончены, а индикатор появился. Для исключения этого вводятся разрешённые переходы. В этом случае для схемы рис. 1.20 б запрещаются переходные состояния, когда, например, выход *a* закончил переход из 01 в 10, а выход *b* ещё не начал переход из 10 в 01. Разрешённые наборы переходов называются спейсерами. В этом случае переход из одного состояния в другое осуществляется в 2 этапа или фазы: из начального набора через промежуточные к спейсеру и от спейсера через промежуточный набор к конечному. Такая дисциплина называется двухфазной, фазу перехода к спейсеру называют спейсерной, а от спейсера к рабочему набору – рабочей фазой [50].

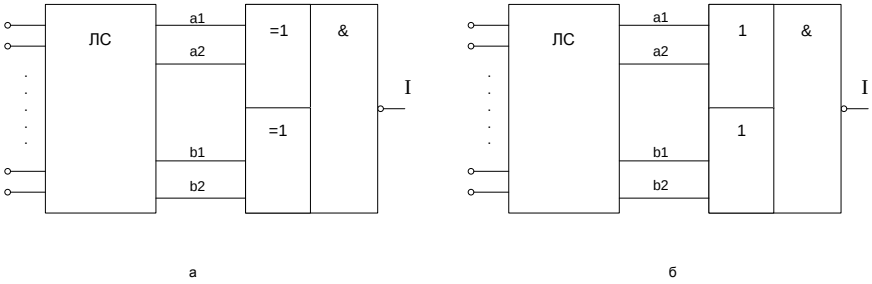


Рис. 1.20. Формирования индикатора в логических схемах.

Формирование индикатора для *RS*-триггера приведено на рис. 1.21. Для схемы 1.21а индикатор *I* формируется с использованием только выходов триггера, когда 1 определяет стабильное состояние, а 0 – переходное (напомним, режим хранения *RS*-триггера – на входах 1, запись происходит при приходе сигнала 0 на один из входов, 00 – неопределённое состояние). Однако такая схема не согласует входы триггера и выхода индикатора с фазой работы. Согласование достигается только когда на входах 00, что является запрещённым состоянием. На рис. 1.21б предусматривается запоминание данных и согласование фаз. Если выход триггера соответствуют входу, то переходные процессы закончились, в противном случае происходит переходной процесс.

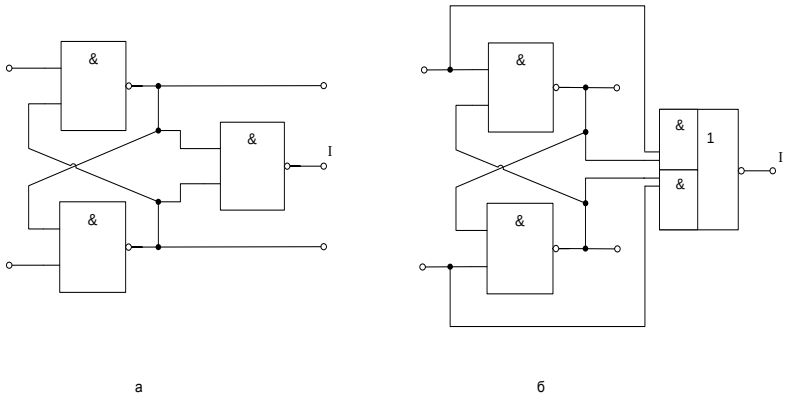


Рис. 1.21. Формирования индикатора в *RS*-триггере.

Обычно в *DI*-схемах используется парафазное и бифазное кодирование информационных сигналов. Это удваивает количество сигналов в интерфейсных шинах между удаленными формирователем и приемником многоразрядных данных. Следствие этого –потери в энергопотреблении и площади топологической реализации цифровых *DI*-устройств. Триггер с унарным входом, показанный на рис. 1.22 [51], позволяет сократить вдвое ширину шины данных, не нарушая запрос-ответного принципа взаимодействия между соседними *DI*-устройствами. Это уменьшает как энергопотребление, так и общий уровень помех.

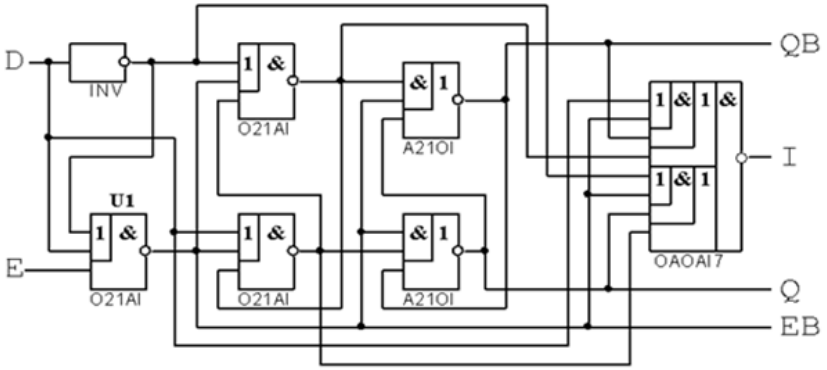


Рис. 1.22. Триггер для входного интерфейса *DI*-схемы.

Унарный информационный вход *D* сопровождается управляющим сигналом *E*, общим для многоразрядной шины данных. Сигнал *E* регулирует фазы работы триггера, обеспечивая прием правильного значения информационного входа *D* и отсутствие гонок на входе триггера. Элемент *U1* на входе триггера обеспечивает индикацию инвертора на входе триггера. Другим узким местом *DI*-схем является формирование информационных выходов триггеров с большой нагрузочной способностью для передачи информации в удаленные цифровые устройства. Традиционный путь – использование триггеров, способных работать на большую нагрузку. Для этого применяются более мощные логические элементы, транзисторы в которых имеют б***о***льшую ширину канала и занимают, соответственно, б***о***льшую площадь при топологической реализации. Альтернативой таким элементам служит триггер, изображенный на рис. 1.23 [52].

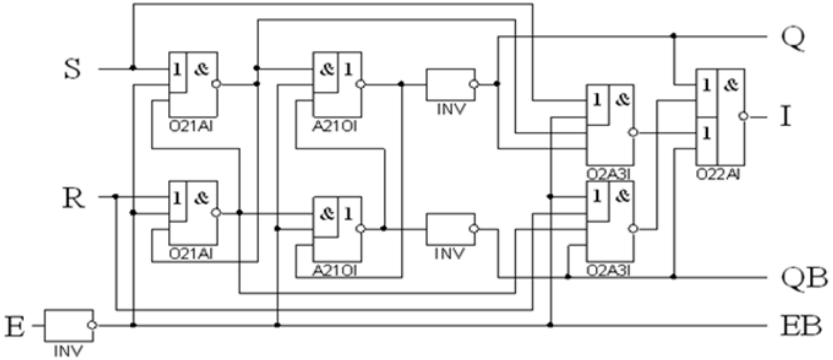


Рис. 1.23. Триггер для выходного интерфейса *DI*-схемы.

Информационные выходы триггера *Q, QB* формируются инверторами; их мощная реализация занимает гораздо меньше места, чем реализация умощненной бистабильной ячейки. Кроме того, инверторы на выходе изолируют выходы бистабильной ячейки, хранящей состояние триггера, от внешних цепей, что повышает помехоустойчивость триггера. Быстродействие же *EDI*-схем определяется реальными, а не наихудшими, условиями эксплуатации. Именно поэтому *EDI*-устройства в нормальных условиях оказываются, как правило, быстрее синхронных аналогов.

Модели библиотечных элементов, используемые системами логического моделирования, содержат предупреждения о возможном

нарушении дисциплины формировании сигналов, поступающих на входы элемента. Это позволяет разработчику *DI*-схемы на этапе функционально-логического моделирования избежать ошибок, связанных с неправильной организацией запрос-ответного взаимодействия между блоками и элементами в схеме.

Механизмы, обеспечивающие системное время в СС-подходе, включены в модель системного поведения и должны быть разработаны вместе с созданием начальной поведенческой спецификации. СС-схемы потенциально должны выигрывать по быстродействию и приводить к снижению потребляемой мощности до 30%. Однако при моделировании и исследовании тестовых кристаллов СС-схем, спроектированных и изготовленных с нормами 65-180 нм, выяснилось, что имеется ряд ограничивающих факторов. СС-схемы весьма чувствительны к перекрестным наводкам, схемы правильно отрабатывают свои функции, однако из-за перекрестных наводок падает быстродействие на десятки процентов, несмотря на то, что размер блоков составляет единицы кв. мм. Таким образом, проектирование СС-схем требует дополнительных усилий по уменьшению различного рода паразитных составляющих, в противном случае заметно падает быстродействие. Представим теперь, что будет происходить на схеме с площадью 100 – 300 мм^2 - площади современного высокопроизводительного микропроцессора. В синхронных схемах принципиальная электрическая схема клокового дерева микропроцессоров очень проста, однако её топология делается по специальному маршруту. Ряд ведущих компаний, включая компанию *Intel*, топологию клокового дерева делают вообще заказным образом для достижения максимальной частоты. То есть вкладываются огромные ресурсы для достижения максимальных частот. В случае СС-схем подобные работы провести невозможно в силу сложности логических уравнений. То есть потери в быстродействии на больших размерах кристалла будут катастрофические. Проведенные оценки показывают снижение быстродействия в несколько раз на размерах порядка 10 мм с нормами 65 нм. Для норм 45 нм и ниже проблемы с перекрестными наводками и импульсными помехами становятся особенно критичными. Кроме того, для таких норм возникают проблемы разброса параметров транзисторов и существенном возрастании токов утечки. В резуль-

тате для достижения предельных параметров становится необходимо оптимизировать каждый проект под конкретный технологический маршрут. Если линейный размер ядра микропроцессора не превышает 2 мм, внутри ядра сохраняются корреляции статических параметров элементов и возможна схемотехническая компенсация технологических вариаций. Для СС-логики эти разбросы должны приводить к снижению быстродействия. Указанные проблемы выдвигают необходимость снижения размера ядер и отдельных функциональных узлов. Именно поэтому многие компании ведут активные работы по созданию микропроцессоров с десятками ядер и созданию сети на кристалле. Для СС-логики это, пожалуй, единственный вариант развития высокопроизводительных микропроцессоров. То есть проект должен разбиваться на многие функционально законченные узлы с небольшой площадью на кристалле (порядка 1 мм^2).

В настоящее время высокопроизводительные микропроцессоры имеют 2 основных вектора развития – модернизация архитектур универсальных микропроцессоров и создание потоковых, прежде всего графических процессоров. Именно на графических процессорах сегодня достигаются предельные производительности. Однако программирование графических процессоров принципиально иное и зачастую проще подготовить новых программистов, чем переквалифицировать старых. Задача компилятора потоковой машины состоит в переводе программ с внешнего языка в их эквиваленты на язык программ графов. Применение СС-логики наиболее естественно для потоковых процессоров. В такой архитектуре вычислительная задача представляется в виде потокового графа, каждый узел которого реализуется на соответствующих вычислительных узлах. Необходимо дать однозначное соответствие каждого узла графа и вычислительного узла и передачу информации между узлами для организации потока вычислений. Для каждой посылки данных должна однозначно определяться стадия выполнения на вычислительном графе задачи. Поскольку необходимо в общем случае передать данные с выхода одного вычислительного узла потоковой машины на вход любого другого, необходимо организовать систему коммутации. Как и в любой системе коммутации, в пересылаемых данных должны быть выделены поля, определяющие направления

коммутации. Кроме того, необходимо определить набор данных, передающихся на один и тот же вычислительный узел одновременно для исключения сбоя процесса вычислений. Вычисление операции (программы узла) активизируется при появлении операндов на всех входах узла - модель вычислений, основанная на управлении данными (*data driven computation*). Нет необходимости использовать счетчик команд, который обязателен в универсальных ЭВМ. Выполнение команды активизируется самими данными, что позволяет осуществлять одновременное исполнение многих команд в машине. Таким образом, во всей цепочке вычислений имеется причинно-следственная связь, которая естественным образом реализуется в СС-логике. В такой постановке задачи при разбиении на относительно мелкие вычислительные узлы (функции) снижается зависимость СС-логики от наличия различных паразитных элементов, перекрёстных наводок и пр. свойственных современным кристаллам с предельными технологическими нормами. Возможно создание гибридного процессора, когда часть функций (управление, ввод/вывод, статистика, и пр.) лежит на стандартном процессоре, а сопроцессор реализован как потоковая машина (см. рис. 1.24).

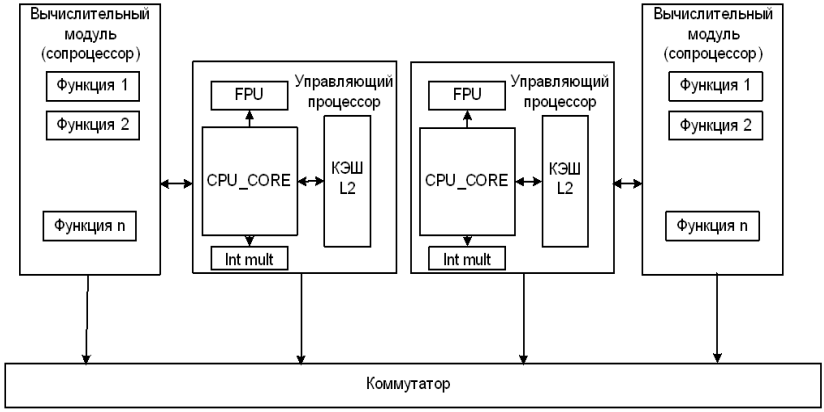


Рис. 1.24 Структурная схема гибридного процессора.

На рис.1.21 для примера показана структурная схема процессора только с двумя узлами, при создании микропроцессора с предельными технологическими нормами число узлов может достигать 16

или более. Вычислительный модуль организован как потоковая машина на базе СС-узлов. Данная архитектура процессора является переходной, когда используются как синхронные, так и самосинхронные узлы. Однако поскольку вычислительные модули занимают большую часть кристалла по сравнению с универсальными ядрами, а процесс вычислений во многом определяется вычислительными ядрами, надёжность такого процессора приближается к надёжности полностью СС-процессора. На рис. 1.24 не показаны коммуникационные каналы и внешние интерфейсы, поскольку в данном рассмотрении они не принципиальны.

1.3. Архитектура микропроцессоров

1.3.1. Классификация микропроцессоров

Под архитектурой микропроцессора понимается набор команд, их формат и способ исполнения (например, счетчик команд). То есть это некий набор свойств и качеств, присущий целому семейству микропроцессоров. Имеются различные классификации архитектур микропроцессоров, как по организации, так и по назначению.

Архитектура микропроцессора во много определяет его производительность. Классифицировать архитектуру микропроцессоров можно по разным признакам. Первая классификация - по памяти, разделяемой или нет на память данных и программ. Соответственно архитектура разделяется на архитектуру Гарвардскую и фон Неймана. Гарвардская архитектура отличается от архитектуры фон Неймана тем, что программный код и данные хранятся в разной памяти. В такой архитектуре невозможны некоторые методы программирования (например, невозможно динамически перераспределять память между программным кодом и данными). Однако гарвардская архитектура больше защищена от сбоев и позволяет добиться большего быстродействия. Архитектура фон Неймана обладает тем недостатком, что она требует последовательных операций. Какой бы огромный массив данных ни требовалось обработать, каждый его байт должен будет пройти через центральный процессор, даже если над всеми байтами требуется провести одну и ту же операцию. Этот

эффект называется узким горлышком фон Неймана. Для преодоления этого недостатка предлагались и предлагаются архитектуры процессоров, которые называются параллельными. Наиболее ранней из известных классификаций параллельных архитектур вычислительных систем является классификация по Флинну, предложенная в 1966 году. Классификация базируется на понятии потока, под которым понимается последовательность элементов, команд или данных, обрабатываемая процессором. На основе числа потоков команд и потоков данных Флинн выделяет четыре класса архитектур: *SISD, MISD, SIMD, MIMD*. Эти четыре класса архитектур схематически представляются в виде квадрата, называемого квадратом Флинна (рис. 1.25).

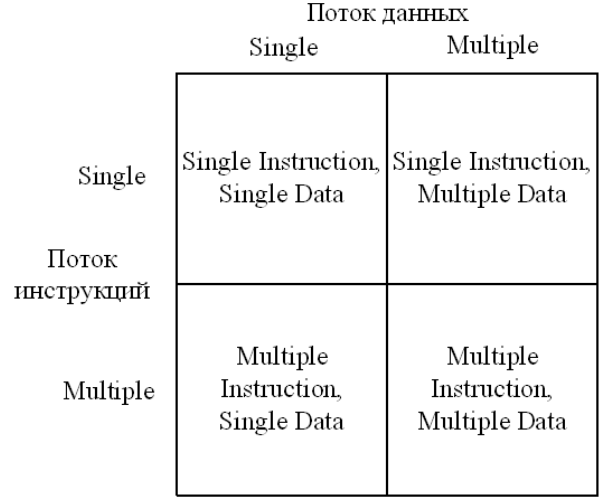


Рис. 1.25. Квадрат Флинна.

SISD (***S**ingle **I**nstruction, **S**ingle **D**ata*) — один поток команд, один поток данных (один процессор выполняет один поток команд, оперируя одним потоком данных);

SIMD (***S**ingle **I**nstruction, **M**ultiple **D**ata*) — один поток команд, много потоков данных;

MISD (***M**ultiple **I**nstruction stream, **S**ingle **D**ata stream*) — много потоков команд, один поток данных;

MIMD (***M**ultiple **I**nstruction stream, **S**ingle **D**ata stream*) — много потоков команд, много потоков данных.

Микропроцессоры с архитектурой ***SIMD*** состоят из одного командного процессора (управляющего модуля), и нескольких модулей обработки данных, называемых процессорными элементами. Управляющий модуль принимает, анализирует и выполняет команды. Если в команде встречаются данные, контроллер рассылает на все процессорные элементы команду, и эта команда выполняется на нескольких или на всех процессорных элементах. Каждый процессорный элемент имеет свою собственную память для хранения данных. Одним из преимуществ данной архитектуры считается то, что в этом случае более эффективно реализована логика вычислений. До половины логических инструкций обычного *SISD* микропроцессора связано с управлением выполнением машинных команд, а остальная их часть относится к работе с внутренней памятью процессора и выполнению арифметических операций. В *SIMD* компьютере управление выполняется управляющим модулем, а арифметические операции - процессорными элементами.

Архитектура ***MISD*** — тип архитектуры параллельных вычислений, где несколько функциональных модулей выполняют различные операции над одними данными. Этот тип архитектуры не сильно распространен. К нему относятся, в частности, отказоустойчивые компьютеры, выполняющие одни и те же команды избыточно с целью обнаружения ошибок.

MIMD архитектура достигает максимальный параллелизм вычислений. К этой архитектуре относятся многоядерные микропроцессоры, функционирующие асинхронно и независимо. В любой момент, различные процессоры могут выполнять различные команды над различными частями данных. Этот класс предполагает, что в вычислительной системе есть несколько устройств обработки команд, объединенных в единый комплекс и работающих каждый со своим потоком команд и данных. Для исключения проблем взаимной блокировки и состязания за обладание ресурсами синхронизация функционирования ядер обеспечивается механизмами, рассматриваемыми в главе 3. Для обмена данными между ядрами может использоваться протокол *MPI*. Обработка данных разделена на несколько потоков, каждый с собственным аппаратным состоянием ядра в рамках единственного определённого программным обеспечением процесса или в пределах множественных процессов.

По типу используемых инструкций микропроцессоры могут разделяться на процессоры *CISC, RISC* (включая суперскалярные и мультитредовые), *VLIW* и *EPIC* процессоры, как развитие *VLIW* архитектуры.

1.3.2. *CISC* процессоры

CISC (Complex instruction set computer) процессор - это процессор, где отдельные инструкции могут состоять из нескольких низкоуровневых операций (таких как: считывание из памяти, арифметическая операция и запись в память) и/или представлять собой многошаговые операции или адресации в пределах одной инструкции. Основные ограничивающие производительность микропроцессора положения архитектуры следующие: изменяемая длина инструкций и расширенный набор команд, включающий редко используемые и сложные команды.

Основоположником *CISC*-архитектуры можно считать компанию *IBM* с ее базовой архитектурой *S*/360, ядро которой используется с 1964 года и дошло до наших дней, например, в таких мейнфреймах как *IBM ES*/9000. Компания *IBM* создала линейку компьютеров с разной производительностью с одним и тем же набором команд. Это позволяло заказчику на первых порах использовать недорогую модель, после же роста компании или сложности решаемых задач проводить модернизацию с использованием более мощной ЭВМ без изменения программного обеспечения. Для этого *IBM* впервые применила технологию микрокода, который применялся во всех моделях серии, кроме самых старших. Затраты на разработку *System*/360 составили около 5 млрд. долларов США (что соответствует 30 млрд. в ценах 2005 г., если сравнивать с 1964). Это был второй по стоимости проект 1960-х годов после программы «Аполлон».

Лидером в разработке *CISC* микропроцессоров с полным набором команд считается компания *Intel* с серией микропроцессоров *x*86 и *Pentium*. Эта архитектура является практическим стандартом для рынка персональных компьютеров.

Для *CISC*-процессоров характерно:

- сравнительно небольшое число регистров общего назначения;

• большое количество машинных команд, некоторые из которых нагружены семантически аналогично операторам высокоуровневых языков программирования и выполняются за много тактов;
• большое количество методов адресации, включая косвенную;
• большое количество форматов команд различной разрядности;
• преобладание двухадресного формата команд;
• наличие команд обработки типа регистр-память.

1.3.3. RISC процессоры

Существенной инновацией в микропроцессорной архитектуре было изобретение архитектуры *RISC* (*Reduced instruction set computer*) Давидом Паттерсоном в Беркли и Джоном Хеннесси в Стэнфорде. Основной их идеей было то, что, жертвуя сложностью инструкций, предпочитая более простые, они существенно улучшали второй член формулы (1.3), а поскольку более простые инструкции соответствовали более простым микросхемам, которые могли работать на более высоких частотах, улучшался и третий член (1.3). Первый член (1.3) несколько ухудшался, поскольку в программах теперь было гораздо больше инструкций, но второй и третий более чем перекрывали это незначительное ухудшение. Событием водораздела был выпуск в 1986 году первого широко распространённого коммерческого *RISC*-процессора *Sun*-4 компании *SPARC*, работающего на частоте 16 МГц. Это был втрое более производительный процессор, чем распространённый тогда же *Digital VAX*-11/780. Как видно из рисунка 1.26 [3], внедрение *RISC* архитектуры позволило произойти экспоненциальному росту частот работы процессоров. Каждый из разработчиков микросхем, включая Intel, применял эту архитектуру. Паттерсон и Хеннесси описывали это так [3]:

"Вдобавок, два существенных изменения на рынке персональных компьютеров сделали гораздо более лёгкой, чем раньше возможность коммерческого успеха с новой архитектурой. Во-первых, практический уход от программирования на ассемблере уменьшил необходимость в совместимости объектного кода. Во-вторых, создание стандартных независимых от поставщика оборудования операционных систем, таких как *Unix* и *Linux*, снизило стоимость и риск

внедрения новой архитектуры. Эти изменения дали возможность успешно разработать новое множество архитектур с более простыми инструкциями, называемыми *RISC* (компьютер с сокращённым набором команд) архитектурами в начале 80-х. Машины, основанные на *RISC*, привлекли внимание разработчиков двумя критическими технологиями для роста производительности - использование параллелизма инструкций (сначала через конвееризацию, а позже через исполнение многих инструкций) и использование КЭШей (сначала в простых формах, потом в более изощрённых реализациях и оптимизациях). Машины, основанные на *RISC*, подняли планку быстродействия, вынуждая существующие архитектуры догонять или умирать. *VAX* не смог, так что был заменён *RISC*. *Intel* ответил на вызов, переведя внутренне инструкции 80*x*86 в *RISC*-подобные, что позволило ему применить многие из инноваций, опробованных на первых *RISC*-процессорах. Поскольку количество транзисторов резко выросло в конце 90-х, аппаратный оверхед транслирования более сложных *x*86 команд стал пренебрежимо мал. В приложениях, таких как приложениях для телефонов, стоимость мощности и площади кристалла для *x*86-трансляции привела к выходу в лидеры *RISC*-архитектуры *ARM*, получившей лидерство в этих областях".

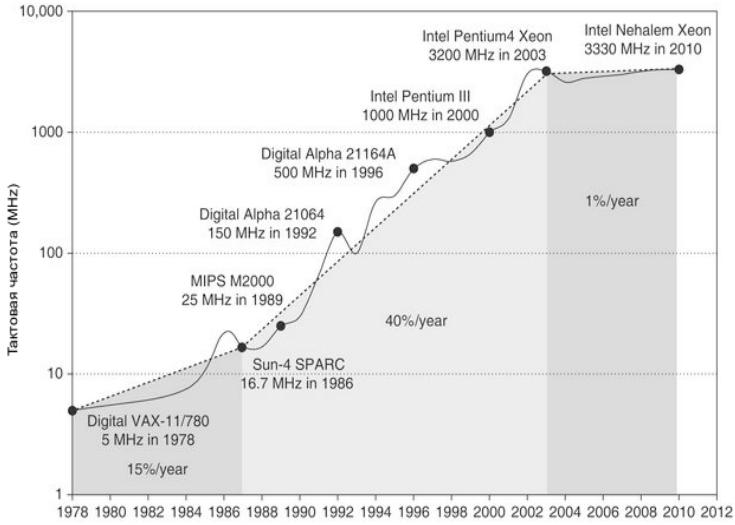


Рис. 1.26. Рост тактовой частоты микропроцессоров.

Таким образом, можно дать следующее определение *RISC* процессору - это процессор, основанный на стратегии упрощённых инструкций, обеспечивающих большую производительность процессора за счёт более быстрого выполнения каждой инструкции. Значительное уменьшение числа транзисторов ядра микропроцессора по сравнению с архитектурой *CISC* позволяет добиться внутреннего параллелизма и увеличить объём регистров. Основные особенности архитектуры *RISC* следующие:

- фиксированный формат инструкций, использующий одно слово с унифицированными полями инструкций и обеспечивающий упрощённое декодирование;
- минимизированный набор команд, аппаратно реализуются только базовые команды, редко используемые сложные инструкции реализуются через набор базовых инструкций;
- одинаковые регистры общего пользования, позволяющие любой регистр использовать в любом контексте, упрощая тем самым разработку компилятора (однако регистры плавающей арифметики, как правило, выделяются в отдельную подгруппу);
- упрощённая адресация, сложная адресация (косвенная адресация) реализуется через набор более простых операций; невозможность переписать содержимое одной ячейки памяти в другую ячейку памяти в одной инструкции;
- выровненные адреса операндов, позволяющие избежать нескольких циклов чтения/записи операндов из/в память.

Развитие архитектуры *RISC* в значительной степени определялось прогрессом в области создания оптимизирующих компиляторов. Именно современная техника компиляции позволяет эффективно использовать преимущества большего регистрового файла, конвейерной организации и большей скорости выполнения команд. Современные компиляторы используют также преимущества другой оптимизационной техники для повышения производительности, обычно применяемой в процессорах RISC: реализацию задержанных переходов и суперскалярной обработки, позволяющей в один и тот же момент времени выдавать на выполнение несколько команд

Развитием *RISC* процессоров являются суперскалярные *RISC* процессора. Суперскалярность это способность выполнения нескольких машинных инструкций за один такт процессора путем увеличения числа исполнительных устройств. Основными компонентами суперскалярного процессора являются устройства для интерпретации команд, снабженные логикой, позволяющей определить, являются ли команды независимыми, что даёт принципиальную возможность их одновременного исполнения (если команды зависимы, должна обеспечиваться последовательность исполнения), и достаточное число исполняющих устройств. В исполняющих устройствах могут быть конвейеры. Суперскалярные процессоры реализуют параллелизм на уровне команд.

Появление этой технологии привело к существенному увеличению производительности, в то же время существует определенный предел роста числа исполнительных устройств, при превышении которого производительность практически перестает расти, а исполнительные устройства простаивают. Как правило, число одновременно выполняемых инструкций не превышает 4.

Возможность одновременного выполнения инструкций во много решается механизмом предсказанием переходов. Суть метода заключается в прогнозировании направления ветвления программы и начале выполнения предполагаемых операций. Повышение эффективности данного метода является актуальным для любого современного RISC микропроцессора. Это объясняется тем, что в обычных программах порядка 20% инструкций являются командами ветвления, которые в простейших случаях реализации архитектуры приводят к остановке конвейера. В современных микропроцессорах глубина конвейера достигает нескольких десятков тактов, и остановка конвейера приводит к потере этих тактов и к соответственному снижению производительности. Базовые подходы к механизму предсказания ветвления одинаковые для всех RISC микропроцессоров – это статическое и динамическое предсказание ветвления. Однако реализация этой функции для разных микропроцессоров разная и занимает в случае динамического предсказания от 2 до 5 тактов, что становится сопоставимым с самим вычислением и приводит к соответствующему повышению потребляемой мощности. Поэтому

оптимизация блока предсказания ветвления для каждого микропроцессора проводится индивидуально.

Можно дать следующее определение модулю предсказания условных переходов (*Branch Prediction Unit, BPU*) — устройство, входящее в состав микропроцессоров, имеющих конвейерную архитектуру, определяющее направление ветвлений (предсказывающее, будет ли выполнен условный переход) в исполняемой программе. Предсказание ветвлений позволяет осуществлять предварительную выборку инструкций и данных из памяти, и начать выполнять инструкции, находящиеся после условного перехода, до того, как он будет выполнен. Предсказатель переходов является неотъемлемой частью всех современных суперскалярных микропроцессоров, так как в большинстве случаев (точность предсказания переходов в современных процессорах превышает 90%) позволяет оптимально использовать вычислительные ресурсы процессора. Как было сказано выше, существует два основных метода предсказания переходов: статический и динамический.

Статические методы предсказания ветвлений являются наиболее простыми. Суть этих методов состоит в том, что различные типы переходов либо выполняются всегда, либо не выполняются никогда. В современных реализациях статических предсказаний предполагается, что любой переход «назад» (т.е. переход на младшие адреса) является циклом и должен выполняться, а любой переход «вперед» (т.е. на старшие адреса) не выполняется.

Динамические методы, широко используемые в современных процессорах, подразумевают анализ истории ветвлений, обновляемой по результатам фактического выполнения команд переходов.

Другим решением повышения производительности является мультитредовая технология (*Multithreading*), аппаратно поддерживающая эффективное выполнение нескольких тредов (*thread*) для каждого ядра. Тред - это минимальный аппаратный функциональный блок микропроцессора, поддерживаемый операционной системой и использующий общие ресурсы с другими тредами (КЭШ память, регистры). Обычно это результаты от разделения компьютерных программ на 2 или более одновременно выполняемые задачи. Запуск многих тредов, это как чтение одной поваренной книги од-

новременно с разных страниц, если при приготовлении блюда освободилось сковородка до получения требуемого компонента блюда, на сковородке можно начать жарить пищу для другого блюда. В микропроцессоре треды делят общие ресурсы ядра: вычислительные блоки, кэш память, буфер трансляции адресов (*TLB*). В многоядерных микропроцессорах для каждого ядра имеется своя группа тредов. На рисунке 1.27 показан процесс выполнения программы для одноядерного процессора с 2 тредами. Мультитредовая организация позволяет одновременно выполнять не один, а несколько потоков команд, что дает возможность увеличить множество выполняемых команд, но важнее – усилить поток одновременно выполняемых операций с памятью. При выполнении определенных условий это дает повышение эффективности выполнения операций с памятью, определяемой уже не задержками выполнения, а темпом выдачи операций с памятью и получения результатов.

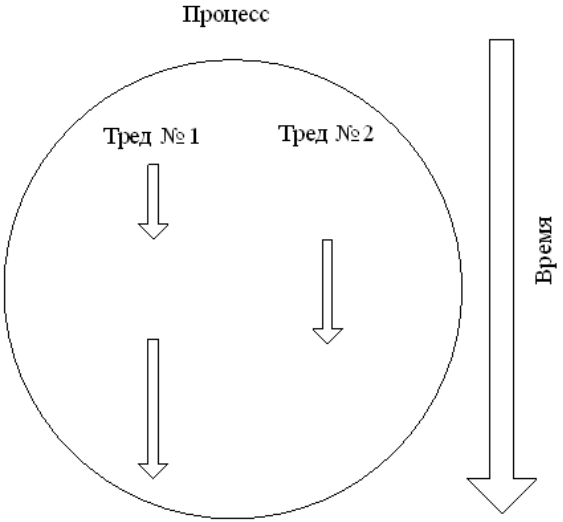


Рис. 1.27. Ход выполнения программы в одноядерном микропроцессоре с 2 тредами.

В России архитектуру суперскалярных *RISC* микропроцессоров развивает пока только одна компания – НИИСИ РАН. Это микропроцессоры с архитектурой *MIPS*64.

1.3.4. *VLIW* и *EPIC* процессоры

VLIW процессор (*Very long instruction word*) - это процессор с явно выраженным параллелизмом вычислений, заложенным в систему команд процессора. Архитектура характеризуется, прежде всего, наличием длинных (*Bundled*) инструкций, то есть инструкций для отдельных вычислительных узлов, связанных в одно длинное командное слово.

В суперскалярном микропроцессоре число вычислительных узлов не соответствует последовательности одновременно выполняемых команд. Каждой инструкции соответствует только одна команда. *VLIW* архитектура является архитектурой *MIMD*. Инструкция *VLIW* процессора состоит из нескольких операций, по крайней мере, по одной для каждого вычислительного узла. Например, если *VLIW* процессор содержит 5 вычислительных блоков, инструкция такого процессора должна содержать 5 полей в командном слове определяющих, что каждый блок должен делать в данный момент времени. Для реализации такой функции, как правило, ширина инструкции составляет 64 разряда или больше. В суперскалярных микропроцессорах возможность параллельного выполнения команд определяется на аппаратном уровне, то есть аппаратура определяет, являются ли последовательность команд зависимой или нет, могут ли они выполняться одновременно. В микропроцессорах *VLIW* возможность параллельного выполнения команд определяется на уровне компилятора, что приводит к существенному упрощению процессора.

Развитием *VLIW* процессоров является *EPIC* (*Explicitly Parallel Instruction Computing*) архитектура. Архитектура *EPIC* (базовые принципы) были разработаны в университете Иллинойса, проект имел название *Impact*. В начале 1990-х годов были заложены теоретические основы самой архитектуры, затем были начаты работы в рамках создания инструментальных средств для разработки процессора *EPIC*.

Архитектура *EPIC* имеет следующие основные особенности:

• Распараллеливание алгоритма между исполнительными модулями производится компилятором на этапе создания машинного кода, когда команды объединяются в связки и не конкурируют

между собой за ресурсы микропроцессора. При этом упрощается блок управления на кристалле. Формат команд имеет много общего с архитектурой с длинным командным словом - параллелизм так же явно выделен. Однако если процессор *VLIW* обычно имеет чётко заданную ширину (хотя в процессорах с нерегулярным длинным командным словом слова могут быть разными), в процессоре *EPIC* могут сопоставляться инструкции из различных тактов выполнения. То есть инструкция явно выполняется за 1-2 такта и в этом случае в образце инструкции

- определено, между какими командами имеется слот задержки (фактически "граница" между командами, выполняемыми в разных тактах). К каждой длинной команде прилагается небольшой ярлык, который специфицирует формат команды.
- Наличие большого регистрового файла.
- Применение предикатов. Предикатный подход исходит из предпосылки, что возросшие мощности микропроцессоров позволяют запускать параллельно команды из разных ветвей условного ветвления вместо того, чтобы ожидать формирования истинных признаков для выбора правильного направления или полагаться на блок предсказания переходов, рискуя прийти к необходимости перезагрузки достаточно длинных конвейеров в случае неудачного предсказания. При этом каждая команда снабжается специальным полем условия (предикатом) (см. рис. 1.28). По мере определения истинных признаков ветвления те команды, предикаты которых указывали, что они выбраны из другой ветви, снимаются с обработки в конвейере. Результаты команд не записываются в приемник до определения правильности направления перехода.

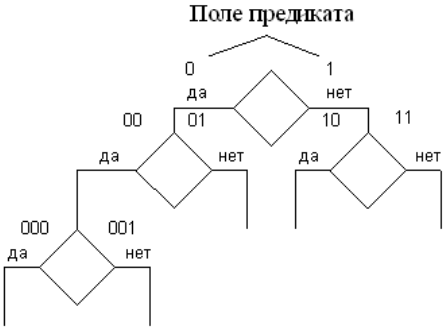


Рис. 1.28. Предикатное исполнение команд.

• Для предсказания инструкций используется поддержка компилятора.

Основные достоинства архитектуры *EPIC* следующие:

1. Упрощается архитектура процессора. Вместо логики распараллеливания на *EPIC*-процессоре можно разместить больше регистров, функциональных устройств и т. п.

2. Процессор не тратит время на анализ потока команд.

3. Возможности процессора по анализу программы во время выполнения ограничены сравнительно небольшим участком программы, тогда как компилятор способен произвести анализ всей программы.

4. Если некоторая программа должна запускаться многократно (а именно так и бывает в подавляющем большинстве случаев), выгоднее распараллелить ее один раз при компиляции, а не тратить на это время каждый раз, когда она исполняется на процессоре.

Однако архитектуре *EPIC* присущ и ряд недостатков:

1. Компилятор производит статический анализ программы, раз и навсегда планируя вычисления. Однако даже при небольших изменениях исходных данных путь выполнения программы существенно изменяется. Кроме того, для каждой программы компилятор может оптимизироваться по-разному.

2. Значительно усложняются компиляторы, следовательно, увеличиваются время компиляции программы и число ошибок в самих компиляторах. Если первый фактор, учитывая высокое быстродействие современных компьютеров, не очень существенен, то на вто-

рой следует обратить определенное внимание. Исследования показывают, что к моменту поставки даже ответственного программного обеспечения в нем содержится примерно 1 ошибка на 10000 строк исходного кода. Следовательно, программа из 500 тыс. строк будет содержать около 50 ошибок. И эти ошибки могут проявиться самым неожиданным образом.

3. Производительность микропроцессора во многом определяется качеством компилятора.

4. Увеличивается сложность отладки, так как отлаживается не исходная программа, а оптимизированный параллельный код. Программисту тяжело определить место и причину появления ошибки, так как в процессе трансляции исходной программы ее отдельные команды будут переставлены компилятором для обеспечения оптимальности работы микропроцессора.

Наибольший вклад в разработку микропроцессоров с архитектурой *EPIC* внесла компания *Intel*. Помимо компании *Intel* микропроцессоры с подобной архитектурой разрабатывали компания *Transmeta* и российская компания МЦСТ.

Transmeta разрабатывала 2 типа микропроцессоров: для настольных малопотребляющих компьютеров и для высокопроизводительных систем, у нее не было задачи создать микропроцессор под множественные области применения, и тем не менее архитектура проиграла конкурентную борьбу и компания прекратила своё существование. Компания *Intel* разрабатывала микропроцессор с архитектурой *IA-64* прежде всего для высокопроизводительных систем. Сейчас данное направление работ многими признано ошибочным и её микропроцессоры с архитектурой *IA-64* могут конкурировать только для математических расчетов и, если бы не огромные ресурсы компании, проект также закончился бы ничем. Даже огромные ресурсы компании не позволили сделать заметное число высокопроизводительных комплексов на базе микропроцессоров с архитектурой *IA-64*. В *TOP500* за 2011 г. находятся только 6 компьютеров с архитектурой *IA-64*, для примера 200 компьютеров из *TOP500* сделано с микропроцессорами *Xeon E*54, 55 с микропроцессорами *Xeon L55,* 49 - *Xeon*51. Наиболее мощные компьютеры создала компания IBM на микропроцессорах *PowerXCell 8i* и *PowerPC 450*. Другим лидером является микропроцессор *AMD x86_64 Opteron Quad Core*.

Российская компания МЦСТ продолжает работы по развитию данной архитектуры.

Во всех высокопроизводительных микропроцессорах используется конвейерная (*pipelining*) обработка данных с целью повышения быстродействия микропроцессора. Процесс вычисления разбивается на отдельные части – стадии. На каждой стадии выполняется отдельная часть операции и далее результат операции передаётся на следующую стадию через регистр. Чем больше число стадий, тем большую частоту микропроцессора можно достичь. Обычно для выполнения каждой команды требуется осуществить некоторое количество однотипных операций, например: выборка команды из ОЗУ, дешифровка команды, адресация операнда в ОЗУ, выборка операнда из ОЗУ, выполнение команды, запись результата в ОЗУ. Каждую из этих операций сопоставляют одной ступени конвейера. В конвейере микропроцессора с архитектурой *MIPS*-1 четыре стадии:

- получение и декодирование инструкции,
- адресация и выборка операнда из ОЗУ,
- выполнение арифметических операций,
- сохранение результата операции в регистре.

В следующих архитектурах *MIPS* число стадий последовательно увеличивалось до 5, 7, 9. В ряде современных микропроцессорах число стадий может достигать нескольких десятков. Преимущества конвейера очевидны. Если время выполнения команды составляет t и при его разбиении на m равных частей (*m*-стадийный конвейер), оно не меняется, то для выполнения n команд в процессоре без конвейера потребуется время равное $n \times (t / m) \times m$, а в процессоре с конвейером это время составит $t+(n-1) \times (t / m)$. Однако существуют дополнительные преимущества конвейера в том, что при разбиении времени выполнения инструкции на отдельные части длительности этих частей можно уменьшить за счёт уменьшения в каждом блоке, реализующем требуемую функцию стадии, длин соединений, паразитных составляющих, логики (упрощение мультиплексоров и пр.). Именно для этого увеличивают число стадий. Однако при нарушении конвейера, например, в случае ошибки в предсказании условного перехода, задержки при многостадийных конвейерах начнут сильно возрастать на новую загрузку конвейера, что приведёт уже к

потере производительности. Производительность микропроцессора будет разная для разных задач. Именно поэтому часто микропроцессоры с разным числом стадий ориентируются под разные классы задач. Для задач реального времени используются, как правило, микропроцессоры с небольшим числом стадий (меньше 9), в персональных же компьютерах используются многостадийные микропроцессоры.

Глава 2. Многопроцессорные системы. СуперЭВМ

2.1. Задачи, решаемые на суперЭВМ

В последние десятилетия одним из основных признаков высокого технического развития страны являлось наличие в стране собственной суперЭВМ. И это понятно почему. Во-первых, создание суперЭВМ требует наличия высоких технологий в самых различных областях: инженерных, электронных, программных. Во-вторых, такие ЭВМ нацелены на решение сложных технических задач, то есть если страна не способна ставить и решать такие задачи, то и суперЭВМ не нужна. К концу 20 века потенциал инженерных технологий теорий микроскопического взаимодействия в рамках моделей сплошной среды был практически исчерпан. Дальнейшее развитие ядерной и термоядерной энергетики, электроники, авиастроения, биотехнологий и т.д. стало невозможным без проведения полномасштабных инженерных расчётов таких сложных технических и биологических систем с учётом атомно-молекулярного взаимодействия, требующих уже в среднесрочной перспективе (до 2020 г.) использования суперЭВМ эксафлопного класса (10^{18} оп/сек). Только те страны, которые будут иметь такие ЭВМ и соответствующее программное обеспечение, способны будут создавать принципиально новые изделия [53]. Таким образом, создание суперЭВМ является одним из актуальнейших направлений развития техники. И те специалисты, которым посчастливится создавать такие ЭВМ, будут на переднем крае развития науки.

Рынок вычислительных средств повышенной производительности принято делить на четыре сегмента. Высший сегмент (*technical capability*) – это рекордные по быстродействию и реальной производительности стратегические вычислительные средства, установленные в крупнейших государственных лабораториях и центрах и решающих важнейшие для государства задачи. Оставшиеся три сегмента систем менее чувствительных к реальной производительности – это корпоративные системы (*technical enterprise*), системы уровня отделения (*technical divisional*) и системы уровня отдела (*technical department*).

Современные суперЭВМ – это многопроцессорные ЭВМ с организацией быстрой доставки в/их ОЗУ данных и быстрыми каналами обмена данных, как между отдельными микропроцессорами, так и между модулями и крейтами. Протоколы обмена между микропроцессорами, модулями и крейтами отличаются от протоколов обмена сети на кристалле, однако могут быть унифицированы на всех уровнях.

В течение 10-12 лет производительность суперЭВМ возрастает в среднем в 1000 раз. Очередным этапом является достижение производительности 1 Эксафлоп/с (10^{18} оп/с), планируемое на 2018-2020 гг. Основные трудности в достижении эксафлопсной производительности – необходимость эффективного функционирования 10^8-10^9 процессорных ядер и преодоление физических ограничений, обусловленных энергопотреблением, надежностью и конструктивными размерами.

Из обширного перечня областей человеческой деятельности, задачи которых требуют применения вычислительных систем эксафлопсной производительности, выделим следующие:

- атомная промышленность (включая ядерно-оружейный комплекс);
- сложные промышленные системы, прежде всего, в авиационной и космической отрасли;
- энергетика и энергосбережение;
- медицина;
- биология;
- информационные технологии.

В США в 2009 году по указанию президента США с целью сохранения лидирующих позиций в области суперкомпьютерных технологий, их внедрению в науку, промышленное производство и социальные сферы разработан план действий под названием *Exascale Initiative* [54]. Министерством энергетики США из специалистов ядерных лабораторий создан специальный комитет *Exascale Initiative Steering Committee* (*EISC*), целями которого являются:

- определение особо значимых научных проблем, требующих наибольшего привлечения вычислительных ресурсов в течение следующего десятилетия;

• определение оптимального пути повышения производительности вновь создаваемых вычислительных систем с учётом потребностей прикладной и фундаментальной науки;

• организация обмена междисциплинарными идеями в интересах развития суперкомпьютерной базы.

Объем ежегодного финансирования исследовательских работ по этому эксафлопсному проекту превышает $100 млн.

Кроме того, министерство энергетики США является одним из основных финансистов международного проекта *International Exascale Software Project (IESP)* [55]; соучредителем проекта является также национальный научный фонд (*NSF*) США. Цель проекта – создание программной базы для суперкомпьютеров эксафлопсного уровня, появление которых ожидается в 2018-2020 г.г. В проекте, кроме американских, широко задействованы европейские и японские фирмы и организации.

Аналогичные работы, предусматривающие достижение эксафлопсной производительности, выполняются под патронажем министерства обороны США через агентство *DARPA* [56] – в частности, укажем проекты *HPCS, UHPC* и *OHPC*. В них объединены ведущие фирмы – *Intel, Cray, Nvidia*, университеты и национальные лаборатории.

Создание систем производительностью 10 Пфлоп/с предусматривается в США уже с 2012 года. В Европе эксафлопсные технологии реализуются в проекте PRACE (он предусматривает появление эксафлопсного компьютера в 2019 году) [57] и финансируемых Евросоюзом проектах *European Exascale Software Initiative (EESI)* и *Towards EXascale ApplicaTions (TEXT)*, направленных на разработку программных технологий, предназначенных для использования в эксаскальных компьютерах.

Китай планы не публикует, но, по мнению экспертов, способен создать суперЭВМ эксафлопсного класса на собственной элементной базе в 2020 году.

Вышеизложенное свидетельствует об интенсивных усилиях ведущих государств мира по созданию эксафлопсных технологий.

Применение эксафлопсных технологий, реализующих принципиально различные дисциплины вычислений на качественно более высоком уровне параллелизма, сложности и неоднородности вычислительных систем требует:

- фундаментальных исследований математических методов, алгоритмов и архитектур, позволяющих достигнуть необходимые значения производительности;
- разработки аппаратных и системных программных средств экспериментальных вычислительных систем, удовлетворяющих требованиям по энергопотреблению, надежности и другим физическим параметрам;
- создания соответствующего прикладного программного обеспечения.

2.2. Основные проблемы создания микропроцессоров для суперЭВМ эксафлопсного класса.

Микропроцессоры требуемой области применения должны содержать десятки ядер и иметь производительность не менее единиц Тфлопс. Достижение таких производительностей требует использование предельных технологических норм изготовления кристаллов и частот их функционирования. На этом пути стоят 2 основные проблемы: **низкая надёжность и высокая потребляемая мощность** микросхем и соответственно всего компьютера. Необходимость эффективного функционирования $10^7 – 10^8$ процессорных ядер такой суперЭВМ и преодоление физических ограничений, обусловленных энергопотреблением, надежностью и конструктивными размерами предполагают иные подходы к архитектуре, схемотехнике и программному обеспечению.

При использовании предельных технологических норм изготовления кремниевых пластин, необходимых для создания эксафлопсных суперЭВМ, невозможно добиться высокой надёжности функционирования микросхем без использования всего комплекса мер повышения надёжности. Это обусловлено:

• Снижением напряжения питания и соответствующим уменьшением возможного разброса порогов логических уровней, повышением требований к качеству питания.

• Уменьшением толщин диэлектриков и соответствующим снижением надёжности, повышением токов утечки, снижением величин паразитных зарядов (порогов), приводящим к сбою или отказу микросхем.

• Уменьшением сечений проводников, что приводит к уменьшению длительности функционирования микросхем за счёт эффекта электромиграции, снижением механической прочности, прежде всего, при снижении температур окружающей среды до предельно низких значений с последующим повышением до предельно высоких.

• Большим количеством элементов, статистически приводящим к повышению вероятности сбоя микросхем.

• Увеличением поражающего фактора для одиночных и локальных радиационных эффектов, связанного с существенно меньшими топологическими размерами активных элементов в сравнении с размерами трека поражающей частицы.

• Существенное влияние фактора микродозовых радиационных эффектов на надежность и долговечность схем, обусловленное структурными изменениями в приборных областях схем при воздействии высокоэнергетичных частиц и потоков атмосферных нейтронов.

Достижения требуемой надёжности функционирования микросхем возможно при применении комплекса мер

• конструктивно-технологических,

• схемотехнических,

• программных и микропрограммных.

К **технологическим мерам** следует отнести использование специальных процессов изготовления кристаллов (кремниевых пластин). Так, компания IBM идёт по пути развития процесса КНИ, приводящего к снижению потребляемой мощности и повышающего рабочую частоту до 30 %.

Следующим фактором, приводящим к существенному повышению надёжности, является создание специальных **радиационно-стойких**

библиотек. Работы компании Боинг показали, что даже при использовании КМОП технологий при наличии таких библиотек возможно достижение стойкости на уровне нескольких сотен Крад (при использовании КНИ – единицы Мрад). Следующими конструктивными решениями повышения надёжности микросхем является разработка правил топологического проектирования, учитывающие требования по защитным кольцам, дополнительным контактам, сетке земли и питания, клоковому дереву, внетрековому (для ионизирующих частиц) разнесению активных элементов и пр. Факторы радиационной стойкости начнут неизбежно проявляться при большом числе микросхем даже в обычные дни, а при наличии солнечных вспышек работа суперЭВМ может полностью блокироваться в силу огромного числа сбоев [58]. Необходимо также отметить, что начало тиристорного эффекта в КМОП схемах можно отследить резким повышением тока в месте прохождения высокоэнергетичной частицы (в КНИ КМОП схемах тиристорный эффект практически отсутствует [58]). Основные радиационные эффекты в микросхемах и методы их уменьшения показаны в табл. 2.1.

Таблица 2.1. Основные радиационные эффекты в СБИС и методы их уменьшения

Название	Причина	Повреждения	Частота возникновения	Решения	Эффективность решений
SEU – одиночные сбои	Заряд, генерируемый в объеме приборного слоя полупроводника ядерной частицей	Одиночный сбой	Доминирует при проектных нормах более 0,18 мкм	КНИ технология, КМОП с эпитаксией	Высокая
				Сбоеустойчивые ячейки (DICE)	Высокая
				Резервирование	Средняя
				Помехоустойчивое кодирование	Высокая
SEL – тиристорный эффект	Активация паразитных тиристорных структур	Сбои, критический отказ	Доминирует при проектных нормах менее 0,25 мкм	КНИ технология, КМОП с эпитаксией	высокая
				Контакты к подложке и n-карману	средняя
				Охранные кольца	средняя
				Детектирование тиристорного эффекта и кратковременный сброс питания	низкая

Название	Причина	Повреждения	Частота возникновения	Решения	Эффективность решений
SET – функциональные сбои	Заряд, генерируемый в объеме приборного слоя полупроводника ядерной частицей	Сбои в комбинационной логике	средняя	КНИ технология, КМОП с эпитаксией	высокая
				Дополнительные контрольные цепи	высокая
TID – накопленная доза	Накопление заряда в подзатворном и захороненном окисле	Сбои, некорректная работа, высокие токи утечки	Зависит от времени экспозиции и флюенса.	ELT – транзисторы с кольцевым затвором	высокая
				Транзисторы с увеличенной длиной канала паразитных транзисторов	низкая
MU – множественные сбои	Вызываются зарядом от отдельной частицы, попавшей под углом к кристаллу	Множественные сбои	Низкая при проектных нормах более 0,25 мкм.	Помехоустойчивое кодирование	средняя
				КНИ технология, КМОП с эпитаксией	высокая
				Сбоеустойчивые ячейки, растянутые в пространстве	неизвестно

Схемотехнические приёмы повышения надёжности микросхем следует разбить на 2 группы: создание высоконадёжной памяти и создание надежных вычислительных и управляющих блоков. Это связано с различными способами реализации таких схем и различными методиками повышения их надёжности. В свою очередь методики создания высоконадёжной памяти следует разбить на 2 группы: создание быстродействующего ОЗУ небольшого объёма (до нескольких десятков Кбайт), прежде всего, Кэш памяти 1 уровня, и ОЗУ объёмом свыше 100 Кбайт. Это связано с тем, что для небольшого ОЗУ можно применить дополнительные схемотехнические решения, приводящие к повышению надёжности, но и увеличивающие площадь ОЗУ до нескольких раз. Для ОЗУ большого объема необходимо применение элементов и блоков помехоустойчивого кодирования, обеспечивающих контроль и исправление не только одиночных, но и множественных ошибок.

Повышение надёжности вычислительных узлов с использованием схемотехнических решений также возможно за счёт применения различного рода дублирований и троирований. Возможно также использования многофазной специальной логики, функционирующей аналогично коду Хэмминга для памяти, позволяющей определить сбой при выполнении арифметической операции.

Ещё один вариант – использование асинхронной или строго самосинхронной логики. Однако все коммерческие САПР проектирования рассчитаны на синхронную логику. Необходимы большие вложения в создание инструментария разработки таких микросхем. Возможным вариантом является первоначальное создание проекта на синхронной логике с последующим его переводом на строго самосинронную логику.

Для схем управления возможно использование кодирования долгоживущих регистров с помощью *ECC* и контроль корректности машин состояния.

Дальнейшим повышением надёжности является **создание встроенного тестирования**, включая тестирование на аппаратном и микропрограммном уровнях. И, наконец, предлагается предоставление максимальных возможностей для тестирования на **программном уровне**, сбора статистики по сбоям, возможностям отключения/включения дополнительных ядер микропроцессора и изменения частоты его функционирования в процессе выполнения программы.

Для создания требуемых микропроцессоров можно выделить следующий комплекс мер по повышению их надёжности.

1. Разработка радиационно-стойкой библиотеки.
2. Разработка правил топологического проектирования.
3. Создание датчиков тока и температуры для каждого функционального блока с целью выявления нештатных ситуаций косвенными методами и принятия мер по их ликвидации на аппаратном и программном уровнях.
4. Создание ОЗУ ёмкостью до нескольких десятков Кбайт на базе ячеек Dice с разнесением половинок ячеек памяти в шахматном порядке и перемежением данных с целью уменьшения вероятности сбоя обоих половинок одновременно.
5. Создание ОЗУ ёмкостью сотни Кбайт – единицы Мегабайт с кодом Хэмминга (или модифицированными кодами, например, Хсяо).

Встроенное помехоустойчивое кодирование данных должно учитывать факторы образования множественных ошибок и сбоев.

6. Дублирование вычислительных узлов (ядер). Ядра могут функционировать или независимо, или в режиме дублирования. В режиме дублирования при отличии результатов выполнения инструкций производится её выполнение заново. При повторной (третьей) фиксации отличия производится внутреннее тестирование для выявления отказавшего ядра и исключения его из процесса вычислений.

7. Схемы контроля дублирования и часть схем управления делаются на более сбоеустойчивых радиационно-стойких элементах (например, кольцевых транзисторах), используются также кодирование долгоживущих регистров с помощью *ECC* и контроль корректности машин состояния.

8. Время выполнения отдельных частей программы, включая тестовые программы, контролируется сторожевыми таймерами. Должно быть достаточное количество таких таймеров с различным временным диапазоном и точностями.

9. Сетки земли и питания для ядер делаются разными.

10. Статистика по сбоям накапливается в специальных регистрах с возможностью выдачи этой информации, например, в локальную сеть Ethernet.

11. Создаются аппаратные возможности внутреннего самотестирования функциональных блоков и, прежде всего, памяти.

12. Имеется несколько независимых ОЗУ для групп ядер с контролем по коду Хэмминга.

13. Каждая группа ядер процессора имеет собственные коммуникационные каналы. Каналы имеют внутренние средства самотестирования и могут отключаться аналогично ядрам микропроцессора.

14. Предусматривается возможность раздельного питания ядер.

15. Предусматривается возможность дублирования генератора микропроцессора.

16. Осуществляется постепенный переход на логику, приводящую к снижению потребления питания и повышению надёжности (например, строго самосинхронную логику).

17. Создаются программные средства тестирования узлов микропроцессора (прежде всего, памяти) в режимах ожидания.

18. Создаётся система сбора статистики сбоев и отказов с выходом, например, на локальную сеть.

2.3. Основные типы архитектур параллельных ЭВМ

Архитектуры универсальных параллельных компьютеров можно классифицировать по параметру, характеризующему степень связности базовых вычислительных устройств системы друг с другом. Параметр степени связности определяется способом размещения памяти в системе и скоростью обмена информацией между различными устройствами, а также между устройствами и подсистемой памяти. По этому признаку выделяют три основных класса архитектур [59,60]:

1. симметричные мультипроцессоры *(SMP - Symmetric Multi-Processors*) - архитектуры с разделяемой памятью (*SM - Shared Memory*);
2. массивно-параллельные системы (*MPP - Massive Parallel Processors*) - архитектуры с разделенной памятью (*DM - Distributed Memory*);
3. архитектура с неоднородным доступом к памяти (*NUMA - Non Uniform Memory Access*), нечто среднее между *SMP* и *MPP*, где память физически разделена, но логически общедоступна.

Более дешевым вариантом *MPP* являются кластерные архитектуры: кластеры, как правило, создаются из компонент высокой степени готовности, и для объединения вычислительных модулей в кластерах используются менее производительные коммуникационные среды [61].

2.3.1. Симметричные мультипроцессорные системы

Симметричные мультипроцессорные системы (*SMP*) состоят из совокупности однородных процессоров и массива общей памяти (см. рис. 2.1). Все процессоры обладают одинаковыми возможностями доступа к памяти и внешним устройствам. Процессоры подключены к памяти либо с помощью общей шины (например, *Power Challenge* фирмы *Silicon Graphics*; многопроцессорные серверы на базе процессоров *Intel*), либо с помощью коммутатора с пространственным разделением (например, модули-узлы систем *HP9000 V-*

class Enterprise Servers [62], *SUN Ultra Enterprise* 10000 [63], *SGI Origin* 2000 [64], *NEC SX*-5 [65]). Аппаратно поддерживается когерентность КЭШ памятей. Наличие общей памяти упрощает взаимодействие процессоров между собой, однако накладывает сильные ограничения на их число - не более 64 в реальных системах (например, *SMP*-узел в системе *HP*9000 *V*-class Enterprise Server может содержать до 32 процессоров). Для построения масштабируемых систем на базе SMP используются кластерные или *NUMA*- архитектуры.

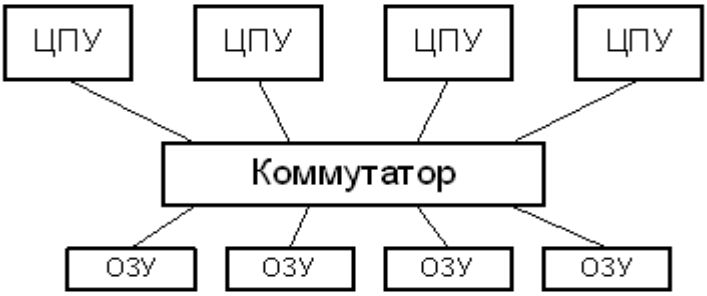


Рис. 2.1. Симметричная многопроцессорная архитектура.

2.3.2. Системы с неоднородным доступом к памяти

Системы с неоднородным доступом к памяти (*NUMA*) состоят из однородных базовых модулей (плат), объединенных с помощью высокоскоростного коммутатора (см. рис. 2.2). Базовый модуль представляет собой *SMP*-архитектуру. Каждый модуль имеет специальный порт, который является прямым расширением внутримодульной коммуникационной среды, через который модули могут быть объединены друг с другом напрямую, либо через внешний коммутатор. Среда специализирована для работы только с определенным видом модулей, системный контроллер которых (или процессоры) имеют соответствующий интерфейс связи с внешней коммуникационной средой. Во всей системе поддерживается единое адресное пространство и аппаратно поддерживается доступ к удаленной памяти, т. е. памяти других модулей. Доступ к локальной памяти, при этом, осуществляется в несколько раз быстрее, чем к удаленной па-

мяти. То есть время доступа к памяти определяется её расположением (близостью) к процессору. В случае если аппаратно поддерживается когерентность КЭШ памятей во всей системе, то говорят об архитектуре *cc-NUMA* (*cashe coherent NUMA*). *NUMA*-системы характеризует большая, по сравнению с *MPP*-системами, скорость обмена информацией между узлами, но меньшая степень масштабируемости. Масштабируемость *NUMA*-систем ограничивается объемом адресного пространства, возможностями аппаратуры поддержки когерентности КЭШ памятей и возможностями операционной системы по управлению большим количеством процессоров. Максимальное число узлов в *NUMA*-системах не превышает нескольких сотен (например, в системе *SGI Origin* 2000 может содержаться до 512 узлов, хотя реальные системы содержат меньше).

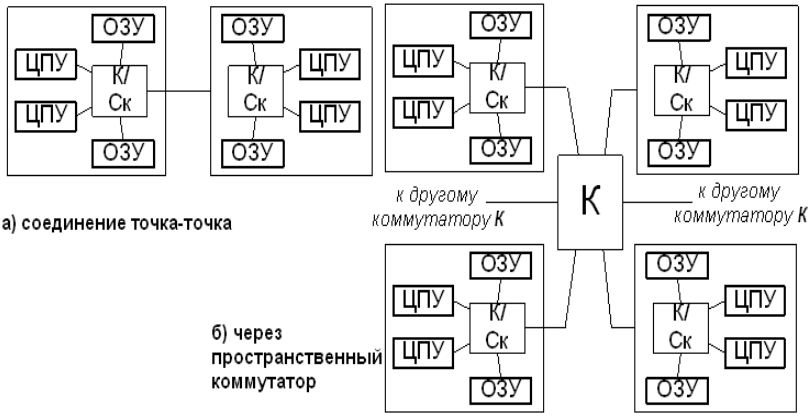


Рис. 2.2. Архитектуры с неоднородным доступом к памяти.

2.3.3. Системы с массовым параллелизмом

Системы с массовым параллелизмом (*MPP*) состоят из однородных вычислительных узлов, включающих один или несколько центральных процессоров, локальную память, коммуникационный контроллер или сетевой адаптер, жесткие диски и другие устройства ввода/вывода (см. рис. 2.3). К системе могут добавляться специальные узлы ввода/вывода, дисковые хранилища данных (например,

RAID – массивы) и управляющие узлы (например, управляющий сервер или рабочая станция). Узлы связаны между собой через коммуникационную среду.

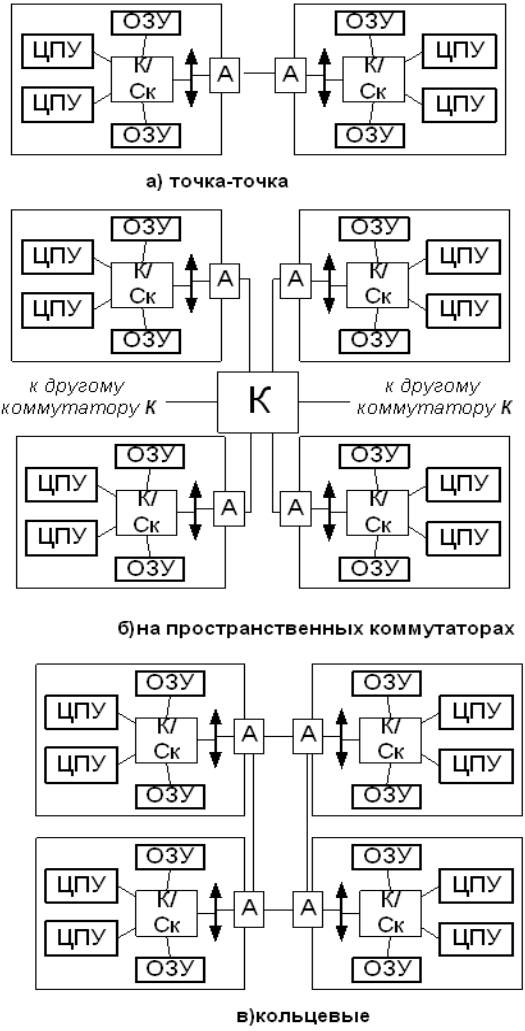


Рис. 2.3. Архитектуры с массовым параллелизмом. А – адаптер.

С учётом того, что в таких системах для организации обмена данными требуется больше дополнительных команд, чем для *SMP* и *NUMA* систем, требования к коммуникационным системам существенно возрастают. Характерные требования к системе связи *MPP*: высокая пропускная способность, маленькая задержка на соединения точка-точка и коммутацию, возможность совмещения по времени передач данных через среду с вычислениями в модулях, базирование на стандартах, надежные протоколы с управлением потоком, поддержка различных топологий, масштабируемость, конфигурируемость. *MPP*-системы обладают высокой степенью масштабируемости, количество узлов может достигать нескольких десятков тысяч. На сегодняшний день наиболее высокопроизводительные в мире компьютеры представляют собой *MPP*-архитектуры.

Архитектуры типа *MPP* являются наиболее производительными, так как они обладают наибольшими возможностями масштабируемости и на их основе можно строить параллельные ЭВМ с большим количеством вычислительных узлов, чем для других типов архитектур.

Второе преимущество архитектур *MPP* в том, что они могут быть построены, как правило, путем объединения стандартных вычислительных модулей друг с другом посредством некоторой коммуникационной среды. При этом отпадает необходимость разработки специализированного вычислительного блока, которая возникает в системах с архитектурой NUMA или *SMP*. На используемые вычислительные модули накладывается лишь два ограничения:

1. наличие одной из стандартных шин устройств ввода-вывода, для которых разработаны адаптеры коммуникационной среды;

2. способность работать под операционной системой, для которой разработано программное обеспечение, обеспечивающее функционирование коммуникационной среды.

В большинстве случаев, это системы модули с шиной *PCIe* и работающие под ОС типа *UNIX*. При построении параллельных ЭВМ типа систем с массовым параллелизмом (*MPP*) часто используются коммуникационные среды, выполненные по схеме: Адаптер может также присоединяться к системной шине. Например, коммуникационные контроллеры *NIC - Network Interface Controller* в системе *Intel ASCI Red* находятся на системной шине двухпроцессорного модуля

[66]), или шине устройств ввода/вывода вычислительного модуля (например, адаптеры *QSW ELAN* фирмы *Quadrics Supercomputer World* присоединяются к шине *PCI* 64/66 [67]), или к специальной шине, разработанной под адаптер (как это сделано в системе *IBM SP2 RS*/6000, где в некоторых модулях коммуникационные адаптеры расположены на специальной мезонинной шине *MX-bus*.

Дополнительным преимуществом архитектуры *MPP* является возможность построения систем с повышенной отказоустойчивостью за счет дублирования коммуникационной среды, объединяющей вычислительные узлы. Резервная коммуникационная среда может создаваться на независимом наборе адаптеров и коммутаторов, которые будут дублировать работу основных коммуникационных устройств при их отказе.

2.3.4. Кластерные системы

Один из создателей кластерной технологии Грегори Пфистер дал кластеру следующее определение: «Кластер — это разновидность параллельной или распределённой системы, которая:

1. состоит из нескольких связанных между собой компьютеров;
2. используется как единый, унифицированный компьютерный ресурс».

Кластерные системы являются более дешевым вариантом массово-параллельных систем (*MPP*). Узлами кластера могут быть рабочие станции или персональные ЭВМ общего назначения, которые одновременно могут использоваться и как узлы кластера, и как персональные компьютеры. В случае, когда это не нужно, они могут быть существенно облегчены и установлены наборами отдельных плат в специальные стойки. Для связи узлов используется одна из стандартных сетевых технологий (*Ethernet*, *Myrinet* и др.) на базе шинной архитектуры или коммутатора.

Кластерные системы принято классифицировать как кластеры высокой доступности (*High-availability clusters*), с балансировкой нагрузки (*Load balancing clusters*), вычислительные (*High performance computing clusters*) и Грид (*grid*) кластеры распределённых вычислений.

Кластеры высокой доступности создаются для обеспечения высокой доступности сервиса в течение заданного промежутка времени. Доступность означает возможность работы группе пользователей на кластере в течение этого времени. Для решения проблемы гарантированного доступа создаётся избыточность в критических узлах, таких как маршрутизатор. Тогда отказ одного компонента не вызовет сбоя приложения, то есть программа закончит свою работу. В случае маршрутизатора предполагается, что имеются резервные пути прохождения данных из одного вычислительного узла в другой. Имеется 3 основных принципа построения таких кластеров:

- с холодным резервированием, когда активный узел работает, а пассивный начинает функционирование только в случае отказа активного;
- горячим резервированием, когда все узлы функционируют и в случае отказа одного узла происходит перераспределение задачи между оставшимися;
- с модульной избыточностью; применяется, когда простой системы недопустим; в этом случае резервируемые узлы выполняют один и тот же запрос и в случае выхода из строя одного из узла результат берётся из резервного.

Кластеры с балансировкой нагрузки предполагают распределения процесса выполнения заданий между несколькими серверами сети с целью оптимизации использования ресурсов и сокращения времени вычислений.

Основная функция **вычислительных кластеров** добиться высоких показателей производительности, прежде всего над числами с плавающей запятой. Задача для таких систем разбивается на параллельно выполняемые части, обмен результатов между которыми выполняется по сети.

Грид системами являются слабосвязанные гетерогенные (разнородные) системы, объединяющие множество ресурсов разных типов, доступ к которым пользователь может получить из любой точки, и выполняющих огромное количество заданий. Грид системы основываются на обычных неспециализированных компьютерах со

стандартными периферийными устройствами соединённых в сеть при помощи стандартных протоколов, как правило, это сеть *Ethernet*. В такой системе невозможно гарантировать работу отдельного узла в каждый момент времени, поскольку узлы подключаются и отключаются в процессе работы. Нестабильность конфигурации компенсируется большим числом вычислительных узлов.

Примерами кластеров на основе дешёвых персональных компьютеров, объединённых сетью передачи данных, служат кластеры *Beowulf*, специально разработанные на основе этого принципа. Успехи таких систем подтолкнули развитие Грид сетей, которые существовали ещё с момента создания операционной системы *UNIX*.

2.3.5. Неоднородные системы

По объему требуемых затрат ресурсов на объединение того же количества вычислительных узлов различные системы в порядке убывания затрат выстраиваются следующим образом: *SMP, NUMA, MPP,* кластер. В случаях, когда параллельная ЭВМ предназначена для решения задач, в которых не требуются одинаковые производительности коммуникационных взаимодействий между всеми блоками системы, более эффективным является использование неоднородной архитектуры. Полученная система приобретает неоднородную иерархическую структуру с разными степенями связности на различных ступенях иерархии.

Примером сильносвязанной распределенной системы с двухуровневой организацией коммуникационной среды являются компьютеры семейства *HP9000 V-class Enterprise Server* фирмы *Hewlett Packard*. В основе сервера *V*-класса лежит архитектура *SCA (Scalable Computing Architecture)*, которая позволяет создавать двухуровневые *ccNUMA*. Архитектура *SCA* включает в себя двухуровневую коммуникационную среду: на первом уровне используется коммутатор *Hyperplane*, формирующий гиперузел (*cabinet*) архитектуры *SMP*, на втором уровне используется многоканальная двухмерная кольцевая магистраль *SCA Hyperlink*, с помощью которой гиперузлы объединяются в единую систему архитектуры *cc-NUMA*. На втором уровне коммуникационная среда представлена соединениями *SCA*

Hyperlink, с помощью которых можно объединить до 4-х гиперузлов, при этом максимальное количество процессоров в системе достигает 128.

Обобщенная схема построения неоднородной параллельной ЭВМ такова: на нижнем слое система состоит, как правило, из *SMP*-узлов, с помощью высокопроизводительных магистралей, объединенных в *NUMA*-гиперузлы, которые через коммуникационную среду более высокого слоя образуют *MPP*-системы, которые, в свою очередь, могут быть объединены локальной вычислительной сетью в кластер. Частная форма реализации приведенной обобщенной схемы зависит от требований, предъявляемых к конкретной создаваемой системе, и от объема капиталовложений и иных ресурсов, которые могут быть потрачены на создание конкретной системы.

Глава 3. Программное обеспечение параллельных компьютеров.

На эффективность использования вычислительной машины при решении прикладных задач очень сильное влияние оказывает применяемое программное обеспечение. На верхнем уровне программное обеспечение представлено двумя моментами: с одной стороны, - это параллельные языки высокого уровня или параллельные расширения языков высокого уровня, на которых пишется параллельная программа для абстрактной параллельной машины; с другой, - система управления ресурсами конкретной вычислительной машины, в рамках которой производится выделение ресурсов под выполнение программы, сегментация и разложение задачи в соответствии с доступными ресурсами. На нижнем уровне программное обеспечение предоставляет системные средства поддержки параллельности, а именно: средства взаимодействия параллельно выполняющихся процессов и средства работы с коммуникационными и сетевыми интерфейсами. Программное обеспечение как верхнего, так и нижнего уровня может быть унифицированным и переносимым на различные системы, и может быть специализированным и предназначенным для использования на конкретной платформе. Во втором случае достигается более высокая производительность и эффективность использования особенностей данной параллельной архитектуры.

Одной из основных задач в области прикладного параллельного программирования является накопление опыта переносимости параллельных программ. В рамках концепции открытых систем и переносимости ПО с платформы на платформу вводятся различные стандартизированные парадигмы представления параллельных вычислений, такие как *PVM* [68] (*Parallel Virtual Machine* - параллельная виртуальная машина), *MPI* [69] (*Message Passing Interface* - интерфейс передачи сообщений) и *OpenMP* [70] (интерфейс прикладных программ для разделяемой памяти). Как в *MPI*, так и в *OpenMP* основной язык не фиксируется и эти средства могут расширять любой компилятор. Наиболее часто используются языки программирования Fortran и *C/C++*.

3.1. Система управления ресурсами.

Система управления ресурсами параллельной машины кроме обеспечения всех стандартных механизмов управления ресурсами однопроцессорной машины должна также включать в себя дополнительные механизмы взаимодействия и управления удаленными ресурсами. Сюда входят средства управления доступом, системное администрирование, управление распределенной файловой системой и устройствами ввода/вывода, механизмы синхронизации работы ресурсов, средства отладки параллельных приложений. В нее также включается непосредственно *run-time* система управления процессами, отвечающая за создание процессов и планирование (*scheduling*) выполнения процессов на распределенных ресурсах.

Параллельные вычисления могут выполняться как в рамках модели *SPMD* (*Single Program Multiple Data* - одна программа для различных данных), так и в рамках модели *MPMD* (*Multiple Program Multiple Data* - различные программы, различные данные). В первом случае на разных вычислительных модулях выполняется одна и та же программа, но данные, поступающие на вход программы, неодинаковые. Во втором случае на разных вычислительных модулях выполняются различные программы.

3.1.1. Модель *SPMD*.

В случае модели SPMD система производит одновременный запуск необходимого (в рамках доступных ресурсов) количества копий программы на различных модулях, передает каждой копии программы свои параметры и инициирует работу средств межпроцессных взаимодействий. В качестве примера рассмотрим механизм планирования процессов для модели *SPMD*, который используется в суперЭВМ компании *Quadrics Supercomputer Worlds* [71].

Наборы процессоров в системе называются партициями (*partitions*). Партиции представляют собой логически независимые виртуальные машины в рамках одной физической системы. Для каждой партиции существует структура управления, называемая мене-

джером партиции (*partition manager*). Менеджер партиции управляет распределением ресурсов, связанных с данной партицией, и отвечает за планирование процессов в рамках партиции.

Параллельное приложение представляет собой набор из одного и более сегментов и выполняется в рамках одной партиции. Сегмент – это совокупность одинаковых процессов, выполняющихся на смежных (соседних) вычислительных узлах системы, при этом все процессы в сегменте выполняют одинаковую программу. Все процессы во всех сегментах параллельной программы запускаются одновременно. Каждому процессу присваивается уникальный индивидуальный номер виртуального процесса в непрерывном диапазоне начиная с 0. Адресное пространство каждого процесса доступно для всех соответствующих ему процессов (*peers*) через сеть. К нескольким процессам со смежными номерами можно обратиться одновременно, через соответствующий широковещательный номер виртуального процесса. При обращении к этому номеру в среде выполняются аппаратные широковещательные пересылки.

Набор процессов в рамках одной программы может принадлежать не только сегменту, но и группе - специальному глобальному объекту. Группы используются для объединения произвольных наборов процессов параллельной программы, включая несмежные и нерегулярные наборы. Группы могут использовать различные глобальные групповые функции, предоставляемые библиотекой коммуникационного интерфейса, включая широковещательные пересылки, барьеры, сборку и глобальные обмены.

Процессы внутри одного сегмента выполняют одинаковый программный код, поэтому размещение глобальных объектов во всех процессах может производиться по одинаковым виртуальным адресам на этапе компиляции, если определять их как статические переменные. Обмен данными между глобальными объектами процессов, выполняемых на разных узлах, может производиться с помощью простых команд прямого доступа к удаленной памяти по заранее известным виртуальным адресам. Коммуникационный интерфейс обеспечивает выполнение таких функций без вовлечения операционной системы.

Системные вызовы, появляющиеся при выполнении процесса, обрабатываются локально на узле, где процесс выполняется. Также

некоторые системные вызовы могут перенаправляться к удаленным узлам-серверам.

3.1.2. Модель *MPMD*.

В рамках *MPMD* обычно реализуется клиент-серверный механизм взаимодействия программ. В качестве примера можно привести прикладной программный интерфейс высокого уровня для сетевых взаимодействий *RPC* (*Remote Procedure Call* - удаленный вызов процедур), на базе которого, в частности, разработана файловая система *NFS* [72]. В случае удаленного вызова процесс, выполняющийся на одном вычислительном узле, запускает процесс на удаленном вычислительном узле (т. е. фактически запускает код процедуры на удаленном узле). Удаленный вызов существенным образом отличается от традиционного локального: если в случае локального вызова программа передает параметры в вызываемую процедуру и получает результат обработки через стек или общие области памяти, то в случае удаленного вызова передача параметров превращается в передачу запроса по сети, а результат работы находится в пришедшем отклике. Данный подход является возможной основой создания распределенных приложений, и, хотя многие современные системы не используют этот механизм, основные концепции и термины во многих случаях сохраняются. Вызывающий процесс традиционно называется клиентом, а удаленный процесс, реализующий процедуру, - сервером. Удаленный вызов включает следующие шаги:

1. Программа-клиент производит локальный вызов процедуры, называемой заглушкой (*stub*). Задача заглушки - принять аргументы, предназначенные удаленной процедуре, возможно, преобразовать их в некий стандартный формат и сформировать сетевой запрос. Упаковка аргументов и создание сетевого запроса называется сборкой (*marshalling*).

2. Запрос посылается по коммуникационной среде на удаленную систему.

3. Заглушка сервера ожидает запрос и при получении извлекает параметры - аргументы вызова процедуры. Извлечение (*unmarshalling*) может включать необходимые преобразования (например, перестановку байтов для некоторых гетерогенных

систем).

4. Заглушка выполняет вызов настоящей процедуры-сервера, которой адресован запрос клиента, передавая ей полученные по сети аргументы.

5. После завершения выполнения процедуры управление возвращается в заглушку сервера, передавая ей требуемые параметры. Как и заглушка клиента, заглушка сервера преобразует возвращенные процедурой значения, формируя сетевое сообщение-отклик, который передается по коммуникационной среде узлу, от которого пришел запрос.

6. Операционная система передает полученное сообщение заглушке клиента, которая, после необходимого преобразования, передает значения, возвращенные удаленной процедурой, клиенту, воспринимающему это как нормальный возврат из процедуры.

Заглушки составляют ядро системы *RPC*, отвечая за все аспекты формирования и передачи сообщений между клиентом и удаленным сервером. Основная концепция *RPC* состоит в том, чтобы полностью спрятать распределенный характер взаимодействия в коде заглушек. И клиент, и сервер в этом случае являются независимыми от сетевой реализации, оба они работают в рамках некой распределенной виртуальной машины, и вызовы процедур имеют стандартный интерфейс.

3.2. Межпроцессные взаимодействия

После запуска программы и выделения вычислительных ресурсов на ее выполнение необходимо организовать механизм межпроцессного взаимодействия и обмена данными в рамках единой распределенной среды. В большинстве современных операционных систем процессы изолированы друг от друга: каждый процесс выполняется в своем процессорном контексте и в своем виртуальном адресном пространстве. Тем самым сведены к минимуму возможности влияния процессов друг на друга, что является необходимым в многозадачных операционных системах. Для взаимодействия процессов необходимо вводить специальные механизмы IPC (Inter-Process Communications, межпроцессные взаимодействия) в рамках

операционной системы. Взаимодействие между процессами необходимо для решения следующих задач:

• Передача данных. Один процесс передает данные другому процессу, при этом их объем может варьироваться от десятков байт до нескольких мегабайт;

• Совместное использование данных. Вместо копирования информации от одного процесса к другому, процессы могут использовать одну копию данных, причем изменения, сделанные одним процессом, будут сразу же заметны для другого. Количество взаимодействующих процессов может быть больше двух. При совместном использовании ресурсов процессам может понадобиться некоторый протокол взаимодействия для сохранения целостности данных и исключения конфликтов при доступе к ним.

• Извещения. Процесс может известить другой процесс или группу процессов о наступлении некоторого события. Это может понадобиться, например, для синхронизации выполнения нескольких процессов.

3.2.1. Стандартные средства поддержки межпроцессных взаимодействий в ОС *UNIX*

К средствам межпроцессного взаимодействия, присутствующим во всех версиях *UNIX*, можно отнести: сигналы, каналы, *FIFO* (именованные каналы), сообщения (очереди сообщений), семафоры, разделяемую память [73]. Последние три типа IPC обычно называют *System V IPC*. Во многих версиях *UNIX* есть еще одно средство *IPC* - сокеты, впервые предложенные в *BSD UNIX*. Для вопроса построения распределенных вычислений наибольший интерес из перечисленных механизмов представляют средства работы с разделяемой памятью и интерфейс сокетов. Эти механизмы применимы как для симметричных многопроцессорных систем с разделяемой памятью, так и для распределенных систем.

Разделяемая память.

Интенсивный обмен данными между процессами с использованием таких *IPC*, как каналы, *FIFO* и очереди сообщений, может вызвать падение производительности системы. Это, в первую очередь,

связано с тем, что данные, передаваемые с помощью этих объектов, копируются из буфера передающего процесса в буфер ядра и затем в буфер принимающего процесса. Механизм разделяемой памяти позволяет избавиться от накладных расходов передачи данных через ядро, предоставляя двум или более процессам возможность непосредственного получения доступа к одной области памяти для обмена данными. Задача кооперативного использования разделяемой памяти, заключающаяся в синхронизации выполнения процессов, решается с помощью семафоров. Примерный сценарий работы с разделяемой памятью выглядит следующим образом:

1. Сервер получает доступ к разделяемой памяти, используя семафор.
2. Сервер производит запись данных в разделяемую память.
3. После завершения записи сервер освобождает разделяемую память с помощью семафора.
4. Клиент получает доступ к разделяемой памяти, запирая ресурс с помощью семафора.
5. Клиент производит чтение данных из разделяемой памяти и освобождает ее.

Разделяемая память является наиболее быстрым способом передачи данных между неродственными процессами. Разделяемая память является частью адресного пространства для каждого из взаимодействующих процессов, поэтому чтение и запись в эту область неотличимы от чтения и записи в область собственных данных процесса, в случае, если данные располагаются в локальной физической памяти.

Сокеты.

Механизм межпроцессного взаимодействия, получивший название сокеты, позволяет взаимодействовать процессам унифицированным образом, независимо от того, выполняются ли они на одном узле или на разных узлах сети. Наиболее оптимальная реализация межпроцессного взаимодействия, удовлетворяющего этому требованию, должна иметь модульную структуру и базироваться на общей подсистеме поддержки сети *UNIX*. При этом могут быть использованы различные схемы адресации объектов, их расположение, протоколы передачи данных и т.д. В этой связи было введено

понятие коммуникационный домен (*communication domain*), описывающее набор обозначенных характеристик взаимодействия. В *BSD UNIX* реализованы следующие основные типы сокетов:

- Сокет датаграмм (*datagram socket*), через который осуществляется теоретически ненадежная, несвязная передача пакетов.
- Сокет потока (*stream socket*), через который осуществляется надежная передача потока байтов без сохранения границ сообщений.
- Сокет пакетов (*packet socket*), через который осуществляется надежная последовательная передача данных без дублирования с предварительным установлением связи. При этом сохраняются границы сообщений.
- Сокет низкого уровня (*raw socket*), через который осуществляется непосредственный доступ к коммуникационному протоколу.

Для создания сокета используется системный вызов

int socket (int domain, int type, int protocol);

Здесь аргумент domain определяет коммуникационный домен, type - тип сокета из перечисленных выше, protocol - используемый протокол. Коммуникационный домен определяет семейство протоколов, допустимых в рамках данного домена. Домен может быть доменом локального межпроцессного взаимодействия в пределах единой операционной системы *UNIX (AF_UNIX)* или доменом взаимодействия удаленных систем (*AF_INET, AF_NS* и т.д.). Для одного домена могут использоваться различные виды протоколов, например, для домена *AF_INET* аргумент protocol может принимать значения *IPPROTO_TCP (TCP), IPPROTO_UDP (UDP), IPPROTO_ICMP (ICMP), IPPROTO_RAW (IP).*

В общем случае, каждый коммуникационный канал определяется двумя узлами - источником и получателем данных, и может быть охарактеризован пятью параметрами:

1. Коммуникационным протоколом.
2. Локальным адресом.
3. Локальным процессом.
4. Удаленным адресом.
5. Удаленным процессом.

Адрес определяет операционную систему (или хост сети), а процесс - конкретное приложение, получающее или передающее данные. Непосредственные значения и формат этих параметров определяются типом и значением коммуникационного домена. Поскольку при создании сокета указывается только один параметр - коммуникационный протокол, то прежде чем передача данных между взаимодействующими процессами станет возможной, необходимо указать четыре дополнительных параметра для коммуникационного канала. Взаимодействующие стороны должны это делать согласованно, используя либо заранее определенные адреса (например, номера портов в семействе *TCP/IP*), либо договариваясь о них в процессе установления связи. Процессы могут обмениваться как с образованием виртуального канала (с предварительным установлением соединения), так и путем обмена датаграммами (без предварительного установления соединения).

На прикладном уровне схема использования механизма сокетов следующая. Взаимодействие между процессами происходит по схеме обмена клиент-сервер. В программе-клиенте и программе-сервере сперва выполняется системный вызов socket для создания сокета с определенными коммуникационным доменом, типом и протоколом. При этом домен может создаваться как для локальных межпроцессных взаимодействий, так и для удаленных, что определяется соответствующим параметром при вызове системной функции. Внутренние механизмы создания сокета от программиста скрыты и возлагаются целиком на систему. Далее происходит связывание коммуникационного домена с локальным и удаленным адресами и процессами. В сервере это выполняется за счет процедуры *bind*, в клиенте это происходит либо при вызове процедуры connect (в случае образования связи с предварительным установлением соединения), либо за счет процедуры *bind* (в случае обмена данными без предварительного установления соединения). Если используется механизм с предварительным установлением соединения, сервер "слушает" (процедура *listen*) поступающие запросы от клиентов, и при приходе запроса либо разрешает соединение (процедура accept), либо отклоняет его (*reject*). Далее обмен данными между процессами осуществляется через процедуры *read/write* (при обмене данными с предварительным установлением соединения) или

sendto/recvfrom (при обмене данными без предварительного установления соединения). После окончание работы с сокетом сокет закрывается процедурой *close*.

На уровне прикладного программирования интерфейс сокетов стандартизован и позволяет создавать легко переносимые программы для распределенных вычислений. Реализация модулей нижнего уровня, на которых базируется интерфейс сокетов, системно зависима и может выполняться по-разному, в зависимости от используемых аппаратных и системных ресурсов. Сокеты используются, главным образом, для организации межпроцессных взаимодействий в системах с общей памятью и в распределенных сетевых системах через протоколы *TCP/IP*. Поверх интерфейса сокетов может быть реализован, например, стандарт *MPI*.

Подсистема потокового ввода/вывода *STREAMS*

На сегодняшний день основным способом реализации сетевых драйверов и модулей протоколов в большинстве операционных систем *UNIX* является подсистема потокового ввода/вывода *STREAMS*. Подсистема *STREAMS* обеспечивает создание потоков - полнодуплексных каналов между прикладным процессом и драйвером устройства. Архитектура *STREAMS* определяет интерфейсы и набор правил, необходимых для взаимодействия различных частей этой системы и разработки модульных драйверов, обеспечивающих такое взаимодействие и обработку.

Большинство сетевых протоколов имеют многоуровневую организацию. Например, модель *OSI* (*Open Systems Interconnections* - взаимодействие открытых систем) иерархии сетевых протоколов, предложенная Международной организацией по стандартам (*ISO*), включает определение функциональности для 7 уровней. Различные семейства протоколов, например *TCP/IP*, имеют то или иное отображение на эту модель. Передача сетевых данных производится в виде пакетов или сообщений, и каждый уровень сетевого протокола производит определенную обработку сообщений и передает их другому уровню. Каждый уровень имеет стандартные интерфейсы взаимодействия с другими (верхним и нижним) уровнями, при этом он может взаимодействовать одновременно с различными протоколами верхнего и нижнего уровней. Например, протокол IP (уровень 3 мо-

дели *OSI*) может поддерживать работу нескольких протоколов верхнего уровня: *TCP* и *UDP*. На нижнем уровне протокол IP также взаимодействует с несколькими протоколами, обеспечивая передачу данных через различные сетевые интерфейсы (например, *Ethernet, Token Ring* или последовательный канал).

Подсистема *STREAMS* в большой степени призвана решить эти задачи. Она представляет собой интерфейс обмена данными, основанный на сообщениях, и обеспечивает стандартные механизмы буферизации, управления потоком данных и различную приоритетность обработки. В *STREAMS* дублирование кода сводится к минимуму, поскольку однотипные функции обработки реализованы в независимых модулях, которые могут быть использованы различными драйверами. Сам драйвер обеспечивает требуемую функциональность, связывая в цепочку один или несколько модулей, подобно тому как программный канал позволяет получить новое качество обработки, связав несколько независимых утилит.

Коммуникационный канал между процессом и драйвером *STREAMS* состоит из самого потока *STREAMS* и интерфейса системных вызовов, через которые прикладной процесс взаимодействует с потоком. Поток полностью располагается в пространстве ядра, соответственно и все функции обработки данных выполняются в системном контексте. Типичный поток состоит из головного модуля, драйвера, и, возможно, одного и более промежуточных модулей. Головной модуль взаимодействует с прикладными процессами через интерфейс системных вызовов. Драйвер, замыкающий поток, взаимодействует непосредственно с физическим устройством или псевдоустройством, в качестве которого может выступать другой поток. Модули выполняют промежуточную обработку данных.

Процесс взаимодействует с потоком, используя стандартные системные вызовы *open(), close(), read(), write()* и *ioctl()*. Дополнительные функции работы с потоками включают *poll(), putmsg()* и *getmsg()*. Передача данных по потоку осуществляется в виде сообщений, содержащих данные, тип сообщения и управляющую информацию. Для передачи данных каждый модуль, включая головной модуль и сам драйвер, имеет две очереди - очередь чтения (*read queue*) и очередь записи (*write queue*). Каждый модуль обеспечивает необ-

ходимую обработку данных и передает их в очередь следующего модуля. При этом передача в очередь записи осуществляется вниз по потоку (*downstream*), а в очередь чтения - вверх по потоку (*upstream*). Модули вниз и вверх по потоку взаимодействуют друг с другом стандартным образом, при этом модулю нет необходимости знать, какой части потока принадлежит следующая очередь - головному, промежуточному модулю или драйверу. Модули могут динамически встраиваться в поток или извлекаться из потока.

Подсистема *SREAMS* также обеспечивает возможность мультиплексирования потоков. Мультиплексирующий драйвер может быть подключен к нескольким модулям как вверх, так и вниз по потоку. С помощью мультиплексирующих драйверов потоки могут быть объединены в единый драйвер протоколов, поддерживающих несколько каналов передачи данных. Именно таким образом реализована поддержка сети во многих версиях операционной системы *UNIX*.

Основными компонентами потока являются модули. Каждый модуль состоит из пары очередей - очереди чтения и очереди записи, а также набора функций, осуществляющих обработку данных и их передачу вверх или вниз по потоку. На нижнем уровне с аппаратурой взаимодействуют драйверы, они же отвечают за обработку аппаратных прерываний. Аппаратура обычно генерирует прерывания при поступлении данных. В ответ на это драйвер копирует данные от устройства в пространство ядра, формирует сообщение и передает его вверх по потоку. Обработку системных вызовов процессов осуществляет головной модуль. Головной модуль потока является единственным местом, где возможно блокирование обработки и, соответственно, процесса, в контексте которого осуществляется операция ввода/вывода. Процесс передает данные потоку с помощью системных вызовов *write()* и *putmsg()*, и получает данные с помощью вызовов *read()* и *getmsg()*.

Все данные передаются в виде сообщений, сообщения являются единственным способом передачи информации между различными компонентами потока. Сообщение описывается двумя структурами данных: заголовком сообщения *msg (message block)* и заголовком данных *datab (data block)*. Обе эти структуры адресуют буфер дан-

ных, где находятся фактические данные сообщения. По мере продвижения по потоку в сообщение могут инкапсулироваться несколько блоков данных. Весь поток находится в системном контексте, поэтому при продвижении сообщения по потоку нет необходимости копировать блоки данных, достаточно передавать лишь указатели. Копирование из пользовательского адресного пространства в пространство ядра и наоборот происходит только при взаимодействии прикладного процесса с головным модулем потока с использованием интерфейса системных вызовов.

Прикладная программа может непосредственно взаимодействовать с подсистемой *STREAMS* для доступа с ее помощью к сетевым ресурсам. Во многих современных системах *UNIX* имеется специальный программный интерфейс взаимодействия прикладных программ с транспортными протоколами - интерфейс транспортного уровня *TLI (Transport Layer Interface)*. Этот программный интерфейс тесно связан с сетевой подсистемой *UNIX*, основанной на архитектуре *STREAMS*, изолируя от прикладной программы особенности сетевой архитектуры. Вместо того, чтобы непосредственно пользоваться общими функциями *STREAMS, TLI* позволяет использовать специальный набор вызовов, предназначенных для сетевых приложений. Схема *TLI* во многом схожа с интерфейсом сокетов и зависит от типа используемого протокола - с предварительным установлением соединения (например, *TCP*) или без него (например, *UDP*). В *TLI* используются те же функции сетевых взаимодействий, что и в сокетах: создание интерфейса, связывание коммуникационного узла, прослушивание канала. передача и чтение данных и др.

В операционной системе *UNIX System V* сетевые протоколы *TCP/IP* реализованы на основе архитектуры *STREAMS*. Модуль *IP* является гибридным мультиплексором, позволяющим обслуживать несколько потоков, приходящих от драйверов сетевых адаптеров (например, *Ethernet* и *FDDI*), и несколько потоков к модулям транспортных протоколов (*TCP* и *UDP*), а модули *TCP* и *UDP* - верхними мультиплексорами, обслуживающими прикладные программы. Как правило, сетевые и транспортные протоколы, составляющие базовый стек *TCP/IP*, поставляются одним производителем, в то время как поддержка уровней сетевого интерфейса и приложений может

осуществляться продуктами различных разработчиков. Соответственно, можно выделить два основных интерфейса взаимодействия, стандартизация которых позволяет обеспечить совместную работу различных компонентов программного обеспечения. Первый интерфейс определяет взаимодействие транспортного уровня и уровня приложений и называется интерфейсом поставщика транспортных услуг (*Transport Provider Interface, TPI*). Второй интерфейс устанавливает правила и формат сообщений, передаваемых между сетевым уровнем и уровнем сетевого интерфейса, и называется интерфейсом поставщика услуг канала данных (*Data Link Provider Interface*).

TPI представляет собой интерфейс предоставления услуг транспортного уровня *OSI* модели. Стандартизация этого интерфейса позволяет изолировать особенности реализации транспортного уровня от потребителя этих услуг и, тем самым, предоставить возможность разработки программного обеспечения независимо от конкретного протокола. *TPI* является интерфейсом между поставщиком транспортных услуг (*transport provider*) и потребителем транспортных услуг (*transport user*). Программный интерфейс *TLI* полностью реализует функциональность *TPI*.

DLPI определяет интерфейс между протоколами уровня канала данных (*data link level*) модели *OSI*, называемыми поставщиками услуг уровня канала данных, и протоколами сетевого уровня, называемыми пользователями услуг уровня канала данных. В качестве примера пользователей услуг уровня канала данных можно привести такие протоколы как *IP* и *IPX*. Снизу поставщик уровня канала данных взаимодействует с различными сетевыми устройствами, обеспечивающими передачу данных по сетям различной архитектуры (*Ethernet, FDDI, QsNet*). Для обеспечения независимости *DLPI* от реализации физической сети драйвер уровня канала данных состоит из двух частей: верхней аппаратно независимой и нижней аппаратно зависимой. Аппаратно независимая часть драйвера обеспечивает предоставление общих услуг, определенных интерфейсом *DLPI,* а также поддержку ряда потенциальных производителей, представляющих семейства протоколов *TCP/IP, NetWare* и *OSI*. Аппаратно зависимая часть непосредственно взаимодействует с адаптером.

Производители высокоскоростных коммуникационных сред, кроме собственных коммуникационных библиотек, иногда также поставляют драйвера *DLPI*, с помощью которых эти среды можно использовать для стандартных механизмов сетевых взаимодействий операционных систем. Например, компания *Quadrics* кроме собственно коммуникационной библиотеки *QNA*, поставляет также драйвер *DLPI*, через который поверх среды *QsNet* можно реализовывать сетевые протоколы *TCP/IP*, например, для создания распределенной файловой системы *NFS*.

3.2.2. Специализированные коммуникационные интерфейсы

Для построения высокопроизводительных многопроцессорных систем стандартные механизмы поддержки сетевых взаимодействий неэффективны. Реализация многоуровневой схемы сетевых взаимодействий требует больших вычислительных ресурсов, она критична по времени, так как каждый уровень производит обработку данных, причем часто это сопровождается многократным копированием данных и переключением контекстов. Достоинство многоуровневых моделей - их универсальность и переносимость, но при этом происходит потеря производительности. В суперЭВМ, где необходимо решать распределенные сильносвязные задачи, такие производительности коммуникационных интерфейсов неприемлемы. Перечисленные выше механизмы могут использоваться лишь в кластерных системах, где степень связности вычислительных элементов относительно невысокая.

Разработчики высокоскоростных коммуникационных сред предлагают специальные коммуникационные интерфейсы, которые, с одной стороны, более полно используют аппаратные возможности коммуникационной среды и, с другой стороны, представляют пользователю более быстрый и непосредственный доступ к коммуникационной среде из прикладной программы. Переносимость прикладного программного обеспечения при этом достигается за счет использования стандартов параллельных языков и параллельных расширений языков высокого уровня. Сами стандартные системы параллельного программирования при этом реализуются непосред-

ственно поверх специализированных коммуникационных библиотек. Хотя наибольшая производительность достигается при прямом использовании в прикладных приложениях специализированных коммуникационных библиотек.

В *SMP*-системах коммуникационный интерфейс обычно реализуется в рамках стандартных средств взаимодействия процессов, предоставляемых операционной системой. Наиболее эффективный механизм при этом - использование разделяемой памяти. Подобный механизм используется, например, интерфейсом *MPICH* при установке его на *SMP*-систему. Специальные коммуникационные интерфейсы имеет смысл разрабатывать для *MPP*-систем. Ниже будут рассмотрены интерфейсы для коммуникационных сред, в которых вычислительные модули связываются со средой через специальные коммуникационные контроллеры или адаптеры.

3.2.2.1. Виртуальный интерфейс *VI*

Виртуальный интерфейс *VI* (*Virtual Interface*) представляет собой открытую спецификацию, предложенную совместно корпорациями *Compaq, Intel* и *Microsoft,* для создания интерфейсов между аппаратным обеспечением высокопроизводительных коммуникационных сред и вычислительными системами [74]. Кроме приведенных трех фирм, в разработке и принятии спецификации приняло участие более ста фирм-разработчиков вычислительной техники и программного обеспечения. Задача этой архитектуры заключается в том, чтобы улучшить производительность распределенных приложений путем уменьшения задержек, связанных с наиболее критичными операциями передачи сообщений. Достигается это за счет уменьшения количества системных операций, выполняемых ядром операционной системы, и обеспечением более тесной связи прикладной задачи с коммуникационным интерфейсом.

В рамках *VI* приложения могут обмениваться данными и управляющей информацией напрямую, без вызовов системных функций ядра операционной системы и без многократного копирования данных из адресных пространств виртуальной памяти процессов и ядра. Достигается это за счет того, что адаптер коммуникационной среды

имеет прямой доступ к виртуальному адресному пространству пользовательского процесса. Производитель виртуального коммуникационного интерфейса (*VI Provider*) должен обеспечить соответствующие механизмы за счет добавления специального модуля (*Kernel Agent*) в ядро операционной системы, который чаще всего реализуется как некоторый драйвер [75].

Взаимодействие процессов происходит через специальные объекты, называемые Виртуальным Интерфейсом *VI*. Доступ к коммуникационной среде предоставляется процессу после вызова специальной функции создания *VI*. Прежде чем взаимодействовать, пара процессов должна установить соединение (connection) своих *VI*, подобно тому, как это организуется в механизме сокетов с предварительным установлением соединения. Обмен данными между несвязанными *VI* в рамках данной спецификации не предусматривается. Интерфейс определяет два основных способа обмена данными между процессами: первый осуществляется путем передачи и приема сообщений, второй - операциями прямого доступа к удаленной памяти.

Все пересылки управляются специальными очередями дескрипторов. Данные передаются напрямую между виртуальными адресными пространствами пользовательских процессов. Адаптер должен иметь доступ к тем областям виртуальной памяти процессов, в которых хранятся управляющие дескрипторы, и к областям, в которых хранятся передаваемые и принимаемые данные. Соответствующие страницы виртуальной памяти должны быть зарегистрированы в операционной системе, т. е. замкнуты в физической памяти и защищены от свопирования и перемещения во вторичную память. Таблицы преобразования виртуальных адресов процессов в физические адреса оперативной памяти должны быть также доступны адаптеру: при создании и регистрации таких страниц соответствующая информация копируется в адаптер. Регистры и служебная память адаптера отображаются в пространство виртуальной памяти пользовательского процесса.

Механизм передачи данных через *VI* следующий. Пользовательское приложение имеет две рабочих очереди дескрипторов (send queue и receive queue) и, возможно, одну очередь возврата исполнения дескрипторов (completion queue) для каждого объекта *VI*. Создав

дескриптор, процесс помещает его в очередь и посылает уведомление в адаптер в специальный регион, называемый *doorbell*, который выполнен как очередь *FIFO*. Получив уведомление, адаптер считывает дескриптор и выполняет его обработку. После завершения соответствующей операции, адаптер может уведомить об этом пользовательский процесс тремя способами: либо, установив соответствующий флаг завершения непосредственно в дескриптор (пользовательское приложение при этом производит периодический опрос соответствующего флага), либо, добавив запись выполнения дескриптора в *completion queue* (которую также периодически опрашивает пользовательское приложение), либо через механизм прерывания. После получения уведомления приложение должно удалить соответствующий дескриптор из очереди.

Существует два основных типа дескрипторов: дескрипторы операций пересылки/получения сообщений (*Send/Receive*) и дескрипторы операций прямого доступа к удаленной памяти (*RDMA - Remote Direct Memory Access*). Через дескрипторы первого типа процессы обмениваются сообщениями. Сообщение хранится в буфере зарегистрированной памяти. Короткие сообщения могут также содержаться непосредственно в дескрипторе (*immediate data*), обработка таких сообщений выполняется быстрей, чем обычных. Каждому дескриптору *Send* на передающей стороне должен соответствовать дескриптор *Receive* на принимающей стороне. Операции прямого доступа к удаленной памяти могут происходить как с уведомлением удаленного процесса, так и без уведомления. Операции *RDMA* выполняются по адресам в виртуальном адресном пространстве удаленного процесса, поэтому эти адреса должны быть заранее известны процессу-инициатору. Либо необходимо использовать механизм псевдоадресов, когда в сообщении передается некоторый тег, по которому на удаленном адаптере вычисляется соответствующий виртуальный адрес соответствующего процесса. Области, доступные для операций *RDMA*, могут быть защищены различными атрибутами.

Спецификация *VI* также содержит описание механизмов обеспечения различных уровней надежности передачи данных. Всего насчитывается три уровня надежности: ненадежная доставка, надежная доставка, надежный прием. Каждый интерфейс *VI* имеет

тот или иной уровень надежности. Обмен данными может производиться только между интерфейсами с одинаковыми уровнями надежности.

Архитектура виртуального интерфейса *VI* непосредственно реализована в коммуникационной библиотеке *Intel Virtual Interface Architecture* [76], а также в коммуникационной среде *Giganet cLAN* [77] и в среде *ServerNet* [78]. Принципы построения скоростных коммуникационных интерфейсов, предложенные в *VI*, так или иначе частично реализуются во многих современных коммуникационных интерфейсах различных производителей.

3.2.2.2. Коммуникационная библиотека *Myricom GM*

Библиотека *GM* [79] предназначена для использования с коммуникационными адаптерами фирмы *Myricom* для коммуникационной среды *Myrinet*. Библиотека *GM* частично реализует механизмы, предложенные в спецификации *VI*.

В интерфейсе *GM* процессы могут взаимодействовать через специальные объекты, которые называются портами. В *GM* реализуется механизм обмена данными без предварительного установления соединения: любые два порта могут взаимодействовать сразу же после своего создания. Данные передаются между виртуальными адресными пространствами пользовательских процессов без участия операционной системы. Сообщения могут иметь один из двух уровней приоритета. Для сообщений с одинаковым уровнем приоритета очередность обработки строго соответствует очередности поступления управляющей информации (т.е. выполняется в порядке заполнения очереди дескрипторов). Сообщения с высоким приоритетом могут обрабатываться вне очереди (для высокоприоритетных сообщений существует своя очередь дескрипторов).

Основным механизмом обмена данными является механизм передачи сообщений. Также в *GM* предусмотрена операция прямого доступа к удаленной памяти на запись (remote write). Данные могут передаваться и приниматься только в специальные области памяти, которые зарегистрированы в операционной системе и помечены, как доступные для *DMA* (*DMAable*). Интерфейс требует точного соответствия дескрипторов сообщений на узле-источнике пересылки

(*Send token*) и на узле-приемнике пересылке (*Receive token*). Атрибуты и размер принятого пакета должны точно совпадать с атрибутами и размерами ожидаемого пакета. Быстрые короткие сообщения могут содержаться непосредственно в дескрипторе, такие сообщения обрабатываются быстрее, чем обычные, так как не требуют дополнительных *DMA*-пересылок.

Интерфейс *GM* для пользователя представлен как библиотека функций, которые он может использовать непосредственно в *API*. Системные функции реализуются на уровне драйвера нижнего уровня (*raw driver*). Например, драйвер для ОС *Linux* имеет четыре стандартные точки входа из пользовательского приложения - *open(), close(), mmap(), ioctl()*. Операции работы с памятью (отображение памяти и регистров адаптера в адресное пользовательское пространство, регистрация памяти и получение информации о соответствии физических и виртуальных адресов процесса и др.) реализуются с использованием вызовов *mmap()* и *ioctl()*. Далее пользовательское приложение может непосредственно взаимодействовать с адаптером, не используя системные вызовы операционной системы.

В состав адаптера входит процессор *LANai*, и большинство интерфейсных функций реализуются микропрограммно на уровне *RISC*-ядра процессора *LANai*. Соответствующие функции описываются на языке Си, при инициализации системы исполняемый код функций записывается в память адаптера.

3.2.2.3. Коммуникационная библиотека *Quadrics QNA API*

Коммуникационный интерфейс для среды *QsNet*, предложенный корпорацией *Quadriqs Supercomputer Worlds*, несколько отличается от интерфейсов, описанных выше. Базовая парадигма интерфейсов совпадает, основная цель - разгрузить центральный процессор и операционную систему от обработки коммуникационных протоколов. Но *QNA* более глубоко интегрируется в систему, чем *VI* и *GM*. Коммуникационный адаптер *Quadrics ELAN* имеет полный доступ ко всему адресному пространству пользовательского процесса. Интерфейс *QNA* при встраивании в ядро операционной системы модифицирует механизм работы с виртуальной памятью с тем, чтобы обес-

печить такой доступ. Ядро поддерживает когерентность таблиц виртуальных страниц в центральном процессоре и в адаптере. В системе также поддерживается когерентность кэш памяти центрального процессора и процессора адаптера *ELAN*. Так как адаптер имеет полный доступ ко всему адресному пространству процесса, нет необходимости выделять и регистрировать специальные страницы. В случае, если адаптер пытается обратиться к виртуальной странице, которая находится во вторичной памяти, в адаптере вырабатывается соответствующее исключение, и управление передается в центральный процессор для того, чтобы система могла перенести страницу в физическую память, к которой имеет доступ адаптер. Для того, чтобы с адаптером одновременно могли работать несколько независимых процессов, каждому процессу в адаптере выделяется свой набор регистров, которые отображаются в пользовательское адресное пространство. По тому, через какую область адресов (физических адресов адаптера) произошло обращение к адаптеру, адаптер определяет, какому процессу соответствует вызываемая функция.

Базовые коммуникационные операции, поддерживаемые средой *QsNet* - это операции удаленной записи и удаленного чтения. Операции записи и чтения выполняются по виртуальным адресам, поэтому локальный процесс должен знать виртуальные адреса удаленного процесса, по которым расположены данные. В рамках модели *SPMD* на разных узлах выполняется одинаковый код, поэтому статические данные процессов должны располагаться по одинаковым адресам. Если объявлять глобальные объекты как статические, они будут автоматически доступны с удаленных узлов, никаких процедур предварительного установления соединения и нахождения соответствия адресов не требуется. Библиотека *QNA API* также предлагает специальные механизмы для размещения объектов по одинаковым или заданным виртуальным адресам. Таким образом, обеспечивается механизм прямого взаимодействия с удаленными данными.

Интерфейс также поддерживает операции прямого доступа к удаленной памяти для выполнения операций блочного чтения и записи. В среде поддерживаются операции коллективного взаимодействия процессов (широковещательные пересылки, сборку, глобальный обмен и др.).

3.3. *MPICH* - переносимая реализация стандарта *MPI*

Интерфейс передачи сообщений *MPI* является универсальным средством программирования параллельных программ в рамках модели передачи сообщений. Стандарт описывает набор функций, используя которые можно создавать переносимые параллельные программы. Реализация стандарта, вообще говоря, является платформозависимой и разрабатывается для конкретной параллельной машины. Самая эффективная реализация *MPI* строится непосредственно поверх библиотеки коммуникационного интерфейса, предоставляемой производителем системы. Однако, существуют переносимые реализации стандарта. Наиболее распространенная переносимая реализация стандарта *MPI* носит название *MPICH* [80] (от *MPI Chameleon*).

В виду того, что на нижнем уровне коммуникационные интерфейсы различаются, переносимость *MPICH* не абсолютная. *MPICH* переносим только на верхнем уровне, предлагая на нижнем уровне точки подключения специализированных интерфейсов к MPICH неким стандартным образом. Самая простая реализация переносимости достигается за счет использования *TCP/IP* стека, так как поддержка протоколов *TCP/IP* поддерживается всеми производителями вычислительных систем. Однако, сами разработчики *MPICH* подчеркивают, что использование *TCP/IP* оправдано только для кластерных систем, состоящих из рабочих станций и серверов. Использование этих протоколов для реализации *MPICH* на *MPP*-системах обладает крайне низкой эффективностью.

Центральным механизмом достижения переносимости MPICH при сохранении высоких производительностей является использование спецификации *ADI* [81] (*Abstract Device Interface* - абстрактный интерфейс устройства). Все функции *MPI* реализованы на основе макросов и функций, составляющих *ADI*. *ADI*, по сути, является точкой подключения переносимого MPICH непосредственно к непереносимому коммуникационному интерфейсу. *ADI* определяет набор минимальных требований - функций, которые должен реализовывать коммуникационный интерфейс, на основе которых можно построить стандарт *MPI*. Различные коммуникационные интерфейсы обладают различными возможностями, и для того, чтобы можно

было наиболее полно использовать все преимущества того или иного коммуникационного интерфейса на высоком уровне *MPI*, *ADI* кроме базовых функций (*channel interface* в *ADI*-1, *ADI*-2 или *ADI core* в *ADI*-3) имеет ряд расширений. Если коммуникационный интерфейс реализует лишь минимальные требования *ADI*, то все остальные функции эмулируются в *ADI* программно на основе базовых функций. Если интерфейс нижнего уровня обладает дополнительными функциями, то соответствующие макросы и функции *ADI* можно переопределить, основываясь на функциях коммуникационного интерфейса, с целью более полно использовать аппаратные и низкоуровневые возможности среды.

MPICH позволяет одновременно в одной системе использовать различные механизмы межпроцессного взаимодействия, позволяя создавать гетерогенные системы. Например, система может состоять из *SMP*-узлов, с помощью высокопроизводительной среды образующих *MPP*-гиперузлы, которые объединены сетью в суперкластеры. На уровне *SMP*-узла *MPICH* может использовать механизмы разделяемой памяти межпроцессных обменов, на уровне *MPP* используется *ADI*, реализованный поверх высокопроизводительного коммуникационного интерфейса, а на уровне кластера используется стек протоколов *TCP/IP* и механизм сокетов.

Кроме собственно реализации *MPI* как системы межпроцессных взаимодействий, *MPICH* также предлагает средства запуска и отладки программ на распределенных ресурсах. Таким образом, *MPICH* предлагает готовое решение *ready-to-market* для быстрой и эффективной реализации стандарта *MPI* практически на любой платформе. *MPICH* поставляется в текстах исходных программ и обладает широкими возможностями по конфигурированию и настройке под конкретную систему.

Глава 4. Компьютерные сети.

Компьютерная сеть, или просто сеть, - это совокупность компьютеров и других устройств, соединенных между собой каналами связи, позволяющие совместно использовать ресурсы и информацию. В сети, по крайней мере, одно устройство должно отправлять/принимать данные и одно устройство соответственно принимать/отправлять данные.

Коммуникационная среда обеспечивает передачу информации между абонентами системы и включает электрические и/или оптические кабели или радиоволны в случае беспроводной сети, а также устройства коммутации, позволяющие выполнить требуемые соединения между абонентами.

Основные аппаратные компоненты коммуникационной среды: сетевые контроллеры (*network interface controller cards - NICs*), рипитеры (*repeaters*), концентраторы (хабы, *hubs*), мосты (bridges), коммутаторы (*switches*), маршрутизаторы (роутеры, *routers*) и межсетевые экраны (брандмауэр, *firewalls*).

Рипитеры - это электронные устройства, принимающие электрические сигналы от передающих абонентов и формирующие их (после прохождения по каналу из-за наличия паразитных элементов сигнал искажается) в соответствие с используемым стандартом сети, и передающие эти сигналы принимающим абонентам. Это позволяет увеличить расстояние для связи абонентов. Рипитеры с несколькими портами являются **концентраторами**. Рипитеры и концентраторы обеспечивают только нижний физический уровень 7-уровневой модели *OSI*, то есть они являются неинтеллектуальными устройствами. При передаче данных через рипитер возникает задержка, в результате в ряде архитектур это приводит к ограничению числа рипитеров (время передачи превышает заданные значения в стандарте). Например, в сети *Ethernet* на пути сигнала не может быть более 4-х репитеров и не более 5-ти сегментов (участков), причем только к трем из них могут быть подключены устройства, в противном случае не все абоненты в сети смогут зарегистрировать возможные коллизии. Эти выводы обычно выражаются в виде правила "5-4-3".

Мосты соединяют несколько сегментов сети, изолируя их физический уровень, и поддерживают 2 уровень 7-уровневой модели OSI. Мосты могут передавать информацию или в рамках одного стандарта, или между сегментами сети разных типов. В сети Ethernet мосты позволяют обойти правило "5-4-3", изолируя сбойные пакеты и коллизии между сегментами. Мосты обеспечивают передачу на все порты, кроме порта, который осуществил приём информации. В отличие от концентраторов мосты не выполняют неупорядоченное копирование на все порты, а определяют, какой адрес доступен и через какой порт. Связав порт с адресом, мост направляет трафик именно на этот адрес и только на выбранный порт. Например, возможен непрозрачный мост между сетями *PCIe* и *RapidIO*, который из пакетов стандарта *PCIe* формирует пакеты *RapidIO* и передаёт их в порт, в соответствии с адресом, содержащимся в передаваемом пакете.

Маршрутизаторы являются межсетевыми устройствами, которые пересылают пакеты между сетями путём обработки маршрутной информации, включенной в пакет или датаграмму (интернет протокол 3-го уровня). Информация о маршрутизации часто обрабатывается в соответствие с таблицей маршрутизации (или таблицей переадресации). Маршрутизатор использует свою таблицу маршрутизации, чтобы определить, куда направлять пакеты. Адрес назначения в таблице маршрутизации может включать в себя "*null*" интерфейс, известный также как "черная дыра", потому, что данные могут войти в него, однако никакой дальнейшей обработки может не проводиться.

Межсетевые экраны - это сетевое устройство для контроля сетевой безопасности и правила доступа. Межсетевые экраны обычно настраиваются для отклонения запросов доступа от непризнанных источников, позволяя действия от признанных источников. Роль межсетевых экранов постоянно растёт в сетевой безопасности параллельно с постоянным увеличением числа кибер-атак.

Основными параметрами коммуникационной среды являются латентность (*latency*, время пересылки сообщения нулевой длины), и максимальная пропускная способность (*bandwidth*, максимальное количество информации которое можно переслать за единицу времени).

4.1. Классификация компьютерных сетей

Сети могут быть классифицированы по целому спектру характеристик, таких как размер территории сети (длина каналов связи), тип устройств для транспортировки данных, протокол связи, масштабируемость, топология, функциональное назначение, тип функционального взаимодействия, скорость передач, необходимость поддержания постоянного соединения, надёжность и пр.

По размеру территории сети можно классифицировать как

PAN (*Personal Area Network*) - персональная сеть, предназначенная для взаимодействия различных устройств, принадлежащих одному владельцу.

LAN (*Local Area Network*) [84] - локальные сети, имеющие замкнутую инфраструктуру для доступа к общим ресурсам и данным. Можно дать определение *LAN* или ЛВС как сети, соединяющей компьютеры и устройства на ограниченных географических пространствах, таких, как дом, школа, научные лаборатории, офисные здания или близко расположенные группы зданий. Каждый компьютер или устройство в сети называют узлом (*node*). Локальные сети являются сетями закрытого типа, где доступ разрешен только ограниченному кругу пользователей.

CAN (*Campus Area Network*) - кампусная сеть, объединяет локальные сети близко расположенных зданий.

MAN (*Metropolitan Area Network*) - городские сети между учреждениями в пределах одного или нескольких городов, связывающие много локальных вычислительных сетей.

WAN (*Wide Area Network*) - глобальная сеть, покрывающая большие географические регионы, включающие в себя как локальные сети, так и прочие телекоммуникационные сети и устройства. Глобальные сети являются открытыми и ориентированы на обслуживание любых пользователей.

Internet (Интернет) является глобальной системой связи компьютеров, использующей стандартный протокол *Internet Protocol Suite* (*TCP/IP*) для обслуживания миллиардов пользователей. Протокол *Internet Protocol Suite* является основой всех современных межсетевых взаимодействий.

По области применения сети можно классифицировать следующим образом.

PCN (*Process Control Network*) - сети управления процессом. Это коммуникационные сети, используемые для передачи инструкций и данных между устройствами контроля и управления и оборудованием, осуществляющим диспетчерское управление и сбор данных (*Supervisory Control* and *Data Acquisition* - *SCADA*). Эти сети с течением времени заимствовали ряд технологий и топологий из других приложений, однако сети *PCN* имеют несколько специальных требований для возможности применения в индустрии, это требования надежности, детерминированности, совместимости. Надежность включает в себя такие требования, как избыточность соединений, пониженная чувствительность к электромагнитному воздействию (*EMI*) и обнаружение и исправление ошибок. Детерминированность предполагает обеспечение того, чтобы каждое устройство имело гарантированный доступ к сети, и ряд механизмов, позволяющих осуществить приоритетную передачу информации (например, при аварии) в системе. Совместимость позволяет *SCADA* и распределенным системам управления (*DCS*) различных производителей обмениваться с контрольно-измерительным оборудованием других производителей. Ранее сети управления процессами строились как последовательные с использованием на нижнем уровне стандарты *EIA RS*-422 и *EIA RS*-485. Один из де-факто стандартов (который стал открытым стандартом) является стандарт *Modbus*, созданный компанией *Modicon*. С использованием сетей управления создаются автоматизированные системы управления технологическим процессом (АСУ ТП). АСУ ТП - это группа решений технических и программных средств, предназначенных для автоматизации управления технологическим оборудованием на промышленных предприятиях. Под АСУ ТП обычно понимается целостное решение, обеспечивающее автоматизацию основных операций технологического процесса на производстве в целом или каком-то его участке, выпускающем относительно завершённое изделие.

SOHO (*Small Office / Home Office* - малый/домашний офис) - домовые сети. Сети, предназначенные для домашнего использования. Сети не предназначенны для больших производственных нагрузок и

не чувствительны к длительным периодам бездействия. Ещё один термин подобных сетей - ***HAN*** (*home area network*, домашняя сеть), сеть предназначенная для связи между цифровыми устройствами, обычно устанавливаемых дома, такими как небольшое число персональных компьютеров, принтеров, мобильных устройств. Важная функция таких сетей – доступ в интернет.

SAN (*Storage Area Network*) - сеть хранения данных, СХД. Это сети для подключения внешних устройств хранения данных, таких как дисковые массивы, ленточные библиотеки, оптические приводы к серверам таким образом, чтобы операционная система распознала подключённые ресурсы как локальные. Несмотря на то, что стоимость и сложность таких систем постоянно падают, сети хранения данных в основном используют только большие предприятия. *SAN* характеризуются предоставлением так называемых сетевых блочных устройств (обычно посредством протоколов *Fibre Channel, iSCSI* или *AoE*), в то время как сетевые хранилища данных (*Network Attached Storage - NAS*) нацелены на предоставление доступа к хранящимся на их файловой системе данным при помощи сетевой файловой системы (такой как *NFS, SMB/CIFS*, или *Apple Filing Protocol*). Следует обратить внимание, что категорическое разделение вида «*SAN* — это только сетевые диски, *NAS* — это только сетевая файловая система» является искусственным: осуществляется взаимное проникновение технологий с целью повышения гибкости и удобства их применения.

SAN (*System area networks*) - высокопроизводительная сеть масштаба вычислительной системы, ориентированная на соединение компьютерных кластеров или создание высокопроизводительных вычислительных систем, единой распределенной вычислительной системы, параллельной ЭВМ. В частности, данный тип сети используется *Microsoft SQL Server* 2005 г. для связи через Виртуальный Интерфейс Адаптера (VIA). Эта технология используется с появлением *Windows* 2000. Термин вводится, чтобы отличить такие среды от сред *LAN*– локальных вычислительных сетей (ЛВС). Особенность *SAN*, которая отличает ее от ЛВС – это направленность на организацию эффективных обменов информацией между процессами, выполняющимися одновременно на вычислительных узлах системы. В то время как задача ЛВС – это организация обмена информацией

между машинами сети. Как видно, существует два термина *SAN*, имеющие разные значения.

Существует целый ряд стандартов и спецификаций для создания коммуникационных сред, таких как *Virtual Interface, InfiniBand* [82], *SCI, Myrinet* [83], *RapidIO* [84], *Ethernet* [85] и др.

EPN (*Enterprise private network* - сеть частных предприятий) - сети, построенной на предприятиях для обеспечения взаимодействия различных участков компании, например, производственные участки, офисы, удаленные офисы, магазины. Подобные сети делаются для совместного использования компьютерных ресурсов.

VPN (*virtual private network* - виртуальные частные сети) - компьютерная сеть, в которой некоторые связи между узлами остаются открытыми или виртуальными в общей большой сети (например, Интернете). В этом случае на сетевом уровне данных определяется тунелирование данных через большую сеть. Одно из применений – защищённый обмен через Интернет. Однако *VPN* не требуется иметь явных функций безопасности, таких как аутентификация или шифрование. *VPN*, например, может использоваться для разделения трафика различных групп пользователей в сети с введением функций безопасности.

Virtual Network (виртуальные сети, не путать с *VPN*). Виртуальные сети определяют поток данных трафика между виртуальными машинами в гипервизоре виртуальной компьютерной среды. Виртуальные сети могут использовать виртуальные коммутаторы, виртуальные маршрутизаторы, виртуальные межсетевые экраны и другие виртуальные сетевые устройства для защищённой передачи данных.

Коммуникационные среды для создания параллельных ЭВМ можно выделить в отдельный класс коммуникационных сред по аналогии с разделением на классы глобальных сетей и ЛВС (см. рис. 4.1).

В качестве параметра разделения по классам можно указать характерные расстояния, а также назначение сред:

- глобальные сети: объединение независимых ЛВС и удаленных пользователей;

• ЛВС: объединение пользователей в рамках рабочих групп, создания стендов тестирования и поверки аппаратуры, управление функционирования объектов;

• среды масштаба ВС: создание параллельной ЭВМ, обеспечение эффективного взаимодействия процессов, идущих параллельно на разных модулях параллельной ЭВМ.

В соответствии с назначением коммуникационных сред масштаба вычислительной системы, можно выделить характерные требования, предъявляемые к таким средам:

• малые времена задержек на межпроцессные обмены (1-1000 мкс, задержка на один физический линк 60-120 нс);

• высокая пропускная способность (1-50 Гбит/с на каждое соединение);

• возможность совмещения по времени передач данных через среду с вычислениями в модулях;

• базирование на стандартах;

• надежные протоколы с управлением потоком и коррекцией ошибок;

• поддержка различных топологий вычислительных систем;

• масштабируемость, конфигурируемость и технологичность исполнения.

Разработчики высокоскоростных коммуникационных сред предлагают специальные коммуникационные интерфейсы, которые, с одной стороны, более полно используют аппаратные возможности коммуникационной среды и, с другой стороны, представляют пользователю более быстрый и непосредственный доступ к коммуникационной среде из прикладной программы. Переносимость прикладного программного обеспечения при этом достигается за счет использования стандартов параллельных языков и параллельных расширений языков высокого уровня. Сами стандартные системы параллельного программирования при этом реализуются непосредственно поверх специализированных коммуникационных библиотек. Наибольшая производительность достигается при прямом использовании в прикладных приложениях специализированных коммуникационных библиотек.

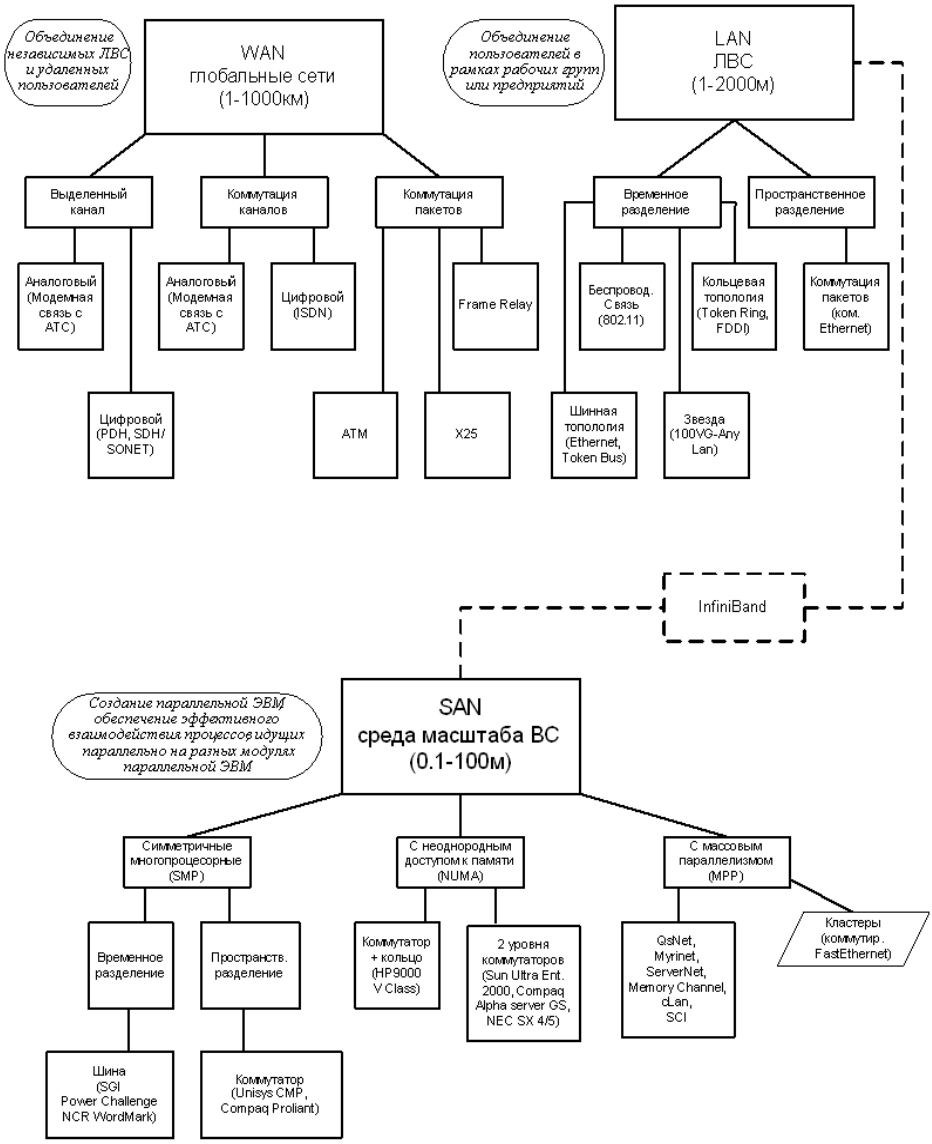


Рис. 4.1. Классы коммуникационных сред

В *SMP*-системах (симметричных мультипроцессорных системах) коммуникационный интерфейс обычно реализуется в рамках стандартных средств взаимодействия процессов, предоставляемых операционной системой. Наиболее эффективный механизм при этом - использование разделяемой памяти. Подобный механизм используется, например, интерфейсом *MPICH* при установке его на *SMP*-систему. Специальные коммуникационные интерфейсы имеет смысл разрабатывать для *MPP*-систем (массивно-параллельных систем).

4.2. Основные типы топологий сетей

Топология сети может быть физическая или логическая. Физическая топология означает физический проект сети, включающий устройства, расположения и кабельные (беспроводные) подключения. Логическая топология означает правила функционирования сети, то есть, как данные передаются в сети в соответствии с реализованной физической топологией.

Физическая сеть может иметь самые различные варианты архитектуры. Связь между отдельными структурами осуществляется по следующим принципам: передача только в одну сторону (симплексная связь), передача в оба направления по раздельным линиям связи (дуплексная связь) и передача в оба направления по одним линиям связи (полудуплексная связь) – см. рис. 4.2.

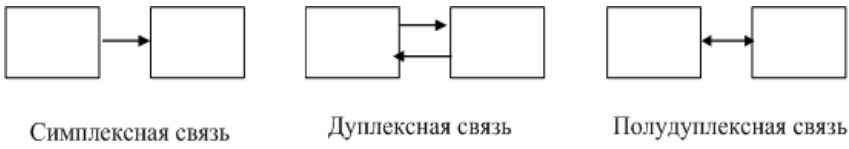


Рис. 4.2. Принципы связи узлов сети.

Наиболее известные архитектуры физической сети: точка-точка (*point-to-point*), шина (*bus*), звезда (star), кольцо (*ring*), полносвязная (*fully connected mesh*), частичносвязная (*partially connected mesh*), дерево (*tree*), трёхмерный тор (3*D torus*). Данная классификация основана на межсвязях вычислительных узлов для физических и логических типов сети.

4.2.1. Топология точка-точка

Топология точка-точка является простейшей топологией связи между 2 конечными точками (*endpoint*). Коммутируемая топология точка-точка является основной моделью в телефонии. Используются коммутируемые цепи или коммутируемые пакеты, связь точка-точка может устанавливаться динамически и обрываться, когда она больше не нужна. Это основная мода в традиционной телефонии.

4.2.2. Топология шина

Топология **шина** (см. рис. 4.3) наиболее распространённая в прошлом топология, используемая не только для связи электронных узлов внутри кристалла, на модуле и между модулями, но и для создания локальной сети, в частности сети *Ethernet*. В сети *Ethernet* с этой топологией каждый узел связан со всеми по одному кабелю через специальные разъёмы. На концах кабеля находятся согласователи – терминаторы, служащие для снижения отражения сигналов. Передаваемый пакет распространяется по обоим направлениям кабеля до тех пор, пока не будет обнаружен требуемый *MAC* или *IP* адрес. Если адрес не обнаруживается, данные игнорируются, при совпадении адресов передачи, находящегося в заголовке пакета и приёма, происходит обмен данных.

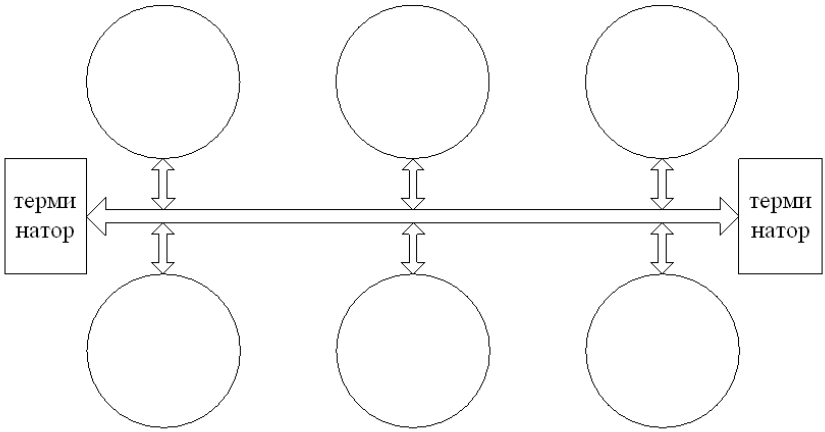


Рис. 4.3. Топология шина.

Топология **линейная шина** – это топология, где все узлы сети соединены с общей средой передачи (часто называемой транком), которая имеет точно 2 конечные точки. Все передаваемые между узлами данные передаются по этой общей среде и логически одновременно (в пределах времени задержек распространения сигналов, что накладывает ограничения на скорость передачи данных) поступают на все узлы. Две конечные точки согласуются устройствами, называемыми терминаторами, которые предотвращают обратное распространения сигнала в транке (отражение).

Топология **распределённая шина** – это топология, в которой все узлы сети соединены с общей средой передачи, которая содержит больше 2 конечных точек, образуя дополнительные ветви к основному транку, при этом все данные между узлами передаются одновременно.

Примечание.

1. Все конечные точки среды передачи обычно терминируются устройствами, называемыми терминаторами.

2. Физически топология распределённой шины иногда некорректно называется топологией дерево, однако, хотя физически распределённая топология шины соответствует физической топологии

дерево, она отличается тем, что отсутствует центральный узел, к которому подключены другие узлы, таким образом, иерархическая функциональность заменена общей шиной.

3. Физически топология линейной шины иногда рассматривается особым случаем физически распределённой топологии, то есть распределённой топологией без сегментов ветвления.

4.2.3. Топология звезда

Топология звезда (см. рис. 4.4) – это топология, когда узлы сети соединены друг с другом через центральный хаб, коммутатор или центральный компьютер. В отличие от топологии шина, каждый узел топологии звезда соединён с центральным узлом связью точка-точка. Все данные в этой топологии проходят через центральный узел. Хаб действует как ripитер. Достоинство топологии в простоте подключения узлов, главный недостаток – в невысокой надёжности, выход центрального узла приводит к выходу из строя всей сети.

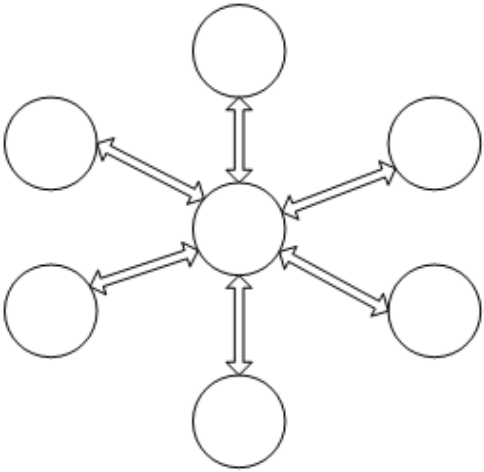


Рис. 4.4. Топология звезда.

Примечание.

Связь точка-точка иногда рассматривается как вырожденный случай топологии звезда, таким образом, простейший тип сети звезда является связь двух узлов через хаб, являющимся арбитром

таких связей. Хаб позволяет осуществить связь только 2 узлов, в то время как коммутатор максимально может связать число узлов, равное целой части $n/2$, где n – число узлов, подключенных к коммутатору. Обмен одного узла с другими может быть широковещательным или нет в зависимости от того, может ли сеть автоматически распространять сигналы от хаба ко всем узлам, или только к индивидуальному узлу.

Расширенная звезда – это сеть звезда, где центральный узел имеет один или более рипитеров, используемых для увеличения расстояния передачи между узлами.

Распределённая звезда – топология нескольких сетей звезда, соединённых друг с другом линейно по типу последовательной цепочки без центрального узла.

4.2.4. Топология полносвязная сеть

Топология полносвязная сеть (другое название топологии «каждый с каждым», см. рис. 4.5) является топологией, где каждый узел соединён со всеми остальными узлами. Число связей в этой топологии равно $n\times(n-1)/2$, где n – число узлов в сети, поэтому такая топология используется только для связи небольшого числа узлов.

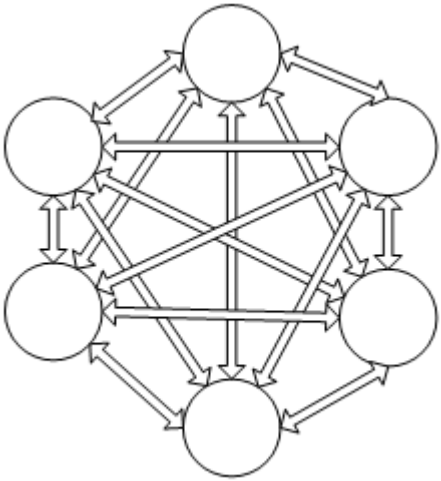


Рис. 4.5. Полносвязная топология.

4.2.5. Топология частичносвязная сеть

Топология частичносвязная (или «каждый с несколькими», см. рис. 4.6) – сеть, в которой один или несколько узлов может иметь связи с несколькими узлами, повышая тем самым надёжность и пропускную способность сети. В случае выхода из строя одного узла, возможна передача между другими узлами по другому, более длинному пути.

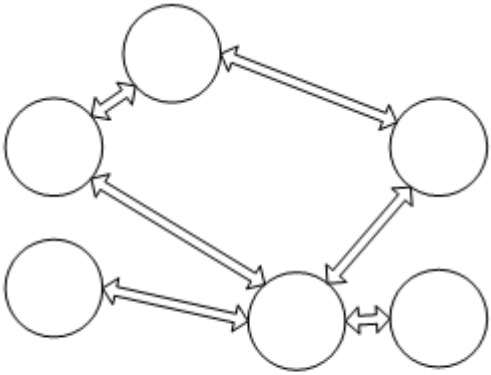


Рис. 4.6. Топология частичносвязная.

4.2.6. Топология дерево

Топология дерево (см. рис. 4.7) имеет ещё одно название иерархическая сеть. В этой топологии имеется центральный узел (верхний уровень иерархии), соединённый связью точка-точка с 1 или более узлами следующего, более нижнего, второго уровня, каждый узел второго уровня соединён с 1 или более узлами следующего уровня, третьего уровня, и т.д. Сеть дерево имеет не менее 3 уровней, если в сети 2 уровня, то это уже архитектура звезда. Эта сеть может содержать большое число узлов при небольшом числе каналов в узле, таким образом, она легко наращивается. Суммарное число связей в

этой топологии на 1 меньше числа узлов в сети. При обмене между узлами узел, имеющий верхний уровень, требует выполнения большего числа операций по сравнению с узлом нижнего уровня (выполняет часть функций этого узла).

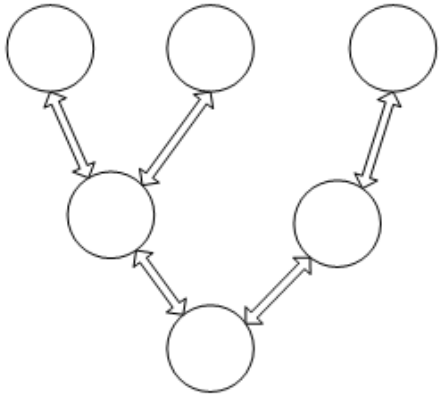


Рис. 4.7. Топология дерево.

4.2.7. Топология трёхмерный тор

Топологию трёхмерный тор можно рассматривать как куб с равномерно распределёнными внутри узлами. В топологии трехмерный тор все узлы, находящиеся внутри куба, соединены со всеми ближайшими соседями, то есть образуется 6 связей каждого внутреннего узла с ближайшими соседями, внешние же узлы соединены с узлами, находящимися с противоположной стороны. Таким образом, узел, находящийся на стороне куба имеет 5 связей с ближайшими соседями, на гране куба – 4 связи с ближайшими соседями, а в вершине – 3 связи.

4.2.8. Топология решётка

Топология решётка представлена на рис. 4.8. Её можно рассматривать как частный случай частичносвязной топологии.

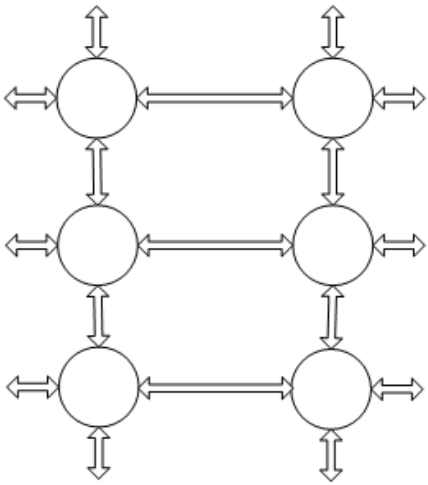


Рис. 4.8. Топология решётка.

4.2.9. Комбинация топологий

Можно также придумать топологии, которые являются комбинацией нескольких топологий. Например, для архитектуры кольцо предусмотреть дополнительные связи в соответствие с рисунком 4.9. Основная цель таких построений уменьшить задержку прохождения данных. В этой структуре каждый узел связан с соседним узлом и с узлом, расположенным через 1 от соседнего узла. Таким образом, каждый узел соединяется с 4 узлами напрямую и только с 1 узлом через дополнительный узел, условно имеется 1 задержка t для соединения с 4 узлами и 2 задержки t для соединения с 1 узлом. При большом числе узлов, каждый из которых имеет 4 канала, можно организовывать дополнительные связи с соседними узлами и с узлами, расположенными через 2 узла, через 3 узла и т.д. Нетрудно получить число возможных узлов без нарушения симметрии для каждой из такой структуры. Если имеются дополнительные связи через 1 узел, то число узлов

$N = 6 + 2 \times n$, где n целое число. Если связи через 2 узла, то число узлов:

$N = 9 + 3 \times n$, если связи устанавливаются через произвольное число i узлов, то возможное число узлов без нарушения симметрии:

$N = 6 + 3 \times i + (2 + i) \times n$, где $i = 0, 1, 2\ldots$; n = 0, 1, 2, …

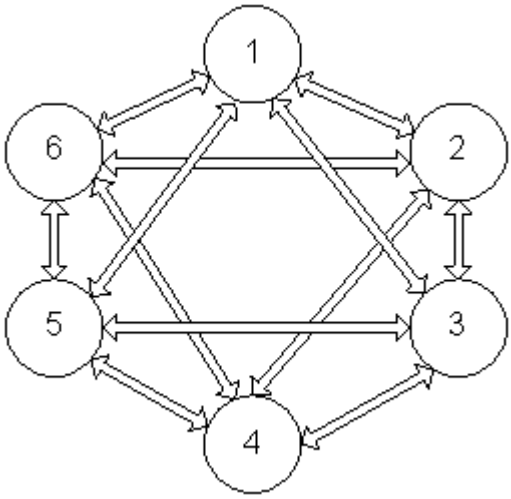


Рис.4.9. Топология кольцо с дополнительными связями «через 1 узел».

Для наглядности в таблице 4.1 представлено распределение числа узлов с разным числом задержек для топологии «через 1 узел» при наличии в узле 4 каналов связи. В таблицах 4.2 – 4.4 представлены топологии с большим числом дополнительных связей. Наиболее важным параметром для таких структур является время прохождение информации до наиболее удалённых узлов.

Таблица 4.1. Распределение числа узлов с разными задержками (разным удалением от каждого узла) в системе с топологией «через 1 узел».

Число (t) задержек	6 узлов в системе	8 узлов в системе	10 узлов в системе	12 узлов в системе	14 узлов в системе	16 узлов в системе
1	4	4	4	4	4	4
2	1	3	4	4	4	4
3	-	-	1	3	4	4
4	-	-	-	-	1	3
5	-	-	-	-	-	-

Таблица 4.2. Распределение числа узлов с разными задержками (разным удалением от каждого узла) в системе с топологией «через 2 узла».

Число (t) задержек	9 узлов	12 узлов	15 узлов	18 узлов	21 узел	24 узла	27 узлов
1	4	4	4	4	4	4	4
2	4	5	6	6	6	6	6
3	-	2	4	5	6	6	6
4	-	-	-	2	4	5	6
5	-	-	-	-	-	2	4

Таблица 4.3. Распределение числа узлов с разными задержками (разным удалением от каждого узла) в системе с топологией «через 3 узла».

Число (t) задержек	12 узлов	16 узлов	20 узлов	24 узла	28 узлов
1	4	4	4	4	4
2	6	7	8	8	8
3	1	2	6	7	8
4	-	-	1	4	6
5	-	-	-	-	1

Таблица 4.4. Распределение числа узлов с разными задержками (разным удалением от каждого узла) в системе с топологией «через 4 узла».

Число (t) задержек	15 узлов	20 узлов	25 узлов	30 узлов	35 узлов	40 узлов
1	4	4	4	4	4	4
2	6	7	8	8	8	8
3	4	6	8	9	10	10
4	-	2	4	6	8	9
5	-	-	-	2	4	6
6	-	-	-	-	-	2

Четыре канала в узле позволяют также строить топологию «решётка». Для 9 узлов на плате максимальная задержка не превышает $4t$. Видно, что она проигрывает по времени задержки топологии «кольцо» с дополнительными связями (топологии «частичносвязная»).

Одной из наиболее распространённых архитектур в современных серверах является группирование четвёрок соединений «каждый с

каждым», то есть 3 канала используются для связи «каждый с каждым», а дополнительные каналы используются для обмена с другими четвёрками. При 16 узлах на плате максимальная задержка для такой топологии не превышает 3t и остаётся ещё 4 канала для связи с другими модулями. Такая топология по быстродействию примерно соответствует топологии «кольцо» с дополнительными связями.

Для топологии звезда с микросхемой коммутатора, имеющей число каналов, больших числа узлов, задержка в передаче данных между 2 любыми узлами составляет 2t (см. рис. 4.4). Для 10-процессорного модуля достаточно иметь 12-канальный коммутатор (2 канала для связи с другими модулями). Данная топология позволяет создавать системы с наименьшими максимальными задержками, однако за счёт того, что здесь отсутствуют связи с задержкой 1t, средняя производительность на отдельных задачах может проигрывать предыдущей архитектуре. Следует также учитывать, что выход из строя коммутатора приведет к выходу всего узла, что свойственно любым архитектурам типа «звезда».

Одним из важнейших параметров коммуникационной системы является её энергопотребление. Высокоскоростные каналы вносят существенный вклад (до 20 % в зависимости от частоты) в энергопотребление коммуникационной системы и уменьшение числа каналов может привести к существенному снижению потребления питания. В этом отношении топология звезда является одной из лучших.

При выборе топологии сети необходимо также учитывать задержки в передаче данных, возникающие вследствие занятости среды передачи другими узлами или занятостью на передачу самого приёмного узла. Для топологии «каждый с каждым» состоящей из n узлов при равновероятной передаче длительностью t_n со средней частотой f_n средняя частота наложений передач $f_н$ составит [87]:

$$f_н = \left[(n-1) \times \frac{f_n}{n-1}\right]^2 \times t_n = f_n^2 \times t_n$$

Для топологий шина и звезда при использовании хаба (для коммутатора зависимость более сложная и для ряда передач она не зависит от числа узлов) частота наложений уже квадратично зависит от числа модулей.

$$f_н = (f_n \times n)^2 \times t_n$$

Логическая топология сети определяет передачу данных по сети от одного узла к другому вне зависимости от физической топологии сети, и не обязательно логическая топология совпадает с физической. Например, витая пара Ethernet это логическая топология в физической топологии звезда, или сеть *Token Ring* является логической топологией кольцо, физически образуя звезду.

Логическая топология сети имеет ту же классификацию, что и физическая сеть и описывает пути передачи данных между узлами вне зависимости от физической топологии. Иногда логическая топология совпадает с физической. Логическая топология часто ассоциируется с *MAC* (*Media Access Control*) протоколом. Эта топология может динамически реконфигурироваться специальными устройствами типа роутеров или коммутаторов.

4.3. Протоколы локальных вычислительных сетей

4.3.1. Классификация локальных сетей

С понятием локальная вычислительная сеть (ЛВС) в настоящее время связывают главным образом следующие протоколы: *ATM, Ethernet, Token Ring, FDDI, SONET, Universal Serial Bus (USB), IEEE 1394 (Fire Wire), ARINC, IEEE 802.11 (Wireless Ethernet), Scalable Coherent Interface (SCI), IEEE 802.12 (100VG-AnyLAN), Fibre Channel, STANAG, HiPPI (High Performance Parallel Interface), IEEE 802.6, RACEway, Mil-Std-1553, MemoryChannel, CAN, World-FIP, PROFIBUS*.

Этот список может быть существенно продолжен. По области применения его можно разбить на сети административного уровня (*Ethernet, Token Ring*), промышленного уровня для обеспечения обменов между контроллерами, датчиками сигналов, исполнительными механизмами (*CAN, World-FIP*).

По популярности протоколы ЛВС условно можно разделить на три группы. Первая группа представлена протоколами *ATM, Ethernet, Token Ring, FDDI* и *SONET*, которые последние 5 лет являются базовыми при работе с ЛВС. *Ethernet* является основным носителем

информации в ЛВС, а *ATM* - основным протоколом связи ЛВС с телекоммуникациями. Частота упоминания этих двух протоколов в печати почти на порядок выше частоты упоминания всех остальных.

Остальные протоколы можно разделить на специальные и редкие протоколы, отнеся к первым протоколы, частота упоминания которых выше 0,1 %. К редким протоколам следует отнести *IEEE* 802.6, *RACEway*, *IEEE* 802.14, *IEEE* 1355, *MIL-STD*-1553.

Сейчас наблюдается тенденция падения интереса к ЛВС *Ethernet*, *Token Ring* и *FDDI* для ряда применений на фоне роста интереса к специальным интерфейсам - в первую очередь к *USB* и *SATA*.

Имеются ряд протоколов ЛВС с явно выраженной тенденцией к росту популярности у исследователей (*IEEE* 802.11, *MYRINET* и *IEEE* 802.14), протоколы с явно выраженной тенденцией падения популярности у исследователей (*IEEE* 802.6, *SCI* и *Fibre Channel*) и протоколов, популярность которых колеблется.

4.3.2. Типы и методы доступа

В основе построения ЛВС лежат два типа соединений между узлами сети: соединение точка-точка и широковещательное соединение. В свою очередь соединение может быть централизованным и децентрализованным. Соединение точка-точка состоит из одного передатчика на одном конце соединения и одного приемника на другом конце соединения. Широковещательное соединение объединяет несколько передающих и принимающих узлов, которые все подключены к одному и тому же разделяемому широковещательному каналу. При этом возникает проблема множественного доступа, когда необходимо скоординировать доступ множества посылающих и принимающих узлов к разделяемому широковещательному каналу. Протокол множественного доступа определяет, какому из посылающих узлов в какой момент предоставляется право передачи в широковещательном канале данных. Протоколы множественного доступа необходимы для организации работы различных вычислительных сетей, использующих проводную, беспроводную среду передачи данных либо спутниковые каналы связи.

Хотя с технической точки зрения каждый узел сети подключается к широковещательному каналу с помощью своего сетевого адаптера, в дальнейшем мы будем ссылаться на узел сети как на передающее и приемное устройство. Поскольку все узлы имеют способность передавать пакеты данных, может так случиться, что более чем два узла будут передавать одновременно. Этот случай называют коллизией, все пакеты, участвующие в коллизии, теряются и если многие модули хотят часто передавать пакеты, многие попытки передачи приведут к коллизиям, и большая часть полосы пропускания широковещательного канала будет потеряна.

Все протоколы множественного доступа могут быть разделены на три категории: протоколы разделения канала, случайного доступа и передачи права.

Рассмотрим общие характеристики протоколов множественного доступа. Идеальный протокол множественного доступа к широковещательному каналу со скоростью передачи сигнала *R* бит/сек должен был бы иметь следующие характеристики:

1. Когда только один узел передает данные, этот узел получает всю пропускную способность канала *R* бит/сек.

2. Когда *M* узлов имеют готовые к передаче данные, каждый из этих узлов должен получить пропускную способность *R/M* бит/с, или, говоря точнее, каждый из этих узлов должен получить среднюю пропускную способность канала *R/M* через некоторый подходящий интервал времени.

3. Протокол должен быть децентрализованным, т.е. не должно быть ведущего узла, который может выйти из строя и остановить всю систему.

4. Протокол должен быть простым, так чтобы его можно было использовать на практике при небольших затратах.

4.3.2.1. Протоколы с разделением канала

Предположим, что канал обеспечивает работу *N* узлов и имеет скорость передачи сигналов R бит/сек. Протоколы с разделением времени (*Time Division Multiplexing Protocol*, или *TDM*) разделяет время на временные пакеты и далее каждый временной пакет - на *N*

временных интервалов. Каждый временной интервал предоставляется одному из *N* узлов. Всякий раз, когда узел имеет готовые к передаче данные, он передает эти данные в отведенном для него временном интервале в ближайшем временном пакете. Обычно размер пакета данных выбирается таким образом, чтобы он был передан в одном временном интервале.

Протоколы с разделением времени исключают коллизии и являются совершенно простыми. Однако каждый узел имеет полосу пропускания *R/N* бит/сек и ждет ближайшего временного пакета даже в том случае, если он является единственно активным узлом в сети.

Протоколы с разделением частоты (*Frequency Division Multiplexing Protocol*, или *FDM*) делят пропускную полосу канала *R* бит/сек на *N* каналов различной частоты, каждый полосой *R/N* бит/сек, и предоставляют каждый канал одному узлу. Протоколам с разделением частоты присущ тот же недостаток, что и протоколу с разделением времени: узел получает полосу пропускания канала *R/N* бит/сек даже в том случае, если он является единственно активным узлом в сети.

Протоколы с разделением кода (*Code Division Multiple Access*, или *CDMA*) предоставляют каждому узлу уникальный код. Каждый узел затем использует этот код для декодирования данных.

4.3.2.2. Протоколы случайного доступа

При использовании протокола случайного доступа (*Random Access Protocol*) узел передает данные с максимальной скоростью *R* бит/сек. Если имеет место коллизия, передачу повторяют до тех пор, пока она не завершится успехом. При этом узел после коллизии не начинает передачу сразу, а ожидает случайный интервал времени.

Тактируемая *ALOHA* (с выделенным интервалом).

Положим следующее:

- все передаваемые пакеты данных содержат ровно *L* бит;
- время разделено на временные интервалы длительностью *L/R* сек, т.е. временной интервал равен времени передачи одного пакета данных;
- узлы начинают передачу пакетов только в начале временного интервала;

- узлы синхронизованы, т.е. каждый узел знает, когда начинается временной интервал;

- если два и более пакета сталкиваются в одном временном интервале, все узлы определяют коллизию прежде, чем временной интервал закончится.

Предположим, что имеется некоторая вероятность p, т.е. $0 \le p \le 1$. При использовании протокол «тактируемая *ALOHA*» каждый модуль выполняет следующие простые операции:

- когда узел получает новый пакет для передачи, он ждет начала следующего временного интервала и передает весь пакет;

- если имела место коллизия, узел определяет это до того, как временной интервал закончится. Узел повторяет передачу в каждом последующем временном интервале с вероятностью p до тех пор, пока передача не завершится успехом.

Тактируемая *ALOHA* имеет важные преимущества перед методами разделения канала: она позволяет единственному активному модулю непрерывно передавать пакеты с полной скоростью канала, она в высокой степени децентрализована и крайне проста. Недостатком метода является то, что, как показывает теоретический анализ, при большом количестве активных узлов пропускная способность канала сети не может быть использована более чем на $1/e$ (или около 37 %), и что необходима синхронизация временных интервалов в каждом узле.

ALOHA.

При использовании протокола *ALOHA* (или «чистая *ALOHA*») узел, получивший пакет для передачи в канал сети, начинает передачу немедленно, не дожидаясь временного интервала; если в результате имеет место коллизия, то повторная передача с вероятностью p начинается немедленно; следующая попытка передачи с вероятностью p повторяется через интервал времени, равный времени передачи пакета; таким образом, в протоколе «чистая *ALOHA*» не требуется синхронизовать узлы. Теоретический анализ показывает, что максимальная пропускная способность протокола при большом количестве активных узлов равна $R/2e$, т.е. ровно половина от пропускной способности тактируемой *ALOHA*.

CSMA/CD – Ethernet.

В 1980 г. была введена спецификация *IEEE* 802.3 *Ethernet*, определяющая механизм *CSMA/CD* - метод множественного доступа с контролем несущей и обнаружением конфликтов (*Carrier Sense Multiple Access* или *CSMA* и *CSMA with Collision Detection*). Этот механизм можно рассматривать как передачу информации станциями случайным образом и состязанием станций за время в среде. Схема обеспечивает то, что всем станциям предоставляется доступ на основе принципа «первый вошел - первый обслужен». Так как Ethernet был создан для перекачки данных, не было предусмотрено никакое обеспечение для качественного обслуживания или приоритета. Механизм *CSMA/CD* только определяет, что одни и те же правила доступа идентичны для всех абонентов сети.

Два главных принципа работы определяют семейство протоколов множественного доступа с контролем несущей и обнаружением конфликтов:

- получив пакет данных (кадр) для передачи в сеть, узел проверяет, свободен ли канал сети; если канал свободен, узел начинает передачу немедленно, если канал занят - ожидает его освобождения;
- если после начала передачи узел обнаруживает, что имеет место коллизия, передача прекращается и возобновляется через случайный интервал времени.

Эффективность протокола *Ethernet*, как показывает теоретический анализ, определяется выражением:

$f = 1/(1+ 5\ t_{prop}/t_{trans})$

где f - эффективность использования пропускной способности канала сети; t_{prop} - время распространения сигнала между двумя наиболее удаленными точками сети; t_{trans} - время передачи пакета данных.

Таким образом, максимальная эффективность метода *CSMA* значительно превышает эффективность ALOHA и тактируемая *ALOHA* и зависит от размера кадра и времени распространения.

Недостатком *Ethernet* является возможность клинча в канале ЛВС при перегрузке, при котором пропускная способность канала падает до нуля.

4.3.2.3. Протоколы с передачей права

Протоколы с передачей права можно разделить на две большие группы протоколов: протоколы с опросом и протоколы с передачей маркера. Протоколы с опросом требуют, чтобы один из узлов, который назначают диспетчером, циклически опрашивал остальные модули. Протоколы с опросом исключают коллизии и пустые временные интервалы, которые присущи протоколам случайного доступа. Это позволяет им обеспечивать более высокую эффективность. Недостатками являются потери времени на процедуру опроса и остановка всей сети при выходе из строя диспетчера.

Протоколы с передачей маркера не используют диспетчера. Для управления каналом сети служит специальный служебный пакет данных, называемый маркером, который передают от узла к узлу. Как показывает теоретический анализ, протоколы с передачей маркера обладают наивысшей эффективностью использования канала при высокой загрузке. Однако они являются неустойчивыми к сбоям в системе генерации маркера (возможно пропадание маркера или появление ложного).

4.3.3. Основные сетевые протоколы

4.3.3.1. *Asynchronous Transfer Mode (ATM)*

Впервые аббревиатура *ATM* прозвучала в 1991 г на конференции *Telecom'91* в Женеве. Разработчики - мировые лидеры на рынке телекоммуникации *Alcatel, Fujitsu, Northern Telecom, AT&T* и т.п. Корни технологии восходят, однако еще к концу 60-х годов, к исследовательским проектам *Bell Laboratory* в области асинхронного переключения каналов с разделением времени.

Обычно владельцы каналов предоставляют услуги с помощью *T*1 (1,544 Мбит/сек), *T*3 (44,736 Мбит/сек) и *OC*-3 с (149,76 Мбит/сек). Архитектура сети: коммутаторы *ATM*, соединенные транковыми каналами *ATM*. На входе коммутатора информация, поступающая из внешнего канала данных, разбивается на ячейки по 53 байта, 5 байт из которых составляют заголовок, а 48 - собственно информацию. Разработан очень большой набор интерфейсов, осуществляющих

стыковку коммутаторов *ATM* практически со всеми видами ЛВС: *Ethernet*, *Token Ring*, *FDDI* и т.п. Коммутаторы *ATM*, расположенные в центре сети, обеспечивают перенос ячеек, разделение транков и распределение потоков данных.

4.3.3.2. *Ethernet*

Наиболее распространенная в настоящее время ЛВС офисного применения. Создатель: *Robert Metcalfe* (1973г., корпорация *Xerox*). Первый вариант имел скорость передачи 2,944 Мбит/сек при передачи в коаксиальном кабеле. В 1977 с соавторами получил патент на сеть и в 1978 - на *HUB*. Объединенные усилия фирм *DEC, Intel* и *Xerox* (*DIX*) привели в 1980 к стандартизации протокола. Топология - шина. Метод доступа к среде передачи - *CSMA/CD* (*Carrier Sense Multiple Access with Collision Detection* - множественный доступ с контролем несущей и обнаружением конфликтов). Первоначально среда передачи - медный коаксиальный кабель. В настоящее время, как правило, используются витая пара и оптоволокно. Топология, оставаясь логически шинной, физически трансформировалась в звезду, а метод множественного доступа - в связь точка-точка. Классическая работа *F.Tobagi* показала основной недостаток сети - склонность к клинчам при загрузке более 30-40% от пропускной способности. Второй бедой метода доступа считаются блокировки, от которых не спасает даже разделение сети на отрезки точка-точка при использовании коммутаторов. Однако *Ethernet* соответствует этому методу только когда используется в полудуплексном режиме. В режиме полного дуплекса *Ethernet* уже не соответствует «духу» метода *CSMA*, т.к. нет необходимости контролировать несущую и делить среду с другими станциями. В этом случае уже никаких коллизий быть не может. Поэтому описанная блокировка возможна только в полудуплексном режиме (коаксиальный кабель или хаб). В режиме полного дуплекса блокировки (клинчи) отсутствуют.

Механизм возникновения блокировок показан на рис. 4.10. На рисунке условно изображены две станции А и Б, имеющие готовые к передаче пакеты. Если произошло столкновение, то станции делают следующую попытку передачи спустя несколько слотовых интервалов, выбираемых случайным образом в диапазоне 0-*n*, где *n*

определяется количеством конфликтов, имевших место при попытке передаче пакета:

n =	0	0 конфликтов
	1	1 конфликт
	3	2 конфликта
	7	3 конфликта
	15	4 конфликта

и т.д. в соответствии с протоколом *Ethernet.*

После первого конфликта станции находятся в равных условиях, однако выигравшая сторона получает существенное преимущество: ее счетчик конфликтов обнулился, в то время как у оппонента он увеличился на единицу. В результате в следующий раз вероятность того, что доступ к каналу сети получит станция А в 2 раза выше вероятности того, что доступ получил станция Б, после следующего конфликта - в 4 раза, затем - в 8 раз и так далее, в результате чего станция Б практически потеряет возможность доступа к сети пока станция А не завершит передачу всего своего блока данных.

Комитет *IEEE* разработал целый ряд стандартов *Ethernet*, различающихся скоростью передачи, расстоянием передачи и средой передачи данных. В табл. 4.5 приведены варианты стандарта *Ethernet* 10 Мб/с.

Табл. 4.5. Варианты сред физического уровня для стандарта *Ethernet IEEE* 802.3 10 Мб/с.

	10*BASE5*	10*BASE2*	10*BASE-T*	10*BASE-FP*
Среда передачи	Коаксиальный кабель (50 Ом)	Коаксиальный кабель (50 Ом)	Неэкранированная витая пара	Оптоволоконная пара 850 нм
Передача сигналов	Немодулированная (манчестерская кодировка)	Немодулированная (манчестерская кодировка)	Немодулированная (манчестерская кодировка)	манчестерская кодировка/релейная
Топология	Шинная	Шинная	Звездообразная	Звездообразная
максимальный размер сегмента (м)	500	185	100	1000
Число узлов на сегмент	100	30	-	33
Диаметр кабеля	10 мм	5 мм	0,4-0,6 мм	62,5/125 мкм

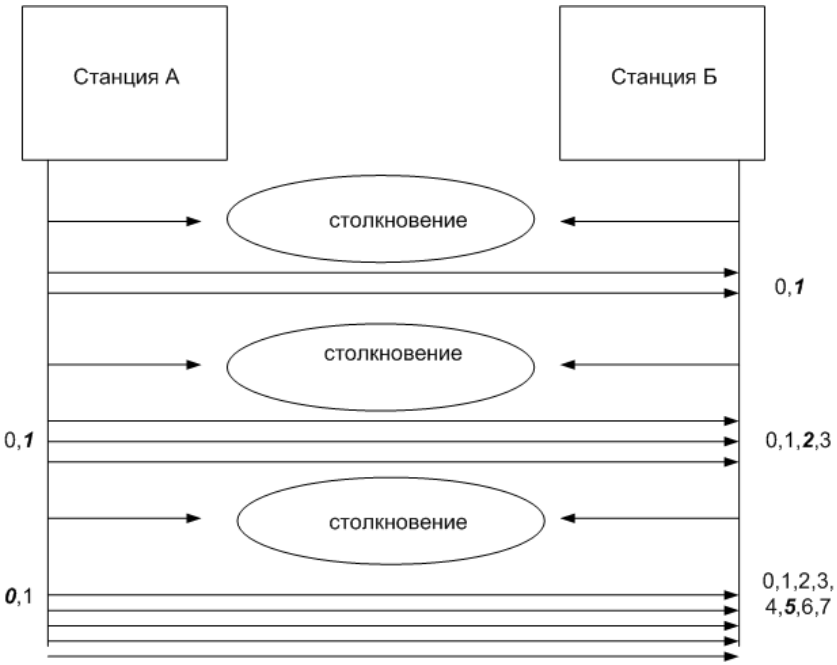


Рис. 4.10. Блокировки в *Ethernet*.

Длину сети можно увеличить за счет ретрансляторов. Так как ретранслятор не использует буферизацию, он не изолирует один сегмент кабеля от другого. Для избегания конфликтов между двумя станциями различных сегментов при попытке передавать одновременно между любыми двумя станциями разрешен только один маршрут распространения кадров, проходящий через сегменты и ретрансляторы. Стандарт позволяет использовать не более 4 ретрансляторов с максимальной эффективной длиной до 2,5 км.

Оптический стандарт 10*BASE-F* фактически включает 3 спецификации:

1. 10*BASE-FP* (пассивный). Топология пассивной звезды с длиной сегмента до 1 км.

2. 10*BASE-FL* (канал). Двухточечный канал для соединения станций или ретрансляторов на расстояние до 2 км.

3. 10*BASE-FB* (магистраль). Двухточечный канал для соединения станций или ретрансляторов на расстояние до 2 км.

Различия 10*BASE-FL* и 10*BASE-FB* в том, что первая асинхронная, а вторая синхронная, когда оптический сигнал, поступающий на ретранслятор, повторно синхронизируется по своим часам и затем уже передается. Благодаря синхронной передаче для 10*BASE-FB* возможно включение до 15 ретрансляторов.

При скорости передачи 10 Мб/сек максимальная удаленность станции в *Ethernet* 2 км. В этом случае еще возможно определение коллизии. Это расстояние определяется между временем, требуемым для передачи пакета минимального размера (64 байта) и возможностью зарегистрировать коллизию (ограничение, известное как задержка распространения “по кругу” - *round-trip*). Когда коллизия имеется, уровень *MAC* (*medium access control* - метод управления доступа к среде) регистрирует его и посылает сигнал остановки, заставляя станцию остановить передачу и затем возобновить ее вновь.

В 1994 г. был принят стандарт 802.3*u* (100*baseT*) в котором скорость передачи возросла до 100 Мбит/с. Увеличение скорости передачи в 10 раз обусловило то, что время, необходимое для передачи кадра, уменьшилось в 10 раз, что в свою очередь уменьшает расстояние с 2 км до 200 м.

Стандарт на ЛВС *Ethernet* со скоростью передачи данных 1 Гбит/сек 802.3z был принят комитетом *IEEE* в июне 1998. *Ethernet* 1 Гбит/сек поддерживает как полнодуплексный контроль потока с *IEEE* 802.3*x* операцией *PAUSE*, так и полудуплексные операции, использующие *CSMA/CD*. Для того чтобы обеспечить использование 100-м длины кабеля, метод доступа *CSMA/CD* был модифицирован. В него включены пункты об увеличенной длине блока сообщения и пакетной передаче блоков. Соответственно минимальная длина блока сообщения была увеличена с 64 байт до 512 байт, что требует от передатчика в случае получения слишком короткого блока сообщения добавлять пустые символы в конце блока, и передающая станция имеет право объединять блоки сообщений в единый пакет, пока таймер продолжительности пакета не будет исчерпан. Пакетная передача позволяет использовать в разделенном режиме

1 Гбит/сек *Ethernet CSMA/CD* и получать при этом удивительно высокую производительность, хотя задержки имеют тенденцию к быстрому росту при загрузке сети более 100 Мбит/сек.

Альтернативным методом обеспечения разделения доступа к каналу 1 Гбит/сек *Ethernet* является *Full-Duplex Repeater* (*FDR*). *FDR* является устройством, которое распределяет поступающие пакеты на все порты, как это делает обычный полудуплексный концентратор. Однако каждый порт, связанный с *FDR*, использует полнодуплексный протокол аналогично пакетному коммутатору. Таким образом, *FDR* представляет промежуточную точку между распределением полосы, распределенной диспетчеризацией (т.е. *CSMA/CD*) и умножением полосы, централизованной диспетчеризацией (т.е. switched Ethernet). *FDR* представляет распределенные полосы с центральной диспетчеризацией, позволяя многим абонентам разделять 1 Гбит/с полосу пропускания без коллизий.

Концентратор *Ethernet* в 10 Мбит/с и 100 Мбит/с направляет поток данных из любого порта во все другие порты. Концентратор не содержит функции *MAC* (контроля доступа к среде передачи) или буферизацию пакетов. Арбитраж передачи осуществляется абонентами с помощью распределенного полудуплексного протокола *CSMA/CD* и двоичного экспоненциального алгоритма отката. В *FDR* концентратор использует полнодуплексную функция *MAC*, буферизацию пакетов, и центральный арбитраж передачи. Для выбора следующего пакета из входного буфера используется либо механизм циклических приоритетов, либо взвешенного справедливого чередования, затем этот пакет направляется на все выходы. Если передачу одновременно ведут несколько абонентов, применяется *IEEE* 802.3*x* контроль для использования пропускной способности 1 Гбит/с с наибольшей эффективностью.

При использовании *FDR* необходим механизм внутреннего backpressure (с помощью увеличения времени опроса) с выходной на входную очередь для предотвращения переполнения входного буфера. Порог определяется в выходном буфере каждого порта и, если он превышен, следующее время опроса увеличивается на время, равное количеству бит на которое превышен порог, деленное на скорость поступления информации от порта. Это дополнительное время позволяет очистить буфер до порогового значения.

Если в одном из портов в очереди находятся маленькие по размеру блоки сообщения, а в другом - большие, то при большой загрузке очередь с большими блоками сообщения получит большую часть пропускной способности канала данных. Чтобы минимизировать эту диспропорцию и дать средство распределения полосы пропускания между портами, используется пакетная передача блоков сообщения. В этом случае порт может продолжать посылать блоки сообщения из своего входного буфера в шину данных ЛВС, пока счетчик переданных байтов, установленный на начальное значение *BURST_SIZE* и уменьшаемый с каждым переданным байтом, не достигнет нулевого значения или входной буфер не будет исчерпан. Этот механизм позволяет ограничить пиковое значение полосы пропускания, используемое портом, до любого заданного заранее значения.

Коммутируемый *Ethernet*, т.е. ЛВС *Ethernet*, в которой абоненты сети объединяются коммутатором, применяется и при скорости 10 Мбит/с, 100 Мбит/с, и 1000 Мбит/с. Этот вид Ethernet задан стандартом *IEEE* 802.1*D*, в котором введена функция прозрачного моста с фильтрацией пакетов на базе *MAC* адресов. Таким образом, коммутатор Ethernet должен содержать таблицы адресов, механизм заполнения таблицы и распознавания адреса и другие функции, необходимые для реализации стандарта *IEEE* 802.1*D*. Для передачи пакетов между портами в коммутаторе используется коммутационная фабрика (высокоскоростная магистраль), пропускная способность которого должна быть, по меньшей мере, равна сумме пропускных способностей портов коммутатора. Благодаря необходимости использования *MAC* функции, высокоскоростных буферов, высокоскоростной магистрали, адресных таблиц и механизмов распознавания стоимость такого коммутатора значительно выше, чем репитера.

Результаты моделирования и измерения на реальных стендах показывают, что обычный *Ethernet* показывает хорошее качество работы при загрузке до 30% от полной пропускной способности шины данных (что следует и из общих технических соображений), в то время как рассматриваемый метод обеспечивает высокое качество до 90% от полной пропускной способности шины данных.

В табл. 4.6 приведены варианты стандарта *Ethernet* 100 Мб/с, 1000 Мб/с и 10 Гб/с.

Табл. 4.6. Варианты сред физического уровня для стандарта *Ethernet* 100 Мб/с, 1000 Мб/с и 10 Гб/с.

Стандарт	Скорость передачи, Мбит/с	Топология	Среда передачи	Максимальная длина кабеля, м
100Base-TX	100	Звезда	Витая пара категории 5 (2 пары)	100
100Base-T4	100	Звезда	Витая пара категории 3, 4, 5 (4 пары)	100
100Base-FX	100	Звезда	Многомодовое или одномодовое оптоволокно	2000 многомодовый 15000 одномодовый
1000Base-T		Звезда	Витая пара категории 5 или выше	100
1000Base-T		Звезда	Кабель типа STP	25
1000Base-T		Звезда	оптоволокно	220-550
1000Base-T		Звезда	оптоволокно	550 многомодовый 5000 одномодовый
10GBase-XX	10000	Звезда	оптоволокно	300-40000 (зависит от типа кабеля и излучателя)

4.3.3.3. *Token Ring*

Сеть также относится к офисному типу сетей. Стандарт *Token Ring* был принят комитетом *IEEE* 802.5 в 1985 году. В это же время компания *IBM* приняла стандарт *Token Ring* в качестве своей основной сетевой технологии. Топология - кольцо. Право на доступ к среде передается циклически от станции к станции по логическому кольцу. Кольцо образуется отрезками кабеля, соединяющими соседние станции. Таким образом, каждая станция связана со своей предшествующей и последующей станцией и может непосредственно обмениваться данными только с ними. Для обеспечения доступа станций к физической среде по кольцу циркулирует кадр специального формата и назначения - маркер. Получив маркер, станция анализирует его, при необходимости модифицирует и при отсутствии у нее данных для передачи обеспечивает его продвижение к следующей

станции. Станция, которая имеет данные для передачи, при получении маркера изымает его из кольца, что дает ей право доступа к физической среде и передачи своих данных. Затем эта станция выдает в кольцо кадр данных установленного формата последовательно по битам. Переданные данные проходят по кольцу всегда в одном направлении от одной станции к другой.

Сети *Token Ring* работают с двумя битовыми скоростями - 4 Мб/сек и 16 Мб/сек. Первая скорость определена в стандарте 802.5, а вторая является новым стандартом де-факто, появившимся в результате развития технологии *Token Ring*. Смешение станций, работающих на различных скоростях, в одном кольце не допускается. Сети *Token Ring*, работающие со скоростью 16 Мб/сек, имеют и некоторые усовершенствования в алгоритме доступа по сравнению со стандартом 4 Мб/сек.

4.3.3.4. *USB*

USB (*Universal Serial Bus*) была разработана группой из семи компаний (*Compaq, Digital Equipment Corp., IBM PC Co., Intel, Microsoft, NEC* и *Northern Telecom*), которые видели необходимость во взаимодействии для обеспечения дальнейшего роста и развития расцветающей индустрии интегрированных компьютеров и телефонии. Основной целью стандарта является:

- Расширение функциональности компьютера. На момент начала разработки стандарта для подключения внешних периферийных устройств к персональному компьютеру использовалось несколько «традиционных» интерфейсов (*PS*/2, последовательный порт, параллельный порт, порт для подключения джойстика, *SCSI*) и с появлением новых внешних устройств разрабатывался и новый стандарт. Предполагалось, что *USB* заменит все эти и интерфейсы и облегчит разработку новых устройств.
- Подключить к компьютеру мобильный телефон. В то время телефоны переходили на цифровую передачу голоса, и ни один из имеющихся интерфейсов не годился для передачи с телефона на компьютер, как речи, так и данных.
- Простота для пользователя. Старые интерфейсы (например, *COM*- и *LPT*-порты) были крайне просты для разработчика, но не

позволяли простым образом подключать/отключать устройства. Требовались новые механизмы взаимодействия компьютера с низко- и среднескоростными внешними устройствами – надёжные и дружественные, позволяющие осуществлять «горячее подключение», то есть подключение без выключения питания.

Спецификация периферийной шины *USB* разработана для подключения компьютерной периферии вне корпуса машины по стандарту *plug-and-play*, в результате отпадает необходимость в установке дополнительных плат в слоты расширения и переконфигурировании системы. Персональные компьютеры, имеющие шину *USB*, позволяют подключать периферийные устройства и осуществляют их автоматическое конфигурирование, как только устройство физически будет присоединено к машине, и при этом нет необходимости перезагружать или выключать компьютер, а также запускать программы установки и конфигурирования. Шина *USB* позволяет одновременно подключать до 127 устройств к одному хост-контроллеру (ведущему контроллеру), таких, как мониторы или клавиатуры, выполняющие роль дополнительно подключенных компонентов, включая хаб (устройство, через которое подключается еще несколько). Для увеличения числа устройств требуется добавить дополнительный порт/хост.

Шина имеет понятие «главное устройство» (хост, он же *USB* контроллер, обычно встроен в микросхему южного моста на материнской плате) и «периферийные устройства [88]. На шине допустим один и только один хост. Спецификация *USB* сама по себе не поддерживает любую форму мультихостинга. Однако в спецификации *On-The-Go*, появившейся в стандарте *USB* 2.0, введен протокол *Host Negotiation Protocol*, который позволяет двум устройствам *USB* договориться, кто будет выполнять роль хоста. Это предназначено для одиночных подключений точка-точка, например, мобильный телефон – персональный компьютер, и не распространяется на хабы и конфигурации компьютеров. Хост *USB* ответственен за выполнения всех транзакций и выбора полосы пропускания.

Транзакция – это группа последовательных операций, которая представляет собой логическую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, соблюдая целостность данных и независимо от параллельно идущих других

транзакций, либо не выполнена вообще и тогда она не должна произвести никакого эффекта. В качестве транзакции можно привести пример передачи данных от главного к периферийному устройству.

Данные могут быть посланы методами различных транзакций, используя *token-based* протокол (протокол, основанный на символах). *USB* использует топологию многоярусная звезда (древовидная). Наверху находится хост, в системе он может быть только один. Для увеличения числа подключаемых устройств используется хаб, который создаёт следующий, более низкий, уровень. На этом уровне находятся исполнительные устройства и возможный хаб для создания следующего уровня. Число уровней ограничено 7. Это связано с временными характеристиками передачи, определёнными в стандарте для хаба и кабелей.

Соединение 2 компьютеров или 2 периферийных устройств пассивным *USB* кабелем невозможно. Существуют активные *USB* кабели для соединения 2 компьютеров, но они включают в себя сложную электронику, эмулирующую *Ethernet* адаптер, и требуют установки драйверов с обеих сторон.

Устройства могут быть запитаны от шины, но могут и требовать внешний источник питания. Поддерживается и дежурный режим для устройств и разветвителей по команде с шины со снятием основного питания при сохранении дежурного питания и включением по команде с шины.

USB поддерживает «горячее» подключение и отключение устройств. Это достигнуто увеличенной длиной заземляющего контакта разъёма по отношению к сигнальным. При подключении разъёма *USB* первыми замыкаются заземляющие контакты, потенциалы корпусов двух устройств становятся равны и дальнейшее соединение сигнальных проводников не приводит к скачкам напряжений, даже если устройства питаются от разных фаз силовой трёхфазной сети. Такой приём используется в самых различных шинах, в том числе и в *PCI*.

На логическом уровне устройство *USB* поддерживает транзакции приема и передачи данных. Каждый пакет каждой транзакции содержит в себе номер конечной точки (*endpoint*) на устройстве. При подключении устройства драйверы в ядре ОС читают с устройства список конечных точек и создают управляющие структуры данных для

общения с каждой конечной точкой устройства. Совокупность конечной точки и структур данных в ядре ОС называется каналом (*pipe*).

Конечные точки, а значит, и каналы, относятся к одному из 4 классов — поточный (*bulk*), управляющий (*control*), изохронный (*isoch*) и прерывание (*interrupt*). Низкоскоростные устройства, такие, как мышь, не могут иметь изохронные и поточные каналы.

Управляющий канал предназначен для обмена с устройством короткими пакетами «вопрос-ответ». Любое устройство имеет управляющий канал 0, который позволяет программному обеспечению ОС прочитать краткую информацию об устройстве, в том числе коды производителя и модели, используемые для выбора драйвера, и список других конечных точек.

Канал прерывания позволяет доставлять короткие пакеты и в том, и в другом направлении, без получения на них ответа/подтверждения, но с гарантией времени доставки — пакет будет доставлен не позже, чем через *N* миллисекунд. Канал используется, например, в устройствах ввода (клавиатуры/мыши/джойстики).

Изохронный канал позволяет доставлять пакеты без гарантии доставки и без ответов/подтверждений, но с гарантированной скоростью доставки в *N* пакетов на один период шины (1 кГц у *low* и *full speed*, 8 кГц у *high speed* режима). Используется для передачи аудио- и видеоинформации.

Поточный канал дает гарантию доставки каждого пакета, поддерживает автоматическую приостановку передачи данных по нежеланию устройства (переполнение или опустошение буфера), но не дает гарантий скорости и задержки доставки. Используется, например, в принтерах и сканерах.

Время шины делится на периоды, в начале периода контроллер передает всей шине пакет «начало периода». Далее в течение периода передаются пакеты прерываний, потом изохронные в требуемом количестве, в оставшееся время в периоде передаются управляющие пакеты и в последнюю очередь поточные.

Активной стороной шины всегда является контроллер, передача пакета данных от устройства к контроллеру реализована как короткий вопрос контроллера и длинный, содержащий данные, ответ устройства. Расписание движения пакетов для каждого периода

шины создается совместным усилием аппаратуры контроллера и ПО драйвера, для этого многие контроллеры используют *DMA* канал со сложной *DMA*-программой, формируемой драйвером.

Размер пакета для конечной точки находится в таблице конечных точек устройства констант и изменению не подлежит. Он выбирается разработчиком устройства из числа тех, что поддерживаются стандартом *USB*.

Имеется несколько спецификаций *USB*.

***USB* 1.0**

Спецификация выпущена в 1996 году.

Технические характеристики:

- два режима передачи данных:
  - режим с высокой пропускной способностью (*Full-Speed*) — 12 Мбит/с
  - режим с низкой пропускной способностью (*Low-Speed*) — 1,5 Мбит/с
- максимальная длина кабеля для режима с высокой пропускной способностью 5 м;
- максимальная длина кабеля для режима с низкой пропускной способностью 3 м;
- максимальное количество подключённых устройств (включая размножители) 127;
- возможно подключение устройств, работающих в режимах с различной пропускной способностью к одному контроллеру *USB*;
- напряжение питания для периферийных устройств 5 В;
- максимальный ток, потребляемый периферийным устройством 500 мА.

***USB* 1.1**

Спецификация выпущена в 1998 году. Исправлены проблемы и ошибки, обнаруженные в версии 1.0. Первая версия, получившая массовое распространение.

***USB* 2.0**

Спецификация выпущена в 2000 году.

USB 2.0 отличается от *USB* 1.1 введением режима *Hi-speed*.

Для устройств *USB* 2.0 регламентировано три режима работы:

- *Low-speed*, 10 - 1500 Кбит/с (используется для интерактивных устройств: клавиатуры, мыши, джойстики);

• *Full-speed*, 0,5 - 12 Мбит/с (аудио-, видеоустройства);

• *Hi-speed*, 25 - 480 Мбит/с (видеоустройства, устройства хранения информации).

***USB* 3.0**

Окончательная спецификация *USB* 3.0 появилась в 2008 году. Созданием *USB* 3.0 занимались компании *Intel, Microsoft, Hewlett-Packard, Texas Instruments, NEC* и *NXP Semiconductors*.

В спецификации *USB* 3.0 разъёмы и кабели обновлённого стандарта физически и функционально совместимы с *USB* 2.0. Кабель *USB* 2.0 содержит в себе четыре линии — пару для приёма/передачи данных, земля и питание. В *USB* 3.0 добавлены четыре линии связи (две витых пары). Новые контакты расположены отдельно в другом контактном ряду. Спецификация *USB* 3.0 повышает максимальную скорость передачи информации до 4,8 Гбит/с, что на порядок больше скорости *USB* 2.0. Таким образом, скорость передачи возрастает с 60 Мбайт/с до 600 Мбайт/с и позволяет передать 1 Тб не за 8-10 часов, а за 40 – 60 минут.

В версии *USB* 3.0 также увеличен максимальный ток единичного места (одна нагрузка) с 500 мА до 900 мА.

Новый разъём *USB* 3.0 *Powered-B* спроектирован с использованием двух дополнительных контактов, что позволяет предоставлять до 1000 мА другому устройству. Это позволяет избежать необходимости в источнике питания для устройств, подключаемого к *Wireless USB* адаптеру. При обычных проводных подключениях к хосту или хабу эти два дополнительных контакта не используются.

Скорости *USB*

В стандарте определены следующие скорости передач:

• *High Speed* – 480 Мбит/с

• *Full Speed* – 12 Мбит/с

• *Low Speed* - 1.5 Мбит/с

Топология *USB* многоярусная звезда или дерево (см. рис. 4.11) имеет некоторые преимущества перед простой топологией *daisy chain*. Первое – потребляемая мощность каждого устройства может отслеживаться и переключения, и перегрузки не влияют на работоспособность других устройств *USB*. Все *high, full* и *low speed* устройства могут поддерживаться одновременно. Хаб отфильтровывает

транзакции *high speed* и *full speed*, таким образом устройства с низкими скоростями не получают данные со слишком высокой скоростью.

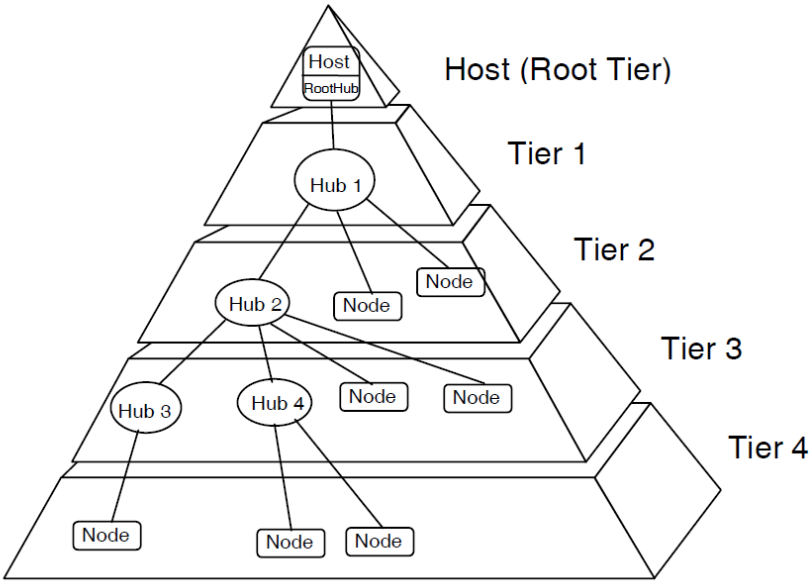


Рис. 4.11. Блокировки в *Ethernet*.

Контроллеры хоста *USB* имеют собственные спецификации. В стандарте *USB* 1.1 имеется две спецификации *Host Controller Interface*:

- *UHCI* (*Universal Host Controller Interface*) – спецификация, разработанная *Intel*, которая перекладывает большинство функций на программное обеспечение (*Microsoft*) и позволяет применять дешевую аппаратуру.
- *OHCI* (*Open Host Controller Interface*) – спецификация, разработанная *Compaq, Microsoft* и *National Semiconductor*, где больше функций возложено на аппаратуру (*Intel*), что упрощает программное обеспечение.

С появлением *USB* 2.0 понадобилась новая спецификация *Host Controller Interface Specification* для описания деталей регистрового уровня, специфичного для *USB* 2.0. Появился интерфейс *EHCI*

(*Enhanced Host Controller Interface*). Один стандарт интерфейса позволяет иметь только один драйвер для реализации в операционных системах.

USB последовательная шина. Она использует 4 экранированных провода, из которых два передают питание (+5 В и *GND*). Остальные два представляют витую пару дифференциальных сигналов данных. Используется схема кодирования *NRZI* (*Non Return to Zero Invert*, без возврата к нулю с инверсией - см. рис. 4.12) для передачи данных с полем синхронизации для синхронизации хоста и приемника.

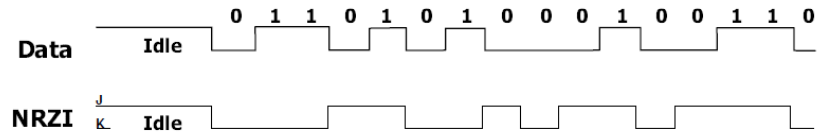


Рис. 4.12. Схема кодирования *NRZI*

USB поддерживает «горячее» (*plug-and-play*) соединение с динамически загружаемыми и выгружаемыми драйверами. Пользователь просто втыкает устройство, подключая его тем самым к шине. Хост детектирует присоединение, опрашивает установленное устройство и загружает требуемый драйвер. Когда пользователь закончил работу с *USB*-устройством, он просто вынимает его (или отсоединяет кабель), хост обнаружит отсутствие устройства и автоматически исключает его.

Загрузка требуемого драйвера осуществляется по комбинации PID/VID (*Product ID/Vendor ID*). *VID* предоставляется организацией *USB Implementor's forum* за деньги.

Другие организации по стандартам предоставляют дополнительные *VID* для некоммерческого использования, такого как обучение, разработка и прочее (радиолюбительство).

USB поддерживает *Control, Interrupt, Bulk* и *Isochronous* режимы передачи. Изохронный режим позволяет устройству резервировать определенную часть от полосы пропускания с гарантируемым временем задержки. Это идеально для аудио и видео приложений, где перегрузка канала может привести к заметной потере данных или

понижению частоты кадров. Каждый режим передачи предоставляет разработчику компромиссы по детектированию ошибок и восстановления, гарантированного времени ожидания и полосы пропускания.

Электрическая спецификация.

В стандарте *USB* используется дифференциальная пара для передачи данных с кодированием по принципу *NRZI* (см. рис. 4.13). В низкоскоростных и высокоскоростных устройствах дифференциальная ‘1’ передается установлением уровня *D*+ свыше +2,8 В при наличии резистора 15 кОм, подключенным к земле и уровня *D*- ниже +0,3 В при наличии резистора 1,5 кОм, подключенным к питанию 3,6 В (максимум, определённый в стандарте). Дифференциальный ‘0’ передается тем же способом, только уровни меняются на противоположные - *D*- свыше 2,8 В и *D*+ ниже 0,3 В при наличии тех же резисторов.

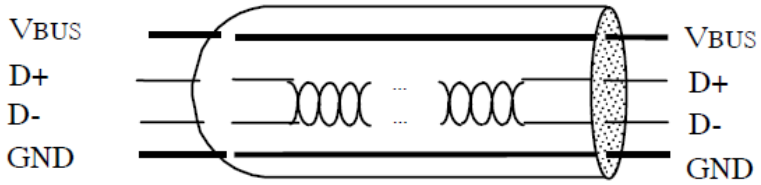


Рис. 4.13. Структура кабеля *USB*.

Приемник распознает дифференциальную ‘1’, если уровень *D*+ больше на 200 мВ, чем D-, и дифференциальный ‘0’, если *D*+ на 200 мВ меньше, чем *D*-. Полярность сигнала инвертируется в зависимости от скорости шины. Таким образом, термины состояний ‘*J*’ и ‘*K*’ состояний используются для обозначения логических уровней. В низкоскоростных передачах состояние ‘*J*’ – дифференциальный 0. В высокоскоростных передачах состояние ‘*J*’ – дифференциальная 1.

Трансиверы *USB* имеют как дифференциальные, так и одиночные выходы. Состояние шины определяется одиночными сигналами на *D* +, *D*- или на обеих линиях. Например, одиночный сигнал ноль или SE0 может использоваться для обозначения сброса устройства, если он удерживается дольше 10 мс. Сигнал *SE*0 генерируется путем удержания и *D*- и *D*+ на низком уровне (< 0.3 *V*).

Шина имеет волновое сопротивление 90 Ом +/- 15%.

Режим высокоскоростной передачи (480 Мбит/с) использует постоянный ток 17,78 мА для передачи сигналов с целью уменьшения шума.

Идентификация скорости.

Скорость устройства *USB* определяется подключением линии *D+* или *D-* к напряжению 3,3 В в момент конца подачи сигнала сброса. Для идентификации устройства как *full speed* использует *pull up* резистор, подключенный к *D+* (см. рис. 4.14), для low speed использует *pull up* резистор, подключенный к *D-* (см. рис. 4.15). Резисторы *pull up* на стороне устройства используются хостом или хабом для определения присутствия устройства на шине (подключено ли устройство в порт *USB*). Без *pull up* резистора считается, что к шине *USB* ничего не подключено. В некоторых устройствах этот резистор встроен в микросхему, в других требуется внешний резистор.

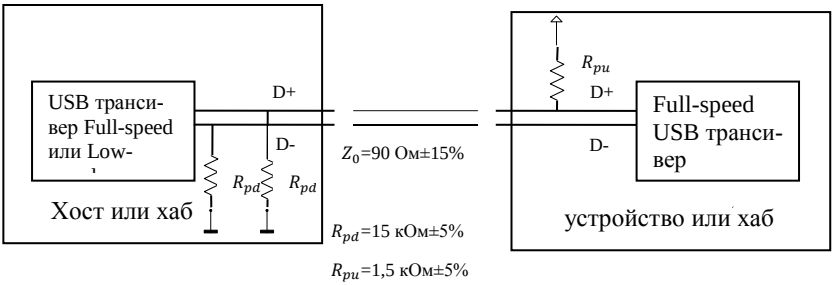


Рис. 4.14. *USB* устройство *Full Speed* (резистор подключен к *D+*).

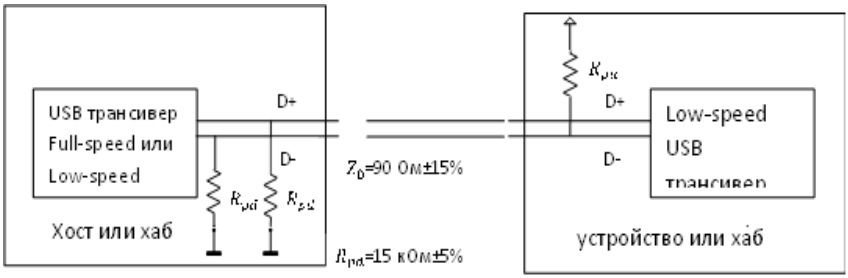


Рис. 4.15: *USB* устройство *Low Speed* (резистор подключен к *D*).

Резисторы R_{pd}подключены к земле, R_{pu} к питанию. После установления соединения и сброса устройство переходит в режим *high speed*, если хаб или хост поддерживает это. Если резистор работает в режиме *high speed*, резистор *pull up* отключается для сохранения баланса линии.

Для совместимости устройства с *USB* 2.0 не требуется поддержка режима *high-speed*. Это позволяет производить более дешевые устройства, если скорость не важна. Это также относится и к *USB* 1.1 low speed устройству, которое не обязано поддерживать *full speed*.

Однако высокоскоростное устройство не должно поддерживать режим низкой скорости. Оно должно поддержать режим *full speed* для первого соединения, а затем перейти в режим *high speed* после успешного взаимодействия. Хаб или хост по стандарту *USB* 2.0 должен поддерживать все три режима - *high speed, full speed* и *low speed*.

Питание (*VBUS*).

Устройства на шине *USB* могут получать питание от шины и не требовать никаких дополнительных кабелей.

Для *USB* устройств указывается энергопотребление в единицах «2 мА» в дескрипторе конфигурации. Устройство не может увеличить энергопотребление свыше указанной величины, даже если у него пропадет внешнее питание. Имеется 3 класса функций *USB*:

- *Low-power* устройства питаемые от шины,
- *High-power* питаемые от шины,
- *Self-powered* устройства (с собственным отдельным питанием).

Устройства Low power получают питание полностью от *VBUS* и не могут потреблять больше одной нагрузки. В спецификации *USB* одна нагрузка соответствует 100 мА.

Устройства *High power* получают питание только от *USB* и не могут потреблять больше одной нагрузки, пока они не будут сконфигурированы хостом, после чего они могут потреблять до 5 нагрузок (до 500 мА), предоставляемых в соответствии с величиной, установленной в дескрипторе.

Устройства с собственным питанием могут потреблять от шины до 1 нагрузки и получать остальное питание от внешнего источника.

При пропадании внешнего источника питания должна быть обеспечена возможность потребления от шины *USB* не больше одной нагрузки.

Протоколы *USB*.

Формат отправляемых данных *USB* использует до 7 уровней протоколов. Большинство контроллеров *USB* берет на себя функцию поддержки нижних уровней протоколов, делая их невидимыми для конечного разработчика.

Каждая транзакция *USB* состоит из:

- *Token Packet* (заголовок, показывающий что должно передаваться),
- необязательный *Data Packet* (содержит полезную информацию),
- *Status Packet* (используется для подтверждения транзакций и предоставления средств коррекции ошибок)

Хост начинает любую транзакцию. Первый пакет, называемый *token*, генерируется хостом для указания того, что следует, будет ли это транзакция чтения или записи, какой адрес устройства и конечная точка. Следующий пакет - пакет данных, за которым идет пакет *handshaking*, сообщающий о том, были ли данные и token приняты успешно и каково состояние конечной точки (остановлена, недоступна для принятия данных).

Общие поля пакета *USB*.

Данные на шине *USB* передаются в порядке, когда первым идет младший значащий бит (*LSB - Least Significant Bit*). Пакеты *USB* состоят из следующих полей:

- *Sync*

Все пакеты должны начинаться с поля синхронизации (*sync*). Поле sync имеет длину 8 разрядов для режимов *low speed* и *full speed* и 32 разряда для *high speed* и используется для синхронизации тактов приемника с тактами передатчика. Последние 2 разряда показывают начало поля *PID*.

- *PID*

PID означает *Packet ID* (идентификационный номер пакета). Это поле используется для обозначения типа отправляемого пакета. В таблице 4.7 показаны возможные значения этого поля.

Таблица 4.7. Значения поля *PID* типа отправляемого пакета.

Группа	**Значение *PID***	**Идентификатор пакета**
Token	0001	*OUT Token*
	1001	*IN Token*
	0101	*SOF Token*
	1101	*SETUP Token*
Data	0011	*DATA0*
	1011	*DATA1*
	0111	*DATA2*
	1111	*MDATA*
Handshake	0010	*ACK Handshake*
	1010	*NAK Handshake*
	1110	*STALL Handshake*
	0110	*NYET (No Response Yet)*
Special	1100	*PREamble*
	1100	*ERR*
	1000	*Split*
	0100	*Ping*

Под поле «*PID*» выделено 4 разряда, однако для обеспечения правильного приема дополнительно введены 4 повторяющегося разряда, то есть всего 8 разрядов. Полученный формат показан на рис. 4.16.

PID0	*PID1*	*PID2*	*PID3*	*nPID0*	*nPID1*	*nPID2*	*nPID3*

Рис. 4.16. Формат поля *PID*.

- *ADDR*

Поле адреса указывает, какому из устройств *USB* предназначен пакет. Адрес имеет длину 7 разрядов, что позволяет адресовать до 127 поддерживаемых одновременно устройств. Адрес 0 недопустим, он предназначен для устройств, адрес для которых пока не назначен. Любое устройство, которому не назначен адрес, должно отвечать на пакет с адресом 0.

- *ENDP*

Поле endpoint составлено из 4 разрядов, что позволяет иметь 16 конечных точек. Однако устройства *low speed* могут иметь только 2 дополнительные конечные точки сверх заданного по умолчанию канала (максимально 4 конечных точек).

- *CRC*

Разряды *CRC* (*Cyclic Redundancy Checks* - Циклическая Избыточная проверка) предназначены для обеспечения бессбойной передачи данных. Все пакеты *token* имеют 5 разрядов *CRC*, пакеты *data* - 16 разрядов *CRC*.

- *EOP*

Разряды *EOP* (*End of packet* - конец пакета) определяются несимметричным нулем (Single Ended Zero, SE0) двумя временными битами, далее следует 1 временной бит состояния J.

Типы пакетов *USB*

В стандарте *USB* определено 4 типа пакетов. Пакеты *token* индицируют тип последующей транзакции, пакеты *data* содержат полезную информацию, пакеты *handshake* используются для подтверждения данных или сообщений об ошибках и пакеты *start of frame* (*SOF*) показывают начало нового фрейма.

- пакеты *Token*

Имеется 3 типа пакетов *token*:

In – информируют устройство *USB* о том, что хост хочет прочитать информацию.

Out - информируют устройство *USB* о том, что хост хочет отправить информацию.

Setup – используется для начала управляющих передач (*control transfers*).

Пакеты token соответствуют формату, приведённому в рис. 4.17.

Sync	*PID*	*ADDR*	*ENDP*	*CRC5*	*EOP*

Рис. 4.17. Формат пакетов *token*.

- пакеты *Data*

Имеется 2 типа пакетов данных, каждый из которых может включать до 1024 байт данных:

Data0

Data1

Режим *High Speed* задает два других формата данных *PID* - *DATA2* и *MDATA*.

Пакеты *Data* имеют формат, представленный на рис. 4.18.

Sync	*PID*	*Data*	*CRC16*	*EOP*

Рис. 4.18. Формат данных *PID* режима *High Speed*.

Максимальный размер полезной информации для low-speed устройств составляет 8 байт. Максимальный размер полезной информации для *full-speed* устройств составляет 1023 байта. Максимальный размер полезной информации для *high-speed* устройств составляет 1024 байта. Данные должны посылаться кратные байтам.

- пакеты *Handshake*

Имеется 3 типа пакетов *handshake*, которые состоят только из *PID*.

ACK – подтверждение о том, что пакет успешно принят.

NAK – сообщение о том, что устройство временно не может отправить или принять данные. Также используется в транзакциях прерывания для информирования хоста о том, что нет никаких данных для передачи.

STALL – устройство находится в состоянии, которое требует вмешательства со стороны хоста.

Пакеты handshake имеют формат, представленный на рис. 4.19.

Sync	*PID*	*EOP*

Рис. 4.19. Формат пакетов *handshake*.

- пакеты *Start of Frame* (*SOF*)

Пакет *SOF* состоит из 11-разрядного номера фрейма, посылаемого хостом каждую миллисекунду (1 мс ± 500 нс) в *full speed* режиме или каждые 125 мкс (125 мкс ± 0.0625 мкс) в *high speed* режиме. На рис. 4.20 представлен формат пакета *SOF*.

Sync	*PID*	*Frame Number*	*CRC5*	*EOP*

Рис. 4.20. Формат пакетов *SOF*.

Функции *USB*

Устройство *USB* может быть периферийным устройством, *USB* трансивером в хосте или в периферии, *USB* хабом и микросхемой контроллера хоста. Большинство функций *USB* реализуется аппаратно в микросхемах (нижние уровни протокола).

У большинства функций есть серия буферов, обычно длиной 8 байт. Каждый буфер принадлежит конечной точке - *EP0 IN, EP0 OUT* и т. п. Например, хост отправляет запрос дескриптора устройства (*device descriptor request*). Функция аппаратно прочитает пакет

setup и определит из поля адреса, предназначен ли пакет именно ей, и если да, то скопирует полезную информацию передаваемого пакета данных в соответствующий буфер конечной точки. Затем отправится пакет *handshake* для подтверждения приема байта и сгенерируется внутреннее прерывание в микроконтроллере для соответствующей конечной точки, обозначающее, что пакет был принят.

Программное обеспечение получит прерывание, по которому оно должно прочитать содержимое буфера конечной точки и проанализировать запрос дескриптора устройства.

Конечные точки (*endpoints*).

Конечные точки могут быть описаны как источники или приемники данных. Поскольку шина является хосториентированной, конечные точки оказываются в конце канала связи. Например, на уровне программного обеспечения драйвер устройства может отправлять пакет в конечную точку *EP*1 устройства. Так как данные поступают от хоста, они попадут в *OUT* буфер *EP*1. Встроенное программное обеспечение (*firmware*) устройства теперь позволяет прочитать эти данные. Если устройство хочет вернуть данные, функция не может просто записать их на шину, так как шина полностью управляется хостом. Поэтому *firmware* помещает данные в буфер *EP*1 *IN*, и эти данные находятся в буфере до тех пор, пока хост не отправит пакет *IN*, которым он запрашивает данные конечной точки. Конечные точки можно также рассматривать как интерфейс между устройством и ПО.

Все устройства должны поддерживать конечную точку 0. Это конечная точка, которая принимает все управляющие запросы и запросы статуса в течение всего времени, когда устройство функционирует на шине.

Потоки, или каналы (*Pipes*).

Когда устройство отправляет и принимает данные на несколько конечных точек, клиентское программное обеспечение передает данные через каналы. Поток – это логическое соединение между хостом и конечной точкой (точками). Потоки имеют набор параметров, характеризующих их – какова полоса пропускания, тип передачи (*Control, Bulk, Iso* или *Interrupt*), направление потока данных и

максимальные размеры пакета/буфера. Например, поток по умолчанию – двунаправленный поток, составленный из *IN* конечной точки 0 и *OUT* конечной точки 0 с контролем типа передач.

USB определяет два типа потоков:

• *Stream Pipes* не имеют предопределенного *USB* формата, поэтому можно послать данные любого типа через stream pipe и восстановить данные на другом конце. Потоки данных последовательны и имеют предопределенную направленность – *IN* или *OUT*. Stream pipes поддерживают типы передач *bulk, isochronous* и *interrupt*. *Stream pipes* могут управляться либо от хоста, либо от устройства.

• *Message Pipes* имеют предопределенный *USB* формат. Они управляются хостом, инициируются запросом, отправляемым от хоста. Данные пересылаются в направлении, заданном в запросе. Таким образом, *message pipes* позволяют передавать данные в обоих контролируемых направлениях.

Типы конечных точек.

Стандарт *USB* определяет 4 типа передач/конечных точек:

• *Control*
• *Interrupt*
• *Isochronous*
• *Bulk*

Передачи *Control*

Передачи *Control* обычно используются для операций команд и состояния. Основная операция является инициализация *USB* устройства последовательностью функций с использованием передач *control*. Это обычно пакетные, случайные по времени пакеты, передающиеся хостом с наивысшим приоритетом доставки. Длина пакета для устройств *low speed* должна быть 8 байт, для *high speed* разрешены размеры пакета 8, 16, 32 или 64 байт, устройства *full speed* должны иметь размер пакета 64 байта.

Передачи *Interrupt*

Прерывания генерируются устройством для указания хосту необходимости взаимодействия с ним. Передачи *Interrupt* имеют следующие особенности:

- Гарантированное время ожидания
- Однонаправленный поток канала
- Определение ошибок и последующая повторная попытка передачи.

Передачи *Interrupt* обычно непериодические, когда *USB* устройство инициирует обмен, требующий ограниченного времени ожидания. Запрос на прерывание ставится в очередь, пока компьютер не опросит устройство *USB* с целью получения данных.

Максимальная длина полезных данных для *low-speed* устройств - 8 байт.

Максимальная длина полезных данных для *full-speed* устройств - 64 байта.

Максимальная длина полезных данных для *high-speed* устройств - 1024 байта.

Передачи *Isochronous*

Изохронные передачи выполняются периодически и непрерывно. Они обычно содержат информацию, передача которой требует ограничение по времени доставки, например, аудио или видео потоки. Если произойдет задержка или повторная передача данных в аудио потоке, то получится, например, искаженный звук. Может также пропасть синхронизация звука.

Изохронные передачи обеспечивают:

- Гарантированное выделение полосы пропускания шины *USB*.
- Ограниченную задержку.
- Однонаправленный поток канала.
- Детектирование ошибок с помощью *CRC*, но без гарантии и повторов доставки.
- Скорости передач *Full speed* и *high speed*.
- Нет переключения данных.

Максимальный размер полезной информации определяется в дескрипторе конечной точки. Этот размер может быть максимум до

1023 байт для *full speed* и до 1024 байт для *high speed* устройств. Поскольку максимальный размер полезных данных влияет на полосу пропускания шины, необходимо тщательно определить его размер.

Передачи *Bulk*

Bulk-передачи можно использовать для пакетных передач большого объема данных. В качестве примера можно привести задание по выводу на печать, посланное в принтер, или изображение, сгенерированное от сканера. *Bulk*-передачи обеспечивают коррекцию ошибок передаваемых данных с помощью поля *CRC*16 и механизма детектирования ошибок и повторной передачи, гарантирующие отсутствие ошибок в передаваемых или принимаемых данных. *Bulk*-передачи используют остаточную полосу пропускания шины после того, как все другие транзакции были распределены. Если шина занята данными *isochronous* и/или *interrupt*, то данные *bulk* могут передаваться через шину медленно. Следовательно, передачи *Bulk* должны использоваться только для интенсивных передач с негарантированным временем доставки. Особенности передачи *Bulk*:

- Используются для пакетных передач большого объема данных.
- Детектирование ошибок с помощью *CRC* с гарантией доставки.
- Не гарантируется полоса пропускания и минимальная задержка.
- Однонаправленный поток данных канала.
- Работает только на скоростях *Full speed* и *high speed*.

Bulk-передачи поддерживаются только *full speed* и *high speed* устройствами. Для конечных точек *full speed* максимальный размер пакета *bulk* бывает 8, 16, 32 или 64 байта. Для конечных точек *high speed*, максимальный размер пакета может до 512 байт длиной. Если размер передаваемых данных меньше максимального размера пакета, не требуется дополнение пакета нулями. *Bulk*-передачу считают законченной, когда она передала точное количество запрошенных данных, передала пакет меньше максимального размера конечной точки или передала пакет нулевой длины.

Дескрипторы *USB*

Все устройства *USB* имеют такой атрибут как дескриптор, являющийся данными с определённым форматом. Дескрипторы должны выдаваться устройством в ответ на стандартные запросы. Дескрипторы *USB* описывают информацию для хоста такую как, что это за устройство, кто его изготовил, какую версию *USB* поддерживает устройство, какими способами устройство может быть сконфигурировано, количество конечных точек и их типы и т. д.

Существуют следующие общие дескрипторы *USB*:

- Дескрипторы устройства
- Дескрипторы конфигурации
- Дескрипторы интерфейса
- Дескрипторы конечной точки
- Дескрипторы строки

Все дескрипторы имеют общий формат. Первый байт указывает длину дескриптора в байтах, второй байт показывает тип дескриптора. Если длина описателя будет меньшей, чем определено в спецификации, то главный компьютер должен проигнорировать его. Однако, если размер больше ожидаемого, хост будет игнорировать дополнительные байты, и будет искать следующий дескриптор в конце возвращенной действительной длины.

Дескриптор устройства *USB* представляет общую информацию об устройстве. Устройство *USB* может иметь только один дескриптор устройства. Дескриптор устройства включает в себя такую информацию, как поддерживаемую устройством ревизию *USB, Product ID* (*PID*, идентификатор продукта), *Vendor* ID (*VID*, идентификатор производителя), используемые для загрузки соответствующего устройству драйвера и возможных конфигураций устройства. Число конфигураций указывает, сколько имеется ответвлений по дескрипторам конфигурации. В таблице 4.8 представлена структура дескриптора устройства.

Таблица 4.8. Значения поля *PID* типа отправляемого пакета.

Смещ.	Название	Размер	Значение	Описание
0	bLength	1	число	Размер данного дескриптора в байтах
1	bDescriptorType	1	Константа	Тип дескриптора устройства
2	bcdUSB	2	BCD	Номер версии спецификации USB в двоично-десятичном коде (т.е. 2.10 представляется как 210H)
4	bDeviceClass	1	Класс	Код класса устройства, присваивается USB
5	bDeviceSubCIass	1	Подкласс	Код подкласса устройства USB (присваивается USB)
6	bDeviceProtocol	1	протокол	Код протокола USB, присваивается USB
7	bMaxPacketSize	1	число	Максимальный размер пакета для нулевой конечной точки (возможны значения 8, 16, 32 и 64)
8	idVendor	2	ID	Идентификатор производителя устройства (ID производителя), присваивается USB
10	idProduct	2	ID	Версия устройства
12	bcdDevice	2	BCD	Номер версии устройства
14	Manufacturer	1	Индекс	Индекс дескриптора строки, описывающий продукт
15	iProduct	1	Индекс	Индекс дескриптора строки (продукт)
16	SerialNumber	1	Индекс	Индекс дескриптора строки, описывающий серийный номер устройства
17	bNumConfigurations	1	Число	Количество возможных конфигураций

Устройство *USB* может иметь несколько различных конфигураций, хотя большинство устройств имеют только одну конфигурацию.

Дескриптор конфигурации указывает величину потребляемой мощности от шины, питается ли устройство от собственного источника либо от шины *USB* и количество интерфейсов, которые есть у конфигурации. Когда устройство идентифицируется, хост читает дескриптор устройства и принимает решение, какую конфигурацию

применить. Хост может разрешить только какую-то одну из конфигураций.

Например, возможно существование конфигурации с высоким потреблением мощности от шины, и есть конфигурация с собственным источником питания. Если устройство подсоединено к десктопу с сетевым блоком питания, драйвер может выбрать конфигурацию с высоким потреблением мощности от шины, несмотря на то, что устройство подключено к собственному источнику, а в случае подсоединения к ноутбуку драйвер может принудительно выбрать конфигурацию с собственным блоком питания, что потребует от пользователя обязательного подключения внешнего блока питания для *USB*-устройства.

Параметры настройки конфигураций не ограничены различиями мощности. Каждая конфигурация может быть настроена на одинаковое питание, но иметь разные интерфейсы или наборы конечных точек. Однако нужно отметить, что изменение конфигурации требует, чтобы вся деятельность по каждой конечной точке остановилась. В то время как *USB* предлагает эту возможность, у очень немногих устройств есть больше чем 1 конфигурация.

Дескриптор интерфейса можно рассматривать как заголовок или группирование конечных точек в функциональную группу, выполняющую единственную функцию устройства. Например, в устройстве факс со сканером дескриптор интерфейса 1 может описывать конечные точки функции факса, а дескриптор интерфейса 2 может описывать функцию сканера. В отличие от дескриптора конфигурации, здесь нет ограничений на количество одновременно разрешённых интерфейсов.

Дескрипторы интерфейса имеют поле *bInterfaceNumbe*r, указывающее номер интерфейса, и поле *bAlternateSetting*, которое разрешает интерфейсу поменять установки на лету. Например, имеется устройство с двумя интерфейсами – интерфейс 1 и интерфейс 2. У интерфейса 1 поле *bInterfaceNumbe*r установлено в 0, что показывает, что это первый дескриптор интерфейса, и поле *bAlternativeSetting*, установленное в 0. У интерфейса 2 поле *bInterfaceNumbe*r установлено в 1, что показывает - это второй интерфейс, и поле *bAlternativeSetting* установленное в 0 (значение по умолчанию). Можно также добавить

другой дескриптор, у которого поле *bInterfaceNumbe*r также установлено в 1 (показывает, что это второй интерфейс), но на этот раз поле *bAlternativeSetting* установлено в 1, что показывает – дескриптор интерфейса может иметь альтернативную настройку, взятую из другого описателя интерфейса 2.

Когда эта конфигурация разрешена, используются первые два дескриптора интерфейса, у которых поле *bAlternativeSettings* равно 0. Однако во время работы хост может послать запрос *SetInterface*, направленный на интерфейс с номером 1 с альтернативной настройкой 1, что разрешает другой дескриптор интерфейса.

Это дает преимущество при использовании двух конфигураций. Возможна передача данные через интерфейс 0, и в то же время менять настройки конечной точки, связанной с интерфейсом 1, не мешая работе интерфейса 0.

Дескрипторы конечной точки используются для описания конечных точек, отличных от конечной точки 0. Конечная точка 0 всегда используется как конечная точка управления, и она конфигурируется сразу автоматически перед запросом информации всех дескрипторов. Хост использует информацию, полученную из описателей конечных точек, чтобы определить требования по полосе пропускания шины. Каждый дескриптор конечной точки используется для указания типа передачи, направления, интервала опроса и максимального размера пакета для каждой конечной точки. Конечная точка 0 всегда подразумевается по умолчанию точкой для управления, и поэтому она не имеет дескриптора.

Дескрипторы строки предоставляют информацию в формате, удобном для чтения человеком, и их указывать необязательно. Если строковые дескрипторы не используются, поле индекса строки дескриптора должно быть установлено в 0, что указывает на отсутствие строкового дескриптора.

Строки закодированы в формате *Unicode*, и устройство *USB* может быть изготовлено с поддержкой многих языков. Строка с индексом 0 должна вернуть список поддерживаемых языков.

4.3.3.5. *IEEE 1394 Fire Wire Serial Bus*

Разработка начата специалистами компании *Apple*. Окончательный вариант стандарта одобрен в декабре 1995. Шина имеет древовидную структуру, вершина которой находится в “корневом” устройстве. Если этим корневым устройством является ПК, то адаптер шины содержит мост между *IEEE* 1394 и шиной *PCI*. Шина *IEEE* 1394 может иметь до 63 узлов. Максимальное количество соединенных шин в системе - 1023. Шина допускает “горячее” подключение. Устройство может подключаться к любому свободному порту (на каждом устройстве имеется обычно 1 - 3 порта). Каждый раз, когда устройство добавляется или исключается из сети, сеть автоматически переконфигурируется. Шины не должны образовывать петли. Длина стандартного кабели, соединяющего два узла, не должна быть более 4,5 м. В разъеме *IEEE* 1394 шесть контактов, сигналы передаются по двум экранированным витым парам; весь кабель в целом также экранирован. Стандарт поддерживает скорость передачи данных 100, 200 и 400 Мбит/с, причем на одной и той же шине одна пара устройств может обмениваться сигналами со скоростью 100 Мбит/сек, а другая - 400 Мбит/сек.

Протокол 1394 предусматривает два режима передачи данных по шине: асинхронный и изохронный. Оба имеют дело с передачей пакетов данных различной длины. В асинхронном режиме пакеты данных посылаются по адресам, заданным явным образом, и после каждого приема пакета назад возвращается подтверждение. Этот режим хорош в тех случаях, когда трафик не требует высоких скоростей или точной синхронизации, например для передачи какой-либо управляющей информации.

В изохронном режиме пакеты данных переменной длины передаются с регулярными интервалами ко всем частям шины без подтверждения. Это идеально для мультимедийных приложений, использующих видео и/или звук в реальном времени.

Протокол 1394 конкурирует с протоколом *USB*. Протокол *SBP*-2 шины *FireWire*/1394 имеет меньшие накладные расходы, чем протокол *USB storage*. Поэтому при подключении внешнего диска или привода *CD/DVD* по *FireWire* удаётся достичь большей скорости передачи данных.

Кроме того, протокол *USB* storage не поддерживался в старых ОС (включая *Windows* 98), и требовал установки драйвера. *SBP*-2 же поддерживался. Также в старых ОС (*Windows* 2000) протокол *USB* storage был реализован в урезанном виде, не позволяющем использовать функцию записи *CD*- и *DVD*-дисков на подключённом по *USB* дисководе, *SBP*-2 никогда не имел таких ограничений.

Шина *USB* строго ориентирована, потому соединение двух компьютеров или же двух периферийных устройств требует дополнительного оборудования. Некоторые производители поддерживают соединение принтера и сканера или же фотоаппарата и принтера, но эти реализации завязаны на конкретного производителя. Шина 1394/*FireWire* не подвержена этому недостатку (например, можно соединить две видеокамеры).

Тем не менее, ввиду лицензионной политики Apple, а также значительно более высокой сложности оборудования, 1394 менее распространён, материнские платы старых компьютеров не имеют контроллера 1394. Что касается периферии, то поддержка 1394 реализована во множестве корпусов для внешних накопителей на основе НЖМД (особенно премиум-сегмента) и приводов оптических дисков, мультимедиа интерфейсах, камкордерах.

Следует также отметить, что *Apple* использует в своих компьютерах порт 1394*b*, известный как *FireWire*800, пиковая скорость передачи данных которого 800 Мбит/сек.

4.3.3.6. *CAN*

Шина относится к промышленным шинам (*fieldbus*), была разработана фирмой *Bosh* в 1983-1985 годах для повышения интеллектуальности автомобилей и значительного уменьшения количества проводов в автомобиле. Основная идея заключалась в том, чтобы заменить соединение элементов друг с другом на сетевую систему с центральным процессором. Центральный процессор осуществляет пересылки сообщений таким образом, чтобы те элементы, которых это сообщение не касается, игнорировали его, либо выбирали единственное интересующее их сообщение, при этом получение сообщения не требовало возврата подтверждения (в реализации протокола

в самом сообщении может необходимость получения подтверждения быть задана специально). Эта большая свобода доступа позволила ограничить поток только самыми необходимыми сообщениями и, следовательно, гарантировать высокое качество.

Сеть *CAN* использовала протокол *CSMA/CD*, при этом обойдя проблему коллизий с помощью т.н. приоритетно-кодового арбитража:

- когда канал сети свободен, станции передают сообщения;
- если канал сети занят, станции дожидаются окончания передачи пакета по сети и начинают одновременную передачу своих идентификационных кодов.

Все идентификационные коды различаются. При этом станции следят за результатом передачи. Поскольку канал сети реализует функцию “ПРОВОДНОЕ ИЛИ”, в результате такого одновременного наложения кодов друг на друга лишь самый старший по значению код передается неискаженным. Станция, определившая, что ее код передан неискаженным, продолжает дальше передавать свое информационное сообщение, в то время как остальные станции прекращают передачу.

Ограничения автомобильной области вынуждают в реализации ориентироваться на шины сенсоров-приводов, тем самым сообщения не могут содержать более 8 бит. *Bosh* вошел в соглашение с *Intel* в реализации шины, и вместе они разработали БИС, реализующие все аспекты протокола. Начиная с этого момента значительное количество разработчиков начали производить по лицензии микросхемы, реализующие интерфейс *CAN*. Однако прикладной уровень интерфейса не определен, так что параметры реализации систем различных поставщиков могут различаться.

Характеристики шины:

- множество ведущих с множественным доступом;
- скорость до 1 Мбит/сек;
- переменная длина поля данных - до 28 байт;
- контрольная последовательность *CRC* 15 бит;
- сложность введения повторителей;
- ограниченный диаметр сети: 40 м при 1 Мбит/сек, 1 км при 50 кбит/сек;

• физическая среда передачи витая пара *RS*-485 (*ISO* 11519), предпочтительно экранированная; при этом используются, как правило, две линии передачи данных сразу. По одной линии передаются синхроимпульсы, в то время как по другой данные, что повышает надежность сети;

• возможность использования альтернативных сред передачи: оптическое волокно, ИК-излучение и т.д.;

• имеется система диагностики состояния линий передачи данных: нормальное состояние, короткое замыкание или обрыв;

• все сообщения имеют приоритеты;

• управление ошибками встроено в БИС;

• имеется несколько прикладных интерфейса стандарта;

• передачи имеют высокий уровень защиты;

• распространение по всему миру.

В настоящее время имеется значительное количество шин *fieldbus*. Все они более или менее стандартизованы, но ни одна из них не является *de facto* лидером, в противоположность компьютерным сетям, в которых безусловные лидеры явно очерчены. В этой области, кроме того, чувствуется влияние национализма: Германия имеет ряд своих шин, среди прочих *Profibus*, обладающей определенными преимуществами; Франция в некоторых системах предпочитает *FIP*; американцы пытаются со свойственной им агрессивной маркетинговой политикой внедрить *LON*. Из этой борьбы, которая ведётся последние несколько лет, пока ничего хорошего не получается, хотя появилась надежна на новое решение, которое будет скорее всего *TCP/IP* с доступом на нижнем уровне типа гнезда *BSD*. В пользу этого говорят следующие факты. *TCP/IP* широко используется в стандартных компьютерах, в частности в *IBM PC*, в последние годы индустриальные *PC* стали активно появляться в промышленности, далее, *TCP/IP* требует довольно высокой мощности от процессора, особенно если его предполагается использовать в различных приложениях, он требует относительно большой памяти. Последние два фактора были сдерживающими до недавнего времени, но благодаря значительному снижению цен в настоящее время этих препятствий больше не существует.

4.4. Основные типы коммуникационных сред

К наиболее распространенным высокопроизводительным коммуникационным средам следует отнести сети *HyperTransport, PCI Express, ASI, RapidIO, VXS, StarFabric, Ethernet 10 Gb, InfiniBand, Myrinet*. На мировом рынке существует разделение по применению этих коммуникационных сред в разных областях. В таблице 4.9 тёмным фоном выделены основные области применения указанных сред.

Таблица 4.9 Основные области применения различных коммутационных сред

Тип КС	внутримодульная связь	межмодульная связь	SAN	LAN
Hyper Transport	+			
PCE Express	+			
ASI		+	+	
RapidIO Parallel	+	+		
RapidIO Serial	+	+	+	
VXS		+	+	
StarFabric	+	+	+	
Ethernet 10Gb		+	+	+
InfiniBand			+	+
Myrinet			+	+

Из данной таблицы видно, что только коммутационные среды *StarFabric* и *RapidIO* ориентированы для внутримодульных, межмодульных и межмашинных передач данных (*SAN*). Таким образом, эти стандарты в наибольшей степени позволяют унифицировать многопроцессорные системы. Однако сеть *RapidIO* имеет лучшие скоростные данные, и она становится одной из основных сред для встроенных применений, включая военные и промышленные применения.

Сформулируем основные требования, предъявляемые к коммутационным средам. Задача состоит в построении наиболее высокопроизводительной и масштабируемой многопроцессорной системы при наличии жестких требований по надежности и потребляемой мощности. Дополнительным требованием является масштабируемость и модернизируемость системы при возможности использования модулей сторонних производителей. Для этого необходимо:

• наличие открытого стандарта;

• пакетная модель передачи данных;

• гибкая и масштабируемая топология среды (возможно применение различных типов коммутационных сред на разных архитектурных уровнях);

• низкая латентность (задержка доставки данных) при пересылке данных;

• наличие механизмов управления трафиком;

• аппаратное детектирование и исправление ошибок.

В ходе эволюции компьютерных систем было создано множество магистралей различного типа и систем на их основе, однако многие из них уже устарели и поддерживаются лишь для обеспечения совместимости. Однако ни один из этих интерфейсов не покрывает всех уровней архитектуры вычислительных систем. Рассмотрим основные сетевые среды.

4.4.1. *HyperTransport (HT)*

HyperTransport (*HT*) является двунаправленной последовательно/параллельной коммуникационной средой с высокой пропускной способностью и малыми задержками. Для разработки и продвижения данной шины был образован консорциум *HyperTransport Technology*. Технология используется целым рядом ведущих компаний компаниями: *AMD* и *Transmeta* в *x*86-процессорах; *PMC-Sierra, Broadcom* и *Raza Microelectronics* — в микропроцессорах *MIPS; nVidia, VIA, SiS, ULi/ALi, AMD, Apple Computer и HP* — в наборах системной логики для ПК; *HP, Sun Microsystems, IBM и iWill* — в серверах; *Cray, Newisys и PathScale* — в суперкомпьютерах, а также компанией *Cisco Systems* — в маршрутизаторах.

HyperTransport функционирует на частотах от 200 МГц до 3,2 ГГц, данные принимаются по обоим фронтам тактирующего сигнала, что позволяет осуществлять до 5200 миллионов посылок в секунду при частоте сигнала синхронизации 2,6 ГГц. Частота сигнала синхронизации настраивается автоматически.

HyperTransport поддерживает автоматическое определение ширины шины от 2-х до 32 разрядов. При 32 разрядах шина в двунаправленном режиме способна обеспечить пропускную способность до 41 600 Мбайт/с=2(DDR)×2×32/8(байт)×2600 (МГц) (максимум в одном направлении — 20 800 Мбайт/с), являясь, таким образом, одной из наиболее производительных шин. Шина может быть использована как в подсистемах с высокими требованиями к пропускной способности (оперативная память и ЦПУ), так и в подсистемах с низкими требованиями (периферийные устройства).

Шина *HyperTransport* основана на передаче пакетов. Каждый пакет состоит из 32-разрядных слов, вне зависимости от физической ширины шины (количества информационных линий). Первое слово в пакете — всегда управляющее слово. Если пакет содержит адрес, то последние 8 бит управляющего слова сцеплены со следующим 32-разрядным словом, в результате образуя 40-разрядный адрес. Шина поддерживает 64-разрядную адресацию — в этом случае пакет начинается со специального 32-разрядного управляющего слова, указывающего на 64-разрядную адресацию, и содержащего разряды адреса с 40 по 63 (разряды адреса нумеруются, начиная с 0). Остальные 32-разрядные слова пакета содержат непосредственно передаваемые данные. Данные всегда передаются 32-разрядными словами, вне зависимости от их реальной длины (например, в ответ на запрос на чтение одного байта по шине будет передан пакет, содержащий 32 разряда данных и флагом-признаком того, что значимыми из этих 32 разрядов являются только 8).

Пакеты *HyperTransport* передаются по шине последовательно. Увеличение пропускной способности влечёт за собой увеличение ширины шины. *HyperTransport* может использоваться для передачи служебных сообщений системы, для передачи прерываний, для конфигурирования устройств, подключённых к шине и для передачи данных.

Операция записи на шине бывает двух видов — *posted* и *non-posted*. *Posted*-операция записи заключается в передаче единственного пакета, содержащего адрес, по которому необходимо произвести запись, и данные. Эта операция обычно используется для обмена данными с высокоскоростными устройствами, например, для *DMA*-

передачи. *Non-posted* операция записи состоит из посылки двух пакетов: устройство, инициирующее операцию записи, посылает устройству-адресату пакет, содержащий адрес и данные. Устройство-адресат, получив такой пакет, проводит операцию записи и отсылает устройству-инициатору пакет, содержащий информацию о том, успешно ли произведена запись. Таким образом, *posted*-запись позволяет получить максимальную скорость передачи данных (нет затрат на пересылку пакета-подтверждения), а *non-posted*-запись позволяет обеспечить надёжную передачу данных (приход пакета-подтверждения гарантирует, что данные дошли до адресата).

Шина *HyperTransport* поддерживает технологии энергосбережения, а именно *ACPI*. Это значит, что при изменении состояния процессора (*C-state*) на энергосберегающее, изменяется также и состояние устройств (*D-state*). Например, при отключении процессора жёсткие диски также отключаются.

Электрический интерфейс *HyperTransport/LDT* — низковольтные дифференциальные сигналы, с напряжением 1,2 В.

Шина *HyperTransport* нашла широкое применение в основном в качестве системной шины микропроцессора, к которой подключаются системные контроллеры и интерфейсные схемы. Это связано с тем, что современные технологии производства пластин не позволяют иметь достаточного числа напряжений питания для периферийных (ввода/вывода) элементов. Так, для норм 45 нм невозможно одновременное питание периферийных элементов сетевых устройств (обычно 1,8 В) и шины *PCI* (3,3 В). Для подключения с шиной *PCI* используются дополнительные устройства для сопряжения шины процессора с шиной периферийных устройств (мосты). Данные адаптеры обычно включают в специализированные наборы системной логики, называемые северный мост и южный мост. Компании, использующие одну системную шину, имеют возможность использовать и одни и те же мостовые схемы. Однако в настоящее время нет единого стандарта на системную шину микропроцессора. В качестве такой системной шины может использоваться одна из рассматриваемых шин.

Другое применение *HyperTransport* — использование для создания многопроцессорных систем в рамках одной печатной платы. Как правило, в таких системах процессоры на одной плате связаны

между собой средой *HyperTransport*, а внешние связи осуществляются в другой среде.

Шина *HyperTransport* эффективно может использоваться для связи процессора с сопроцессором вследствие своей высокой пропускной способности. Для этих целей был разработан разъём для подключения сопроцессоров по шине *HyperTransport*, получивший название *HTX* (англ. *HyperTransport eXpansion*), и использующий разъём, механически совместимый с тем, который используется для подключения устройств 16*x PCI Express*. Использование разъёма *HTX* позволяет подключать карту расширения для обмена данными с процессором, а также осуществлять DMA доступ к системному ОЗУ.

В консорциум *HyperTransport* входят такие компании, как *Advanced Micro Devices (AMD), Alliance Semiconductor, Apple Computer, Broadcom Corporation, Cisco Systems, NVIDIA, PMC-Sierra, Sun Microsystems, Transmeta*. Он управляет спецификациями HyperTransport, проводит новые разработки и продвижение стандарта.

Однако средства управления трафиком в стандарте ограничены - нет поддержки потоков различных типов. “Горячее” подключении/отключение не поддерживается.

4.4.2. *PCI Express. ASI*

Шина *PCI Express*, сокращённое название *PCIe*, или *PCI-e* — компьютерная шина, использующая программную модель шины *PCI* и высокопроизводительный физический протокол, основанный на последовательной передаче данных. Развитием стандарта *PCI Express* занимается организация *PCI Special Interest Group*. В отличие от шины *PCI*, использовавшей для передачи данных общую шину, *PCI Express* является пакетной сетью с топологией типа звезда. Устройства *PCI Express* взаимодействуют между собой через среду, образованную коммутаторами, при этом каждое устройство напрямую связано соединением типа точка-точка с коммутатором.

Кроме того, шиной *PCI Express* поддерживается:

- горячая замена карт;

• гарантированную полосу пропускания (*QoS*);
• управление энергопотреблением;
• контроль целостности передаваемых данных.

Для подключения устройства *PCI Express* используется двунаправленное последовательное соединение типа точка-точка, называемое линией; это резко отличается от *PCI*, в которой все устройства подключаются к общей 32(64)-разрядной параллельной двунаправленной шине.

Соединение (англ. *link* — связь, соединение) между двумя устройствами *PCI Express* состоит из одной (*X*1) или нескольких (*X*2, *X*4, *X*8, *X*12, *X*16 и *X*32) двунаправленных последовательно соединённых линий. Каждое устройство должно поддерживать, по крайней мере, соединение с одной линией (*X*1).

На электрическом уровне каждое соединение использует низковольтную дифференциальную передачу сигнала (*LVDS*), приём и передача информации производится каждым устройством *PCI Express* по отдельным двум проводникам, таким образом, в простейшем случае, устройство подключается к коммутатору *PCI Express* всего лишь четырьмя проводниками.

Использование подобного подхода имеет следующие преимущества:

• карта *PCI Express* помещается и корректно работает в любом посадочном месте (слоте) той же или большей пропускной способности (например, карта *X*1 будет работать в слотах *X*4 и *X*16);

• слот большего физического размера может использовать не все линии (например, к слоту *X*16 можно подвести проводники передачи информации, соответствующие *X*1 или *X*8, и всё это будет нормально функционировать; однако, при этом необходимо подключить все проводники питания и заземления, необходимые для слота *X*16).

В обоих случаях, на шине *PCI Express* будет использоваться максимальное количество линий, доступных как для карты, так и для слота. Однако это не позволяет устройству работать в слоте, предназначенном для карт с меньшей пропускной способностью шины *PCI*

Express. Например, карта *X*4 физически не поместится в стандартный слот *X*1, несмотря на то, что она могла бы работать в слоте *X*4 с использованием только одной линии.

PCI Express пересылает всю управляющую информацию, включая прерывания, через те же линии, что используются для передачи данных. Последовательный протокол никогда не может быть заблокирован, таким образом задержки шины *PCI Express* вполне сравнимы с таковыми для шины *PCI* (заметим, что шина *PCI* для передачи сигнала о запросе на прерывание использует отдельные физические линии *IRQ#A, IRQ#B, IRQ#C, IRQ#D*).

Во всех высокоскоростных последовательных протоколах (например, в гигабитном *Ethernet*) информация о синхронизации должна быть встроена в передаваемый сигнал. На физическом уровне, *PCI Express* использует метод канального кодирования 8*b*/10*b* (8 бит в десяти, избыточность — 20%) для устранения постоянной составляющей в передаваемом сигнале и для встраивания информации о синхронизации в поток данных. В *PCI Express* 3.0 используется более экономное кодирование 128*b*/130*b* с избыточностью 1,5 %.

Некоторые протоколы (например, *SONET/SDH*) используют метод, который называется скремблинг (англ. scrambling) для встраивания информации о синхронизации в поток данных и для "размывания" спектра передаваемого сигнала. Спецификация *PCI Express* также предусматривает функцию скремблинга, но скремблинг *PCI Express* отличается от такового для *SONET*.

Битрейт (максимальная скорость прохождения информации) в *PCIe* 1.0 составляет 2,5 Гбит/с, в *PCIe* 2.0 - 5 Гбит/с. Для расчёта пропускной способности шины необходимо учесть дуплексность и избыточность 8*b*/10*b* (8 бит в десяти). Например, дуплексная пропускная способность соединения *X*1 составляет:

2,5 × 2 × 0,8 = 4 Гбит/с

где 2,5 — битрейт, Гбит/с; 2 — учёт дуплексности (двунаправленности); 0,8 — учёт избыточности 8*b*/10*b* для 1.0 и 2.0; 0,985 — для 3.0;

Шина *PCI Express* создавалась для использования только в качестве локальной шины. Так как программная модель *PCI Express* во

многом унаследована от *PCI*, то существующие системы и контроллеры могут быть доработаны для использования шины *PCI Express* заменой только физического уровня без доработки программного обеспечения. Высокая пиковая производительность шины PCI Express позволяет использовать её вместо шин *AGP* и тем более *PCI* и *PCI-X*.

В 2007 г. группа *PCI-SIG* выпустила спецификацию *PCI Express* 2.0. Основные нововведения в *PCI Express* 2.0 следующие:

- Увеличенная пропускная способность. Спецификация *PCI Express* 2.0 определяет максимальную пропускную способность одной линии в 5 Гбит/с, при этом сохранена совместимость с *PCI Express* 1.1. Внесены усовершенствования в протокол передачи между устройствами и программную модель. Таким образом, плата расширения, поддерживающая стандарт *PCI-e* 1.1 может работать, будучи установленной в слот *PCI-e* 2.0. Устройства же с интерфейсом *PCI Express* 2.0 смогут работать в материнских платах, оснащённых слотом *PCI Express X*16 поколения PCI Express 1.*x*, но только на скорости 2,5 Гбит/с. Это вполне закономерно, ведь старый системный контроллер (чипсет) не может поддерживать удвоенную скорость передачи данных.
- Динамическое управление скоростью (для управления скоростью работы связи).
- Оповещение о пропускной способности (для оповещения ПО об изменениях скорости и ширины шины).
- Расширение функций управляющих регистров для лучшего управления устройствами.
- Службы управления доступом — опциональные возможности управления транзакциями точка-точка.
- Управление таймаутом выполнения.
- Сброс на уровне функций — опциональный механизм для сброса функций внутри устройства.
- Переопределение предела по мощности (для переопределения лимита мощности слота при присоединении устройств, потребляющих большую мощность).

Внешняя кабельная спецификация *PCIe* позволяет использовать кабели длиной до 10 метров, работающие с пропускной способностью 2,5 Гбит/с.

По физическим характеристикам *PCI Express*

2.1 (скорость, разъём) соответствует 2.0, в программной части добавлены функции, которые в полной мере планируют внедрить в версии 3.0. Так как большинство системных плат продаются с версией 2.0, наличие только самой видеокарты с 2.1, не позволяет задействовать данный режим.

В 2010 году были утверждены спецификации версии *PCI Express* 3.0. Интерфейс обладает скоростью передачи данных *8 GT/s* (Гигатранзакций/с). Но, несмотря на это, его реальная пропускная способность была увеличена вдвое по сравнению со стандартом *PCI Express* 2.0. Этого удалось достигнуть благодаря другой схеме кодирования 128*b*/130*b*, когда 128 бит данных пересылаемых по шине кодируются 130 битами. *PCI Express* 2.0 обладает скоростью передачи данных 5 *GT/s* и схемой кодирования 8*b*/10*b*. При этом сохранилась совместимость с предыдущими версиями *PCI Express*. Летом 2011 года *Gigabyte* официально представила материнскую плату *G*1.*Sniper* 2, построенную на чипсете *Intel Z*68 и поддерживающую интерфейс *PCI Express* 3.0. *PCI Special Interest Group* (*PCI SIG*) заявила, что *PCI Express* 4.0 может быть стандартизирован до 2015 года. Он будет иметь пропускную способность 16 *GT/s* или более, то есть будет в два раза быстрее *PCIe.*

Текущий стандарт *PCI Express* занимает ту же нишу, что и *Hyper Transport*. Однако гибкость коммуникационной среды *PCI Express* хуже, основной поток данных обеспечивает связи центрального процессора с памятью и периферией, а обратный поток и межпроцессорный обмен оказываются затруднены. В таком виде он не удовлетворяет требованиям, заявленным выше. Интерес представляет развитие этого стандарта - *ASI* (*Advanced Switching Infrastructure*). В рамках нового стандарта предполагается переделать *PCI Express*, убрав из него атавизмы *PCI* и дополнить стандарт механизмами межмодульной и межмашинной связи. Преемственность программного обеспечения теряется, но при этом *ASI* становится единой КС высокопроизводительных систем. Средства управления трафиком

присутствуют в полном объеме, “горячее” подключение поддерживается. Однако зачастую конкурентную борьбу выигрывал не лучший стандарт с точки зрения инженерных решений, а стандарт, который стали поддерживать ведущие компании по ряду экономических и политических причин. Распространённость стандарта *PCI Express* и невысокая стоимость коммутаторов и адаптеров привела к тому, что на базе него стали создавать высокопроизводительные комплексы вплоть до суперЭВМ.

4.4.3. *Ethernet 10Gbit*

В предыдущей главе уже рассматривался стандарт *Ethernet*. Изначально он разрабатывался как локальная сеть. Однако характеристики стандарта *Ethernet* 10*Gbit* позволяют его одновременно отнести к сетям *LAN* и *SAN*. Самая известная сеть среди всех перечисленных. За прошедшее время была проделана большая “работа над ошибками”, в результате был создан стандарт, описывающий сеть, приближающийся по характеристикам к традиционным суперкомпьютерным сетям. Обработка стека протоколов *TCP/IP* выполняется аппаратно - эта технология получила название ускоритель *TCP/IP*. В итоге производительность сети стала достаточна для приложений с высоким трафиком, но из-за большого заголовка сеть не очень эффективно использует канал связи. Максимальная скорость передачи данных - 10 Гбит/с (4*X*). Основной недостаток сети – большое время задержки на начало передачи пакета, то есть обмен сообщениями в сети крайне неэффективен.

4.4.4. *VME extensions. VXS*

Разработка стандарта ведется в рамках проекта *OpenVME* компанией *Fujitsu*. *VXS* является очередным витком развития VME – он дополняет существующий *VME* стандарт последовательными высокоскоростными каналами, на базе которых строится коммуникационная среда. При этом используются уже готовые каналы от *HyperTransport, InfiniBand, PCI Express, RapidIO* и *StarFabric*, а стандарт оговаривает лишь использование этих каналов. Такое решение про-

диктовано необходимостью обеспечить совместимость устаревающих систем с новым оборудованием. Поэтому пропускная способность этих каналов ограничивается величиной 1 Гбит/с, но все остальные свойства этих каналов поддерживаются.

4.4.5. *StarFabric*

StarFabric разработан компанией *StarGen* и является открытым стандартом. *StarFabric* является высокопроизводительной средой систем обработки данных, при этом организация потоков данных несложная. Технология *StarFabric* проектировалась для поддержки семи классов трафика, включая асинхронный, изохронный, широковещательный, а также предоставление ресурсов в приоритетном порядке. Ранее для каждого из типов трафика конструкторам приходилось разрабатывать отдельную систему межсоединений.

Шлюзу для передачи голоса поверх *IP* (*voice-over-IP*) сегодня нужны три независимые шины: для трафика в режиме уплотнения по времени, для пакетного трафика и для управляющего трафика. С помощью *StarFabric* каждому модулю коммуникационной системы можно предоставить один или несколько мостов в зависимости от тех типов трафика, которые этот модуль должен поддерживать. Эти мосты (например, *PCI-to-StarFabric* или *H*.110-*to-StarFabric*) подключаются к двум резервным модулям с помощью дублированного двухточечного соединения, организуемого через объединительную панель.

Ассоциация *PCI Industrial Computer Manufacturing Group* утвердила спецификации *PICMG* 2.17 *CompactPCI StarFabric Specification*, описывающие порядок использования *StarFabric*. Переход к *StarFabric* от существующих открытых платформ, которые базируются на параллельной шинной архитектуре, достаточно прост. Новая технология обладает полной обратной совместимостью со спецификациями *PCI, H*.110 и *Utopia*. Архитектура *StarFabric* поддерживает двухточечные соединения с пропускной способностью 2,5 Гбит/с и позволяет использовать стандартные кабельные соединения и разъемы (в частности, проводка Категории 5 с разъёмами *RJ*-45).

В традиционном оборудовании *ATM* (например, концентратор доступа или граничный коммутатор) сетевые процессоры приходится включать в каждый модуль. Такая децентрализованная архитектура требует установки дорогостоящих линейных плат. С помощью *StarFabric* такие платы можно освободить от выполнения интеллектуальных функций, а, следовательно, снизить их цену. Данные, попадающие в систему через линейную плату, передаются с помощью коммутации через *StarFabric* для централизованной обработки.

Граничный коммутатор *ATM*, использующий традиционные архитектуры, для каждой линейной платы должен иметь сетевой процессор второго уровня и связанную с ним память. Такая нагрузка оказывается весьма обременительной в плане стоимости и энергопотребления. Если нужно добавить новую службу, а сетевой процессор второго уровня не обладает достаточной мощностью, все линейные платы придётся заменить.

В случае построения оконечного коммутатора *ATM* на базе технологии *StarFabric* линейные платы характеризуются более высокой плотностью размещения портов и не требуют установки локальных интеллектуальных компонентов. Платы сетевых процессоров (*Network Processing Unit, NPU*) представляют собой высокопроизводительные централизованные платы, которые подключаются к линейной плате через коммутирующую структуру. Трафик поступает в систему через линейные платы, проходит через *NPU* и выходит наружу через платы глобальной сети.

На современном рынке борьба за лидерство развернулась сразу между несколькими архитектурами межсоединений. *HyperTransport* и *RapidIO* — технологии организации высокоскоростного соединения между двумя микросхемами — непосредственно соперничают с *PCI-Express*. А поскольку архитектура *StarFabric* предназначена для организации соединения между платами и шасси, она не является прямым конкурентом данных технологий.

StarGen поддерживает единственную 4*X serial* магистраль, обеспечивающую поток данных до 5 Гбит/с. Средства управления трафиком присутствуют в полном объеме, так же как и “горячее” подключение. Однако упрощенный протокол приводит к большой из-

быточности данных и требует программной поддержки для реализации стека поддерживаемых протоколов. Дополнительно заявлена совместимость с *PCI*, что также требует дополнительного трафика для поддержания когерентности памяти всех вычислительных элементов. Такой подход не позволяет создавать архитектуры с максимально доступной производительности.

4.4.6. *InfiniBand*

InfiniBand является высокопроизводительной коммуникационной средой для создания мощных серверов и суперЭВМ. Высокая скорость, небольшие задержки, масштабируемость и высокая надёжность передач делают эту шину стандартом де факто для создания суперЭВМ. Спецификация *InfiniBand* определяет связь между процессорным узлом и высокопроизводительными узлами ввода/вывода, такими как устройства памяти. Стек протоколов *InfiniBand* сложнее *TCP/IP*. Несмотря на открытость спецификации, его реализация является коммерческим продуктом компании - разработчика. При этом приложения, использующие только часть этого стека, не могут считаться *InfiniBand* приложениями. Подобно *Fiber Channel, PCI Express, Serial ATA*, и другим современным стандартам, в *InfiniBand* используются двунаправленные последовательные линки точка-точка для соединения процессора с быстродействующими периферийными устройствами, такими как диски. В *InfiniBand* допускаются также операции мультикастинг. Базовая скорость стандарта — 2,5 Гбит/с в каждом направлении, применяются порты, состоящие из групп в 1*X*, 4*X* и 12*X* базовых двунаправленных линий. Существуют режимы Single Data Rate (SDR) - работа с базовой скоростью, *Double Data Rate* (*DDR*) - битовая скорость равна удвоенной базовой и *Quad Data Rate* (*QDR*) - соответственно, учетверенной. Соответственно в SDR скорость 2.5 Гбит/с в каждом направлении, *DDR* - 5 Гбит/с, *QDR* - 10 Гбит/с, *FDR* - 14.0625 Гбит/с и *EDR* - 25.78125 Гбит/с на линию.

Для *SDR, DDR* и *QDR* линков используется кодирование 8*b*/10*b* – каждые 10 бит информации содержат 8 бит данных. Таким образом, реальная скорость передачи данных - $^{4}/_{5}$ от указанных скоростей,

соответственно реальные скорости *SDR, DDR* и *QDR* - 2, 4 и 8 Гбит/с. Для *FDR* и *EDR* линков используется кодирование 64*b*/66b – каждые 66 бит передаваемой информации содержат 64 бита данных. Линки могут содержать 4 или 12 линий, называемых соответственно 4*X* или 12*X*. Скорость линка 12*X QDR* составляет 120 Гбит/с или 96 Гбит/с полезной информации. С 2009 г. большинство систем используют линки 4*X* со скоростью 10 Гбит/с (*SDR*), 20 Гбит/с (*DDR*) или 40 Гбит/с (*QDR*). Соединения с 12*X* линками - это обычно связи в кластерах и суперЭВМ для соединений коммутаторов между собой с целью создания многопортовых коммутаторов. В табл. 4.10 приведена максимальная скорость передачи для различных типов каналов.

Таблица 4.10. Теоретическая скорость передачи данных для различных конфигураций каналов передачи и памяти *InfiniBand*.

	SDR	*DDR*	*QDR*	*FDR*	*EDR*	*HDR*	*NDR*
1*X*	2 Гб/с	4 Гб/с	8 Гб/с	14 Гб/с	25 Гб/с		
4*X*	8 Гб/с	16 Гб/с	32 Гб/с	56 Гб/с	100 Гб/с		
12*X*	2 Гб/с	48Гб/с	96Гб/с	168Гб/с	300Гб/с	1500Гб/с	

Для *SDR* коммутатора задержка составляет 200 нс, *DDR* – 140 нс и *QDR* – 100 нс. Задержка на передачу от одного конечного устройства к другому составляет 1,07 мкс на протоколе *MPI Mellanox ConnectX QDR HCAs* до 1,29 мкс для *MPI Qlogic InfiniPath HCAs* и 2,6 мкс для *MPI Mellanox InfiniHost DDR III HCAs*. Пропускная способность для всех передаваемых данных и для полезных данных приведена в табл. 4.11.

Таблица 4.11. Пропускная способность интерфейса *InfiniBand*, полные/полезные данные.

	SDR	*DDR*	*QDR*
1*X*	2,5/2 Гбит/с	5/4 Гбит/с	10/8 Гбит/с
4*X*	10/8 Гбит/с	20/16 Гбит/с	40/32 Гбит/с
12*X*	3 /24 Гбит/с	60/48 Гбит/с	120/96 Гбит/с

В *InfiniBand* не стандартизируется программное обеспечение, в стандарте только перечисляются функции, которые должны выполняться. Стандарт де-факто разработан *OpenFabrics Alliance*, который используют большинство пользователей под операционными системами (ОС) *Linux, FreeBSD* и *Windows*.

Программное обеспечение *InfiniBand*, разработанное *OpenFabrics Alliance*, реализуется как "*OpenFabrics Enterprise Distribution (OFED)*", под лицензией двух ОС – *GPL2* или *BSD* лицензия для

Linux или *FreeBSD* и "*WinOF*" при выборе *BSD* лицензии для *Windows*. В большинстве систем все передачи начинаются или заканчиваются в канальном адаптере. Каждый процессор содержит ведущий канальный адаптер (*host channel adapter - HCA*), а каждое периферийное устройство содержит ведомый канальный адаптер (*target channel adapter TCA*).

В *InfiniBand* данные передаются пакетами объёмом до 4 Кбайт. Посылки могут быть:

- Прямым доступом в память, запись или чтение (*Remote Direct Memory Access - RDMA*). Группа протоколов удалённого прямого доступа к памяти, при котором передача данных из памяти одного компьютера в память другого компьютера происходит без участия операционной системы. При этом исключается участие процессора в обработке кода переноса и необходимость пересылки данных из памяти приложения в буферную область ОС, то есть данные пересылаются напрямую на соответствующий сетевой контроллер.
- Канальная посылка или приём.
- Транзакция.
- Мультикастинг (групповая посылка, то есть одновременная посылка нескольким устройствам).

4.4.7. *Myrinet*

Логический протокол *Myrinet* был развит в *Caltech Submicron Systems Architecture Project*. Продолжение работ осуществлялось в рамках *Caltech Project* и *USC Information Sciences Institute ATOMIC Project* по созданию экспериментальной высокоскоростной сети на базе компонент *Caltech*. Оба эти проекта финансировались *DARPA* Министерства обороны США. На настоящий момент среда Myrinet является открытым стандартом. Впервые эта коммуникационная технология была предложена в 1994 году, а на сегодняшний день имеет уже более 1000 инсталляций по всему миру. Однако в последние годы популярность среды резко падает.

Среда *Myrinet* стандартизует формат передачи данных на канальном уровне - *Data Link level*, на физическом уровне среда может быть

реализована в различных вариациях. Обмен данными в среде происходит посредством передачи пакетных сообщений между сетевыми адаптерами вычислительных модулей через систему коммутаторов crossbar. Среда *Myrinet* позволяет строить структуры произвольных топологий. В среде стандартизированы формат пакетов, способ адресации узлов, а также механизм управления потоком данных. Каналы в среде полнодуплексные, пропускная способность превышает 1 Гбит/с на канал в одном направлении. В стандарте предусматривается аппаратная реализация обработки стека протоколов сети, благодаря чему *Myrinet* имеет минимальную задержку доставки данных (латентность). Задержки на передачу коротких сообщений пользовательский процесс - пользовательский процесс могут достигать величин, меньших 5 мкс.

Основным производителем компонент *Myrinet* является американская компания *Myricom*. Основной функциональной единицей адаптеров *Myricom* является процессор *LANaiX*, кроме которого в состав адаптера также входит блок синхронной статической памяти, контроллер локальной шины вычислительного модуля и контроллер канала *Myrinet*. С шиной вычислительного модуля адаптер связан через специальный мост: шина *BM* (например, *PCI*- 2.2) - шина *EBUS* процессора *LANai*, этот мост выполнен в отдельной СБИС и находится на плате адаптера. Кроме устройств обеспечения интерфейса процессора *LANai* с системной шиной, в состав моста также входит *DMA*-контроллер для осуществления прямых пересылок данных между памятью адаптера и системной памятью вычислительного модуля.

Компании *CSPI* и *Myricom* встроили сеть *Myrinet* в среду *VME* и продемонстрировали совместно разработанный метод интеграции *Myrinet* в *VME*. Эта технология позволяет строить масштабируемые сети с пакетной передачей данных со скоростями обмена информацией порядка гигабайта в секунду между *VME* модулями, крейтами и системами с одной стороны и внешними рабочими станциями и персональными компьютерами с другой. В спецификации "*Myrinet on VME*" предлагается реализация Системой сети *Myrinet* (*Myrinet System Area Network - Myrinet SAN*) в *VME*-среде через передние панели и объединительную панель. Кроме того, в предлагаемом стандарте определяется канальный и физический уровни сети *Myrinet* (в

соответствии с эталонной моделью *ISO*). В спецификации канального уровня сети *Myrinet* определяются форматы пакетов данных, методы управления потоками информации и способы коммутации передаваемых пакетов. В спецификации физического уровня определяются электрические сигналы, физическая среда передачи данных и назначение выводов разъемов.

Основные приложения сети такие же, как и у *InfiniBand*, но стек протоколов этой сети проще. При этом присутствуют все необходимые механизмы управления трафиком.

4.4.8. *RapidIO*

Интерфейс *RapidIO* предложен компаниями *Mercury Computer Systems* и *Motorola* (ныне *Freescale*) как развитие шины, применявшейся в многопроцессорных системах цифровой обработки сигналов компании Mercury.

Стандарт *RapidIO* разрабатывался специально для удовлетворения важнейших требований приложений реального времени: обеспечения малых задержек, детерминизма, надежности и масштабируемости, снижения энергопотребления, размеров и веса, обусловленного требованиями встроенных систем. Однако высокие показатели системна его основе по скорости передачи и надёжности привели к тому, что *RapidIO* начинает использоваться и для построения серверов, прежде всего в США.

Стандарт развивается консорциумом *RapidIO Trade Association*, в который входят ведущие производители электроники. Целью является создание единой коммуникационной системы для встраиваемых систем, обрабатывающих большие потоки информации. Стандарт широко используется в военной и промышленной областях, применяется также и для создания высокопроизводительных вычислительных комплексов. В 2012 году по результатам конкурса существующих коммуникационных систем *NGSIS* группа выбрала стандарт *RapidIO* как стандарт для создания новых электронных космических систем и правительство США с этим согласилось. В настоящее время в группу *NGSIS* включены практически все ведущие компании космической отрасли: *BAE Systems, Boeing Space Systems, Cisco, Honeywell, Lockheed Martin, NASA/JPL, NASA/Goddard, Orbital*

Sciences, Sandia National Laboratories, Space Systems/Loral and the US Air Force Research Laboratory. Система *RapidIO*, представляет собой архитектуру межсоединений типа «точка-точка» и может быть построена при помощи совокупности переключающих элементов (коммутаторов). Пакет с данными пересылается от устройства к устройству через коммутаторы, которые в свою очередь могут однозначно интерпретировать эти пакеты и определить оконечную точку передачи данных. На рис. 4.21 показан пример построения такой системы.

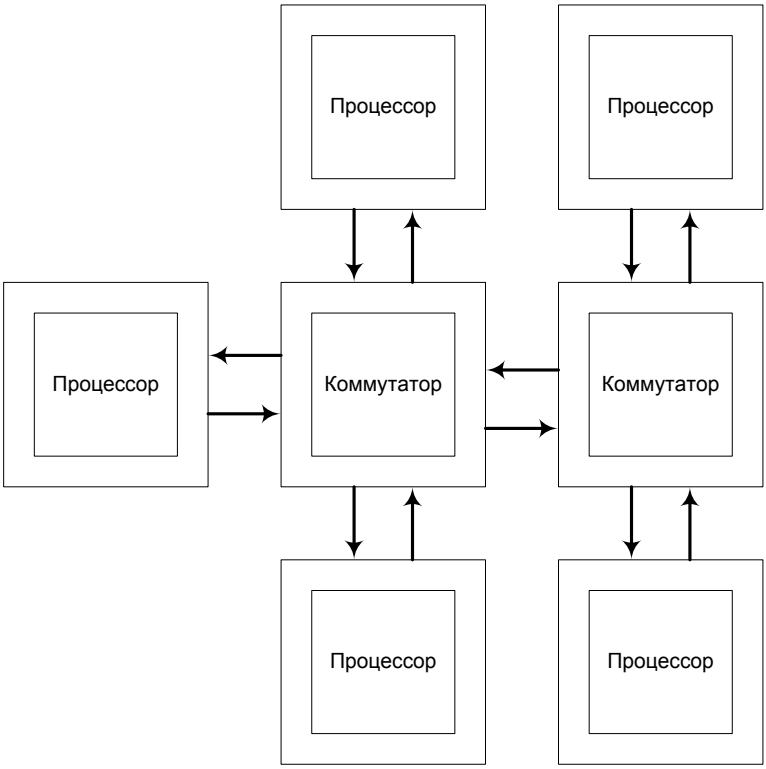


Рис. 4.21. Процессорная система на переключающих элементах

В стандарте *RapidIO* используется 3-уровневая иерархия: физический (соответствует физическому и канальному уровню 7-уровневой модели *OSI*), транспортный (соответствует сетевому уровню модели *OSI*) и логический (соответствует транспортному уровню модели *OSI*) уровни. Эта иерархия показана на рис. 4.22 [89].

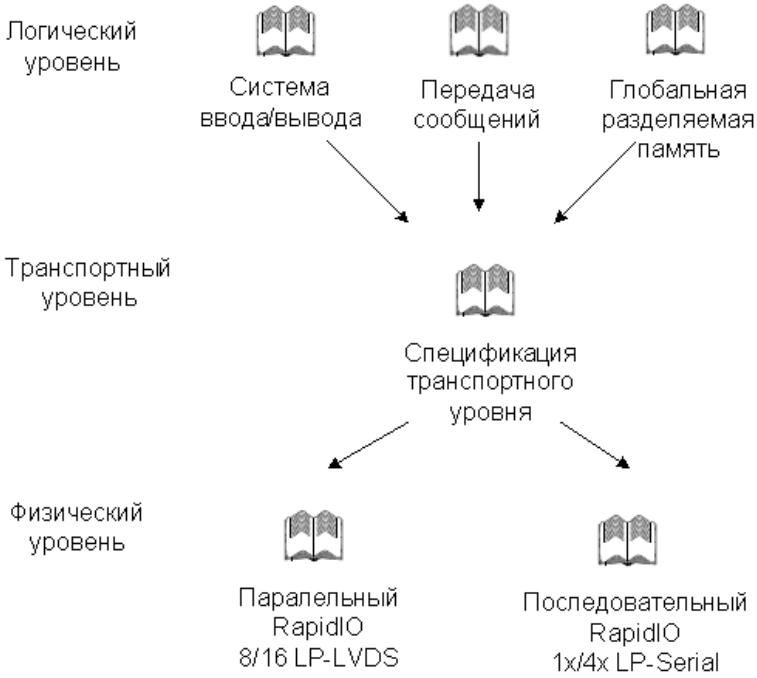


Рис. 4.22. Иерархическое построение стандарта *RapidIO*

Имеется две спецификации физического уровня:

• *LP-LVDS* — канал точка-точка представляет собой параллельный 8- или 16-разрядный дуплексный интерфейс, частота тактового сигнала — 250, 375, 500, 750 или 1000 МГц;

• *LP-Serial* — канал точка-точка представляет собой последовательный дуплексный интерфейс, состоящий из 1, 2, 4, 8 или 16 подканалов (англ. lane), скорость передачи битового потока: 1,25, 2,5, 3,125, 5 или 6,25 Гбод. В готовящемся стандарте RapidIO 3.0 скорость планируется поднять до 12,5 Гбод.

Параллельный интерфейс *RapidIO LP-LVDS* используется, прежде всего, для внутримодульного обмена и обмена по объединительной (кросс) плате. Интерфейс *LP-Serial* позволяет осуществлять внутримодульную, межмодульную и межмашинную передачу данных. Для *Parallel RapidIO* частота передачи изменяется от 250 МГц до 1 ГГц и данные захватываются по обоим фронтам тактового сигнала, соответственно полоса пропускания до 4 Гбайт/с для 8-разрядного интерфейса и до 8 Гбайт/с для 16-разрядного интерфейса.

Стандарт *RapidIO LP-LVDS* предполагает синхронную дифференциальную передачу данных в каждом направлении, при этом происходит выравнивание входного тактового сигнала и входных данных для надёжного распознавания входной информации, а также выравнивание пакетов и контрольных символов по 32 бита для правильной организации потока данных.

Непосредственно сама инициализация устройств осуществляется посредством обмена обучающими последовательностями (ОП). Это специальная битовая последовательность может быть легко распознана входной логикой устройства, и проигнорирована в случае, если устройства уже синхронизированы. ОП не может быть включена в состав пакета или использована для прерывания пакета. Все транзакции должны быть заблаговременно остановлены. ОП представляет собой 64 - разрядную последовательность для 8-разрядного порта. Формат ОП предусматривает передачу 32 единиц в каждом разряде, чередуемую с передачей 32 нулей в каждом разряде.

В процессе прохождения пакетом разных уровней иерархий в него добавляются необходимые данные и служебная информация. В конце пакет с включёнными в него контрольными символами разбивается на посылки по 8 бит и непосредственно передаётся по системе передачи данных.

Управление процессом передачи осуществляется сигналом *FRAME* (см. рис. 4.30). Сигнал управления передатчика *FRAME* служит для определения начала пакета или контрольного символа и представляет собой *NRZ* (*no-return-to-zero*) сигнал. Также *FRAME* предваряет первый байт пакета или первый байт контрольного символа и остаётся неизменным для других байтов пакета, включая *CRC*. Однако *FRAME* должен предварять также и пустые (*idle*) контрольные символы между пакетами или в составе пакетов. Сигнал

управления передатчика *FRAME,* также, как и все пакеты и контрольные символы, выровнен по 32 бита. Это значит, что максимальная частота переключения составляет каждые 4 байта.

Для *Serial RapidIO* скорость передачи может быть от 1,25 до 6,25 Гбод на линию (скорость передаваемых данных соответственно составляет от 1, 2, 2,5 и 5 Гбод из-за кодирования 8*b*/10*b*). По стандарту могут быть однопроводные (1*X*), четырёхпроводные (4*X*) и восьмипроводные (8*X*) каналы в каждую сторону. “Горячее” подключение поддерживается только стандартом *Serial RapidIO*, средства управления трафиком присутствуют в полном объеме. Этот стандарт активно поддерживается фирмой *Freescale Semiconductors*. Сейчас доступны микропроцессоры со встроенными каналами *RapidIO* от целого ряда компаний.

Логический уровень верхнего уровня иерархии определяет общие протоколы и формат пакетов. Транспортный уровень средней части иерархии определяет необходимую информацию по коммутированию для передающихся пакетов от одной конечной точки до другой. Физический уровень нижней части иерархии определяет детальную информацию по интерфейсу, такую как механизм передачи пакета, контроль передачи, электрические характеристики, управление контроля ошибок нижнего уровня.

Логический уровень не предполагает определенного транспортного и физического интерфейса. Процесс доставки пакетов определяется транспортным и физическим уровнями протокола *RapidIO*.

Пересылки производятся пакетами различного размера. Ведущий формирует пакет запроса (*Request Packet*) и передает его в коммутатор, который пересылается его ведомому. Ведомый выполняет необходимую операцию (например, чтение или запись памяти) и отправляет ведущему пакет ответа (*Response Packet*). Когда ответный пакет достигнет ведущего, операция считается завершенной. Не все операции требуют ответного пакета. Например, для увеличения скорости передачи операция записи может выполняться без подтверждающего пакета ведомого. Кроме пакетов по шине передаются управляющие контрольные символы (КС), которые используются для управления потоком, подтверждения приема пакетов, начальной инициализации и повторной синхронизации каналов.

Транспортный уровень определяет механизм передачи пакетов от источника к приемнику. Каждое адресуемое устройство в системе должно иметь как минимум один уникальный идентификационный номер - *ID*. Когда источник формирует пакет для передачи приемнику, *ID* приемника указывается в заголовке пакета. *ID* источника также помещается в заголовок для формирования ответного пакета. Когда приемник формирует ответный пакет, он меняет местами поля *ID* источника и *ID* приемника и отправляет ответный пакет.

Маршрутизация пакетов в коммутаторах производится на основе *ID* приемника. Каждый коммутатор вычислительной системы содержит таблицы коммутации, с помощью которых выбирается выходной порт, в который передаётся пакет с заданным *ID* приемника. Также протокол *RapidIO* поддерживает режим передачи пакетов от одного источника нескольким приемникам (multicasting).

На транспортном уровне *RapidIO* определены два типа систем – малые (до 256 устройств) и большие (до 65535 устройств). Использование в системе число *ID* до 256 позволяет упростить конструкцию коммутаторов.

Физический уровень определяет следующее:

- инициализацию смежных устройств в системе;
- механизм передачи пакетов и контрольных символов между двумя устройствами;
- обработку ошибок;
- управление потоком;
- характеристики сигналов.

Пакеты и символы контроля.

Функционирование *RapidIO* основывается на транзакциях (см. определение в 4.3.3.4) запроса и получения подтверждения. Пакеты являются элементами коммуникации между конечными точками. Ведущий (другие принятые термины инициатор и мастер - транскрипция английского слова) инициирует транзакцию запроса, который передаётся ведомому (другое название слэйв – транскрипция английского слова). Ведомый генерирует ответную транзакцию ведущему для завершения операции. Данные по транзакции включены

в передаваемые пакеты, включающие необходимые битовые поля для гарантии надёжности передачи ведомому. Каждый пакет имеет приоритет. Всего приоритетов – 4 (0,1,2,3). Приоритет 0 – самый низший, приоритет 3 – самый высокий. В зависимости от приоритета находится очередность доставки пакетов.

На рис. 4.23 приведён пример начала ведущим операции в системе - передача транзакции запроса. Пакет запроса передаётся коммуникационному устройству, обычно коммутатору, который отвечает контрольным символом. Далее пакет направляется ведомому через коммуникационное устройство. Это завершает фазу операции запроса. Пакет запроса ассоциируется с транзакцией возврата через коммутатор, использующий контрольный символ для подтверждения каждого отрезка передачи. В то время, когда ответный пакет достигает ведущего, и он подтверждён, операция считается завершённой.

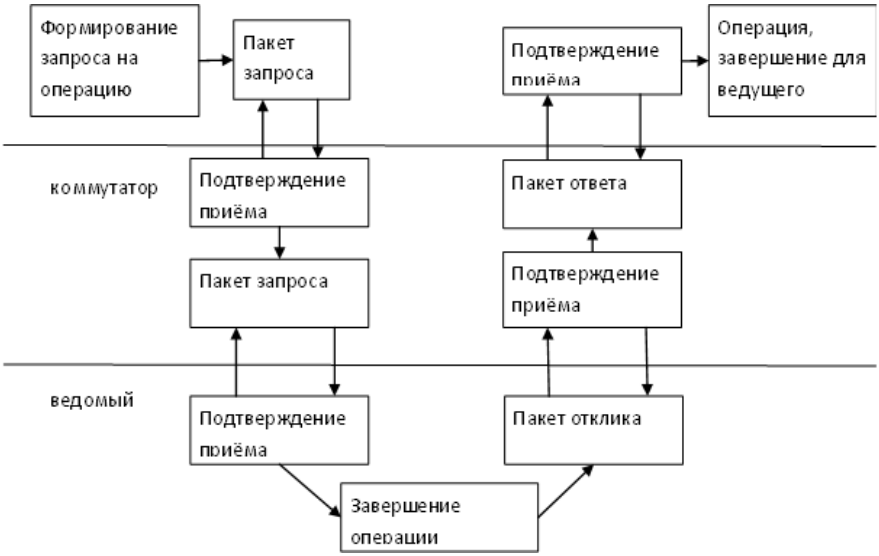


Рис. 4.23. Транзакции в сети *RapidIO*

Формат пакета.

Пакеты *RapidIO* состоят из полей 3-уровневой иерархической структуры. На рис. 4.21 показаны типичные пакеты запроса и ответа. Эти примеры касаются параллельного физического уровня. Пакеты, пересылаемые по последовательному каналу, слегка отличаются.

Пакет запроса начинается с поля физического уровня. Разряд «*S*» указывает, что это, пакет или контрольный символ. Единичное значение «*S*» бита указывает на контрольный символ, а нулевое – на пакет. Поле «*AckID*» служит для подтверждения приёма пакета. Поле «*PRIO*» служит для задания приоритета обработки пакета. Разряды *rsrv* зарезервированы для дальнейшего использования. В свою очередь разряды «*CRF*» могут быть использованы для расширения текущей системы приоритетов. Поля «*TT*», «Адрес приёмника» (*Destination ID*) и «Адрес источника» (*Source ID*) определяют тип механизма передачи пакета между коммутаторами: указывают тип используемого адресного механизма передачи, адрес устройства, куда пакет должен быть доставлен и где он порождён. «*Ftype*» и «Транзакция» (*Transaction*) указывают тип выполняемой передачи и рассмотрены в следующем подразделе. Разряды «Размер» (*size*) указывают размер передачи. Полезные данные в составе пакета *RapidIO* могут составлять от 1 до 256 байт. Поле «*TID* источника» указывает идентификационный номер (*ID*) передачи. Устройства *RapidIO* могут иметь до 256 отложенных передач между 2 конечными точками. Адрес в памяти приемника указан в поле "*Device Offset Address*". В поле "*Optional Data Payload*" содержаться передаваемые данные (при необходимости) Все пакеты завершаются 16-разрядным кодом *CRC* для повышения надёжности передач. Пакеты ответа подобны пакетам запроса. Поле «Статус» указывает, успешно ли завершена передача. Поле «*TID* приёмника» соответствует полю *ID* источнику пакета передачи.

Физический
Транспортный
Логический

RapidIO LP-LVDS. Пакеты запроса

1	3	1	1	2	2	2	4	8 или 16	8 или 16	4
Предыдущий пакет	*S*	*AckID*	*rsrv*	$\bar{S}$	*rsrv*	*Prio*	*TT*	*Ftype*	Адрес приёмника	Адрес источника	Транзакция

Size	*TID источника*	*Device Offset Address*	*Optional Data Payload*	*CRC*	Следующий пакет
4	8	32, 48, 64	От 8 до 256 байт	16	

RapidIO LP-LVDS. Пакеты ответа

1	3	1	1	2	2	2	4	8 или 16	8 или 16	4
Предыдущий пакет	*S*	*AckID*	*rsrv*	$\bar{S}$	*rsrv*	*Prio*	*TT*	*Ftype*	Адрес приёмника	Адрес источника	Транзакция

Size	*TID* приёмника	*Optional Data Payload*	*CRC*	Следующий пакет
4	8	От 8 до 256 байт	16	

RapidIO LP-Serial. Пакеты ответа

5	2	1	2	2	4	8 или 16	8 или 16	4
Предыдущий пакет	*AckID*	*rsrv*	*CRF*	*Prio*	*TT*	*Ftype*	Адрес приёмника	Адрес источника	Транзакция

Size	*TID* приёмника	*Optional Data Payload*	*CRC*	Следующий пакет
4	8	От 8 до 256 байт	16	

Рис. 4.24. Структура пакетов *RapidIO*.

Формат пакетов интерфейсов *RapidIO LP-LVDS* и *RapidIO LP-Serial* отличаются только полями заголовка (см. рис. 4.21). Принятый

пакет попадает во входной блок интерфейса *RapidIO*, где проверяется целостность пакета. Если пакет принят верно, он помещается в буфер, а передающему устройству отправляется контрольный символ подтверждения принятия пакета. В заголовке пакета имеется поле номера устройства назначения («*Destination ID*»), которое используется для определения порта коммутатора, куда будет передан пакет. Номер выходного порта определяется через таблицу маршрутизации порта, принявшего пакет. Далее, если выходной блок имеет возможность принять пакет (свободное место в буферной памяти), внутренний блок коммутации начинает передачу пакета из входного блока в выходной. Затем выходной блок передает пакет на выход по интерфейсу *RapidIO*. Внутренний блок коммутации не ждёт окончания приема пакета входным блоком, а начинает передачу в выходной блок, как только определится порт назначения. Также выходной блок начинает передачу на *RapidIO* при поступлении первого слова пакета. Если по окончании приема обнаружена ошибка *CRC*, передача пакета прерывается, пакет выбрасывается из буферов и, посредством контрольных символов, передатчик информируется о необходимости возобновить передачу этого же пакета.

Контрольные символы интерфейса *RapidIO* служат для синхронизации устройств, подтверждения приёма пакета, обеспечения управления обменом пакетами, используются для обнаружения и исправления ошибок. Контрольные символы стандарта *RapidIO LP-LVDS* выровнены по 32 разряда и представляют собой совокупность полей суммарной шириной 16 разрядов + те же инвертированные 16 разрядов. На рис. 4.25 представлен формат контрольных символов стандарта *RapidIO LP-LVDS*. Поле «*stype*» определяет тип контрольного символа. Поля «*ackID*», «*buf_status*» содержат в себе информацию, необходимую для организации управления потоком данных.

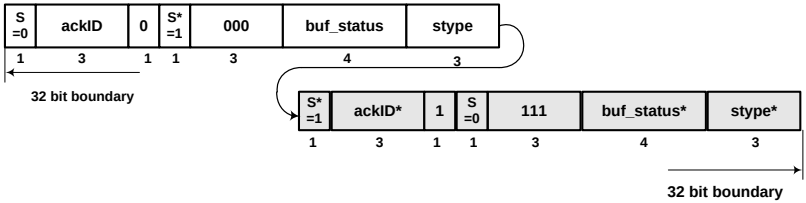


Рис. 4.25. Формат контрольных символов стандарта *RapidIO LP-LVDS*.

Контрольные символы стандарта *RapidIO LP-Serial* выровнены по 32 разряда и представляют собой специальный граничный символ «*Delimiter /SC/ or /PD/*» шириной 8 разрядов + совокупность полей суммарной шириной 24 разряда, включая поле «*CRC*» шириной 5 разрядов. На рис. 4.26 представлен формат контрольных символов стандарта *RapidIO LP-Serial*.

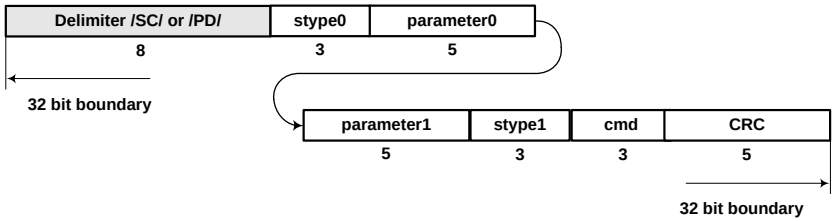


Рис. 4.26. Формат контрольных символов стандарта *RapidIO LP-Serial*.

Поля «s*type*0» и «*stype*1» определяют тип контрольных символов. Таким образом, в 32-разрядной границе, можно передать сразу два типа управляющих посылок. Поля «*parameter*0» и «*parameter*1» содержат в себе информацию, необходимую для организации управления потоком данных. Поле «cmd» дополнительно определяет тип контрольного символа «stype1». Поле «CRC» содержит в себе контрольную сумму 19-разрядной посылки, включающей последовательно поля «*stype*0», «*parameter*0», «*stype*1», «*parameter*1», «*cmd*». Стандарт *RapidIO* определяет 15 типов контрольных символов, которые разбиты на подгруппы. Каждая подгруппа может содержать

конечное число резервных позиций, которые могут быть использованы как определённые пользователем контрольные символы.

Ниже приведены основные типы и подгруппы контрольных символов.

Контрольные символы подтверждения:

- *packet-accepted* – свидетельствует о том, что противоположное устройство успешно приняло пакет;
- *packet-retry* – сообщает об ошибке буфера транспортного уровня противоположного устройства;
- *packet-not-accepted* – сообщает об общей ошибке при приёме пакета противоположным устройством.

Общие контрольные символы:

- *idle* – служат для управления потоком;
- *status* – служит для определения текущего состояния сети и исправления ошибок;
- *stomp* – прерывает заведомо ошибочный пакет;
- *start-of-packet* – обозначает начало пакета;
- *end-of-packet* – обозначает конец пакета;
- *restart-from-retry* – свидетельствует о возобновлении передачи пакетов из-за ошибки буфера;
- *throttle* – противоположное устройство требует прервать передачу пакетов;
- *multicast-event* – используются для передачи служебной информации.

Контрольные символы обслуживания сети:

- *link request/send-training* – противоположное устройство требует осуществить процесс синхронизации;
- *link request/input-status* – посылается в ответ на packet-not-accepted контрольный символ;
- *link request/reset* – противоположное устройство выполнило команду *Reset* (сброс);
- *link response* –служит для исправления ошибок, посылается в ответ на link request/input-status контрольный символ.

Контрольные символы *stomp, link request, restart-from-retry* могут быть использованы для прерывания передачи пакета. Прерванный

пакет не исследуется на предмет возможных ошибок и не сохраняется во входном буфере транспортного уровня. Остальные контрольные символы могут быть включены в состав пакета.

Для уменьшения величин задержек в системе контроль ошибок и механизм надёжной доставки пакета с использованием контрольных символов может быть отключён. В системах, где необходимы пересылки данных между памятями процессоров, задержки на передачу (задержки при прохождении пакета, например, через микросхему коммутатора) становятся определяющим фактором для увеличения производительности многопроцессорных систем. Для вычислительной среды с большой чувствительностью к задержкам интерфейс *RapidIO LP-LVDS* имеет преимущества перед *RapidIO LP-Serial* за счёт меньшей величины задержки, которые могут быть меньше 100 нс.

Форматы и типы транзакций.

Полная программная прозрачность связей требует большого разнообразия типов транзакций (операций связи, выполняемая посредством взаимодействия приёмной и передающей станций). Передачи *RapidIO* описываются двумя полям: формат пакета *Ftype* и *Transaction*. Для уменьшения накладных расходов дешифрирования передач, передачи сгруппированы по формату, показанному в таблице 4.12.

Таблица 4.12. Форматы комплекта передач *RapidIO*

Функции	Тип транзакции
Функции некогерентного ввода/вывода	*NREEAD* (чтение локальной памяти) *NWRITE, NWRITE_R, SWRITE* (запись локальной памяти) *ATOMIC* (чтение-модификация-запись локальной памяти)
Функции порта	*DOORBELL* (генерация прерываний) Message (запись в порт)
Функции системной поддержки	*MAINTENANCE* (чтение или запись регистров конфигурации, статуса и контроля)
Функции, определяемые пользователем	Открыты для применения специфичных транзакций

Функции	Тип транзакции
Функции когерентности кэш-памяти	*READ* (чтение глобально общей кэш-линии) *READ_TO_OWN*(запись глобально общей кэш-линии) *CASTOUT* (отказ от функции глобально общей кэш-линии) *IKILL* (недействительность кэш инструкций) *DKILL* (недействительность кэш данных) *FLUSH* (возврат глобально общей кэш линии в память) *IO_READ* (чтение некэшируемой копии глобально общей кэш линии)
Функции поддержки операционной системы	*TLBIE* (недействительность *TLB*) *TLBSYNC* (завершение недействительности *TLB*)

Операции, поддерживаемые протоколом *RapidIO* можно разделить на несколько групп:

Операции некогерентного ввода/вывода:

- *NREAD* - чтение памяти;
- *NWRITE, NWRITE_R, SWRITE* - запись в память;
- *ATOMIC* - чтение-модификация-запись;

Операции передачи сообщений:

- *MESSAGE* - передача сообщений;
- *DOORBELL* - выдача прерывания;

Системные функции:

- *MAINTENANCE* - чтение или запись конфигурационных, управляющих и статусных регистров;

Операции по поддержке когерентности памяти:

- *READ* - чтение разделяемой памяти;
- *READ_TO_OWN* - запись в разделяемую память;
- *CASTOUT, IKILL, DKILL, FLUSH, IO_READ* – обеспечение когерентности памяти.

На рис. 4.27 проиллюстрированы примеры вышеперечисленных транзакций.

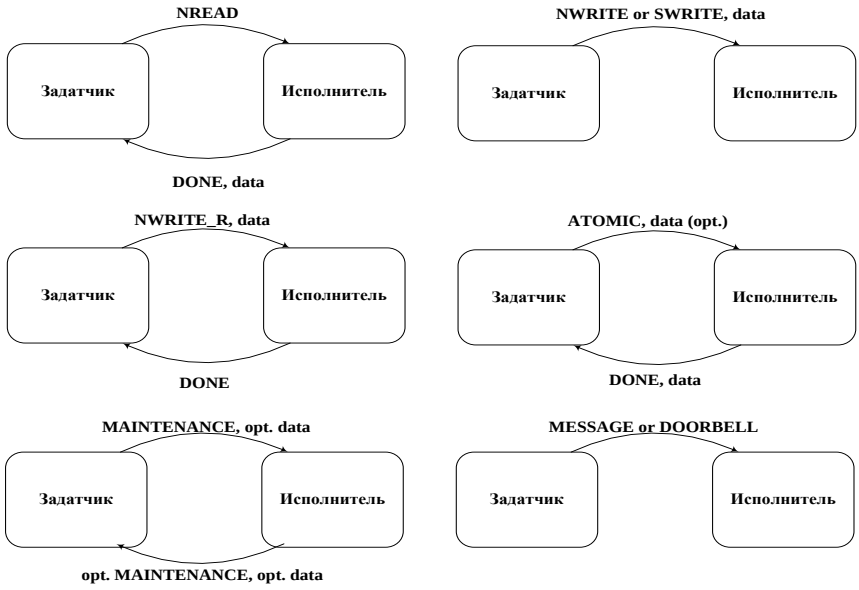


Рис. 4.27. Примеры транзакций протокола *RapidIO*.

Операция записи может передаваться 3 видами пакетов: *NWRITE, NWRITE_R, SWRITE*.

NWRITE – «отложенная» (*posted*) запись без подтверждения о получении. Быстродействие высокое, рекомендуется для использования в пространстве памяти, которая не отображается на устройства ввода-вывода (УВВ), и в пространстве «*prefetched*» памяти.

NWRITE_R – запись с подтверждением о получении. Обладает самым низким быстродействием, рекомендуется для использования в пространстве ввода-вывода, конфигурационных циклах и для доступа к устройствам ввода-вывода, отображаемых на пространство памяти.

SWRITE – «отложенная» (*posted*) запись без подтверждения о получении, выполняется только 8-ми байтными словами. Обладает самым высоким быстродействием, рекомендуется использовать в контроллерах ПДП (зарезервировано).

Запрос на операцию чтения передается пакетом *NREAD*. Ответ передается пакетами *DONE* либо *ERROR*. Эти два типа обобщаются стандартом как *RESPONSE* тип пакетов. Обладает низким быстродействием, задержка зависит от количества коммутаторов, через которые доставляется пакет. Увеличение быстродействия достигается предвыборкой в пространстве «*prefetched*» памяти.

В протоколе имеется возможность назначить приоритет для пакетов-запросов *NREAD, NWRITE, MESSAGE, DOORBELL*. Спецификация *RapidIO* требует, чтобы приоритет пакета ответа на данный пакет запроса был выше приоритета пакета запроса.

Передача сообщений.

Когда различные процессорные элементы в системе имеют доступ к одним данным, необходимо наличие протокола, обеспечивающего работоспособность системы в таком режиме функционирования. Во многих встроенных системах этот протокол поддерживается программными средствами. Если к памяти возможен доступ нескольких мастеров, должны быть использованы механизмы семафоров или захватов для того, чтобы гарантировать последовательный доступ различных устройств. В противном случае процессорные элементы должны иметь только собственную неразделяемую память. В системе, где ресурсы не разделяются, необходимы механизмы передачи данных из памяти одного процессорного элемента в память другого. Это может быть реализовано путём использования почтового ящика сообщений.

В системах передачи сообщения обычно используется 2 типа механизмов для передачи данных или команд от одного устройства к другому. Первых механизм — это прямой доступ к памяти (*DMA*), второй - сообщения. Основное отличие между этими моделями в том, что *DMA* передача управляется источником, в то время как сообщение приёмником. Это означает, что *DMA* источник не только требует доступа к приемнику, но также должен иметь доступ в адресное пространство приёмника. Источник сообщений требует доступ только к приёмнику и не требует доступа к памяти приёмника. В распределённых системах обычно имеются оба механизма.

Стандарт *RapidIO* имеет удобный механизм передачи сообщений, не зависящий от типа процессора и операционной системы (в системе одновременно могут находиться разные процессора, функционирующие с разными операционными системами). Протокол передачи сообщений описывает транзакции, которые разрешают механизмы почтового ящика или звонка. Почтовый ящик в *RapidIO* это порт, через который одно устройство может послать сообщение другому. Приёмное устройство определяет, что нужно делать с сообщением после его приёма. В *RapidIO* сообщения могут иметь размер от 0 до 4096 байт. Приёмное устройство может иметь от 1 до 4 очередей сообщений для записи сообщений.

Звонок *RapidIO* является транзакцией в порт, которая может использоваться для организации внутреннего прерывания или семафора. Сообщение звонок имеет 16-разрядное программно определяемое поле, которое может быть использовано для различных целей по организации передачи сообщений между 2 устройствами.

Операция Звонок, состоящая собственно из звонка и транзакции отклика, используется процессорным элементом для посылки короткого сообщения к другому процессорному элементу через коммуникационную сеть. Информационное поле транзакции Звонок используется для захвата информации по транзакции. Транзакция не содержит передаваемых данных.

Процессорный элемент, который принимает транзакции звонков, берёт пакеты и кладёт их в очередь транзакции звонков в процессорном элементе. Эта очередь может быть реализована в процессорном элементе или локальной памяти. Процессор считывает очередь для определения процессорного элемента, пославшего звонок, и считывает информационное поле для определения требуемых действий.

Указанные механизмы не описываются определённой программной моделью, пакеты имеют стандартный формат для межпроцессорного взаимодействия.

Глобально разделяемая память.

Один из протоколов, определённых в стандарте *RapidIO*, определяет глобально разделяемую память (*GSM – Globally Shared Memory*). Это означает, что память может физически находиться в

разных местах в системе и быть кэшируема для различных процессорных элементов. В этом случае говорят, что должна быть обеспечена когерентность кэш памяти с использованием механизмов *write through* (кэш с одновременной записью в основную память) или *write back* (кэш с записью модифицируемых при записи данных в основную память при запросе). То есть, если процессор обратился к какой-либо памяти, находящейся физически в другом месте, он всегда получит истинные данные, будет предусмотрен механизм записи изменённых данных из процессора во внешнюю память. Это несложно сделать, когда несколько процессоров работают с одной памятью, но сложно, когда в нескольких локальных памятях находятся копии одних данных и происходит их модификация в кэш памяти нескольких процессоров. Для поддержки когерентности в многопроцессорных системах обычно используются широковещательные передачи. Эти передачи видны всем процессорам, общая память которых кэшируются. Этот механизм иногда называют протоколом когерентности типа отслеживание запросов шины (*bus-based snooping*). Для этого метода каждый контроллер памяти ответственен за действия, где текущее значение каждого элемента данных доступно всем процессорным элементам системы. Обычно для этого создаётся центральный контроллер памяти, к которому все устройства имеют равный и унифицированный доступ. На рис. 4.28 показан типичный пример реализации такой системы с отслеживание запросов шины.

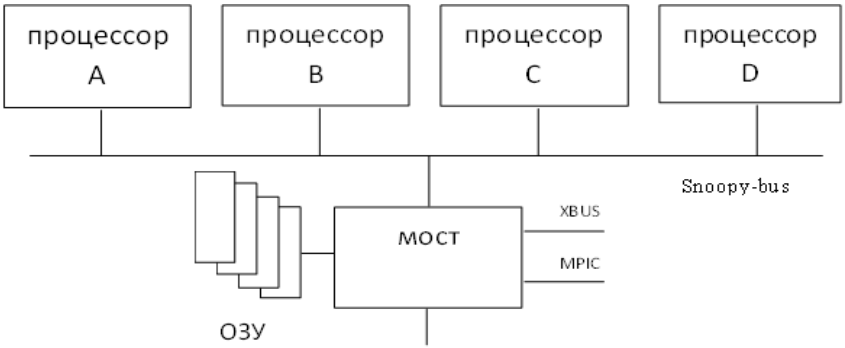


Рис. 4.28. Многопроцессорная система с отслеживанием запросов шины.

Для небольших систем (обычно до 4 микропроцессоров) механизм *snoopy bus* не масштабируется до верхнего уровня мультипроцессирования. В суперкомпьютерах и кластерах, имеющих распределённую память, должны использоваться различные механизмы когерентности памяти. Поскольку широковещательный протокол snoopy не эффективен в таких ЭВМ, так как имеется много узлов и требуются минимальные задержки на передачу, используются такие механизмы как каталоги памяти или таблица распределённых связей для сохранения пути обращения к последней копии данных. Эта схема часто называется, как когерентность памяти протокола неоднородного доступа к памяти (*cc-NUMA*). Типичная архитектура систем с распределённой памятью показана на рис. 4.29. В такой системе механизмы когерентности должны обеспечиваться между группами процессоров.

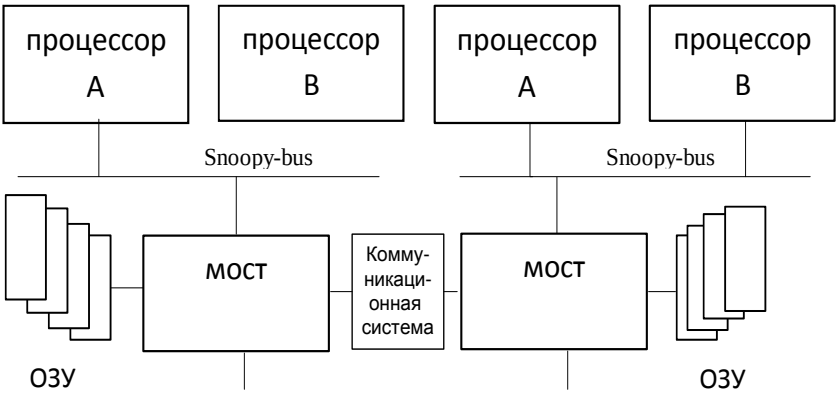


Рис. 4.29. Система с распределенной памятью.

Для систем *RapidIO* обычно используется относительно простая схема *bus-based snooping*. В этом случае каждый контроллер памяти отслеживает трассу с нахождением последней копии данных. Расширение *RapidIO GSM* обеспечивает набор поддержки контроля кэш памяти и операционной системы, такие как очистка блока памяти и синхронизация *TLB* (блок трансляции виртуальных адресов памяти в физические адреса) для построения больших многопроцессорных

систем с когерентной памятью. Для уменьшения затрат в системах с большим числом процессоров архитектура оптимизируется под небольшие кластеры с 16 процессорами, известные как когерентные домены.

В системах с распределённой памятью, если процессор хочет записать по какому-нибудь адресу памяти, все остальные процессора, имеющие доступ к этой ячейки должны сделать недействительным значение этой ячейки в своей кэш памяти. И только после этой операции инвалидации значения в кэш памяти может быть проведена операция записи.

Таким образом, спецификация *GSM* определяет следующие возможности:

- когерентность *cc-NUMA* систем обеспечивается для предоставления глобально разделяемой памяти через связи точка-точка в отличие от традиционной шинной архитектуры;
- размер поддерживаемых запросов памяти соответствует шагу, с которым поддерживается когерентность памяти (размеру кэш линии) или меньше;
- протокол *GSM* поддерживает контроль кэш памяти и другие операции, такие как очистка блоков памяти; эти функции обеспечиваются так, чтобы система могла работать под операционной системой и с приложениями.

Размер адресного пространства в системе может быть до 64 разрядов. Использование когерентности памяти снижает потребность в широковещательных передачах и повышает производительность системы.

Контроль передачи.

Контроль передачи является важной частью любой коммуникационной системы. Контроль передачи определяет правила и механизмы, используемые коммуникационной средой для определения, какие из возможных передач посланы в заданное время. Передача должна быть завершена без блокировок другими передачами. Соединения по шинному принципу используют алгоритмы арбитрации, позволяющие гарантировать первоочерёдность транзакции с

верхним приоритетом над транзакции с нижним приоритетом. В коммуникационной системе транзакции происходят из различных точек системы и нет необходимости в централизованном механизме арбитрации. Это определяет необходимость для иных методов управления транзакциями.

В *RapidIO* используется несколько механизмов, обеспечивающих передачу данных в системе без блокировок.

Контроль передачи на уровне линка.

Одна из задач разработчиков интерфейса была снижение накладных расходов и сложности стандарта, в особенности в области контроля передач. В *RapidIO* контроль передач определяется как часть физической спецификации. Это связано с тем, что транзакции во многом зависят от физического соединения и системного разбиения. Каждый пакет *RapidIO* несёт приоритет транзакции, ассоциируемый с транзакцией запроса. Определено 4 уровня транзакций запроса, что позволяет транзакции с верхним приоритетом передаваться раньше транзакций с меньшим приоритетом. Порядок транзакций управляется в потоке запросов, а не между потоками.

Потоки транзакций запросов делают возможным в системе структурировать потоки через механизм детерминирования поведения, то есть гарантии результатов, не зависящих от вероятностных факторов. Например, приложение может требовать, чтобы какие-то критичные по времени операции были выполнены с гарантированной задержкой, то есть не было бы превышение времени выполнения операций. Эти данные могут быть определены с высшим приоритетом запроса. Для управления критичными данными коммуникационная система может содержать структуру организацию очередей, позволяющую потоки с высшим приоритетом идти раньше потоков с низшим приоритетом. Очередь запросов может включать интервалы времени для гарантии того, что потоки от различных источников направленные к общему порту назначения, давали возможность передачи критичных данных. Это позволяет достичь изохронность, то есть равномерность передачи для выделенных классов данных.

На уровне линка в *RapidIO* описывается 3 типа механизмов контроля потоками: повтор (*retry*), дроссель (*throttle*) – регулятор, и разрешение (на передачу очередного пакета данных), (*credits*). Механизм повтора является простейшим механизмом и требуется не только для контроля потока, но и как компонент восстановления после аппаратной ошибки. Приемник, который не может принять пакет из-за отсутствия ресурсов или из-за того, что принимаемый пакет повреждён, может послать символ контроля «повтор». Источник в этом случае повторит передачу пакета.

Механизм дроссель позволяет использование символа контроля ожидания. Символы контроля ожидания могут быть установлены в пакеты в течение передачи пакета. Это позволяет устройству вставить состояния ожидания в промежутки пакетов. Принимающее устройство может также послать символ «дроссель» передающему устройству, указывая, чтобы он снизил посылку символов контроля ожидания.

Механизм разрешение полезен для устройств, которые имеют приёмный буфер (*FIFO*), особенно для коммутаторов. В этом случае определённые символы контроля содержат поле буфера статуса, которое предоставляет текущий статус приёмных буферов для каждого потока транзакций. Источник отправляет пакеты только тогда, когда он знает, что приёмник имеет достаточное буферное пространство.

Контроль непрерывных передач.

В дополнение к механизму контроля передачи на уровне линка в *RapidIO* определён механизм непрерывного контроля потоков. Механизмы уровня линка управляют потоками информации между соседними устройствами. В более сложных системах может возникать ситуация на какой-то период времени, когда комбинированные передачи от многих источников к одному или более адресу получателя могут быть причиной серьёзной деградации системы. Механизм непрерывного управляющего потока использует специальный блокирующий управляющий пакет, который может генерироваться коммутатором или конечной точкой. Блокирующий управляющий пакет посылается обратно через коммуникационную сеть к источникам

передач и закрывает выходной поток на период времени. Это имеет эффект снижения блокирующего потока в системе за счёт снижения потока в источнике.

Не смотря но то, что механизмы контроля на уровне линка , который специфицируется как часть физического уровня и обеспечивается контрольным символом, механизмы контроля непрерывных передач определяются как логический уровень транзакций и реализуется с пакетами. Существующая коммуникационная система посылает новые потоки управляющих пакетов, хотя они и не генерируют, и не отвечают на новые пакеты.

Физический уровень параллельного *RapidIO LP-LVDS*.

Логический пакет *RapidIO* определяется как последовательную цепочку бит независимую от физического уровня реализации. Это означает, что протокол *RapidIO* корректно функционирует одинаково с параллельным и последовательным интерфейсом. Физический уровень параллельного интерфейса *RapidIO* определён как 8- или 16-разрядный канал с низковольтовыми дифференциальными сигналами (8/16 *LP-LVDS*). Эта спецификация определяет использование 8 или 16-разрядную передачу данных в оба направления с одновременной передачей тактового сигнала и сигнала *FRAME* в оба направления с использованием стандарта *IEEE LVDS*. Так как все сигналы передаются по дифференциальным парам проводов, число сигнальных контактов увеличивается вдвое. Для 8-разрядного интерфейса требуется 40 контактов разъёма, для 16-разрядного – 72. Это число контактов относительно невелико по сравнению с числом контактов шинного интерфейса (порядка 200).

Интерфейс 8/16 *LP-LVDS* является синхронным интерфейсом. Это означает, что тактовый сигнал передаётся одновременно с данными, а также то, что не определён фазовый сдвиг между приёмными и передающими тактовыми сигналами и их частоты могут отличаться. Синхронизация по тактовому сигналу (клок) источника позволяет увеличить расстояние передачи и частоту передачи. Принимающая логика использует входящую клоковую пару сигнала для синхронизации приёма данных.

Сигнал *FRAME* используется для определения начала пакета или контрольного символа. Он формируется как *NRZ* (*no-return-to-zero*) сигнал, когда любое изменение его уровня трактуется как событие.

На рис. 4.30 представлен *RapidIO* 8 *LP-LVDS* механизм передачи данных между устройствами.

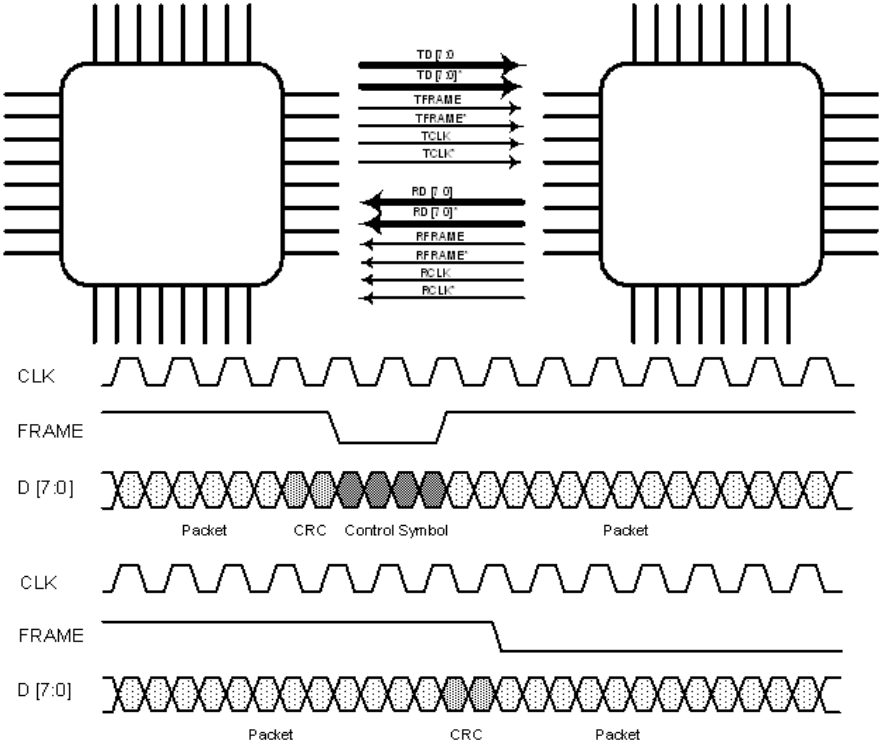


Рис. 4.30. *RapidIO* 8 *LP-LVDS* механизм передачи данных между устройствами.

Используемый *RapidIO* дифференциальный метод передачи *LP-LVDS* обладает существенно меньшей чувствительностью к общим помехам, чем простая однопроводная схема. Достоинством данного метода является то, что шумы, наводящиеся на двухпроводной линии, симметричны и не нарушают дифференциального сигнала, к

которому чувствителен приёмник. Также дифференциальный метод обладает меньшей чувствительностью к искажениям сигнала от внешних электромагнитных полей.

Метод *LVDS* использует токовый режим выхода передатчика и малые перепады дифференциального напряжения (до 350 мВ) на двух линиях печатной платы или сбалансированного кабеля, что обеспечивает малый уровень шума и снижение потребляемой мощности во всём диапазоне скоростей передачи.

Схема соединения *LVDS* передатчика с приёмником через линию с волновым сопротивлением 100 Ом представлена на рис 4.31. *LVDS* выход нагружен на дифференциальную пару линии передачи. Основной приёмник имеет высокий входной импеданс, поэтому основная часть выходного тока передатчика протекает через 100 Ом резистор терминатора линии, создавая на нём падение напряжения до 350 мВ, приложенное к входу приёмника.

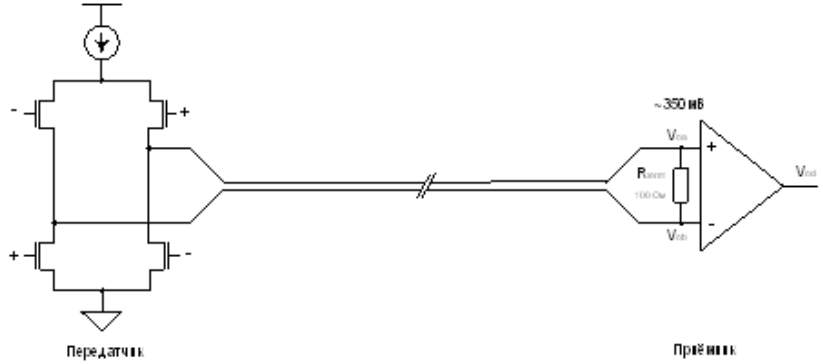


Рис. 4.31. Схема соединения *LVDS* передатчика с приёмником

При переключении выхода передатчика направление протекания тока через терминатор меняется на противоположное, обеспечивая достоверные логические состояния “0” или “1”.

Технология LVDS отражена в двух стандартах: *TIA/EIA* (*Telecommunications Industry Association/Electronic Industries Association*) - *ANSI/TIA/EIA*-644 (*LVDS*) и стандарт *IEEE* (*Institute for Electrical and Electronics Engineering*) - *IEEE* 1596.3.

RapidIO использует общий мультисистемный стандарт *ANSI/TIA/EIA*-644. Данный стандарт определяет выходные характеристики передатчиков и входные характеристики приёмников, т.е. он определяет только электрические характеристики. Он не ограничивает функциональные спецификации, протоколы, характеристики кабелей, соединений, т.е. он независим от конкретных применений.

ANSI/TIA/EIA требует поддержки другими стандартами, специфицирующими законченный интерфейс (кабели, соединители, протоколы и т.д.). Это позволяет успешно адаптировать данный стандарт для различных применений.

Стандарт *ANSI/TIA/EIA* рекомендует максимальную производительность в 655 *Mbps*, и оговаривает теоретический максимум в 1.923 *Gbps* ограниченный потерями в среде распространения. Это позволяет по стандарту специфицировать требуемую максимальную производительность, зависящую от качества сигнала, длины и типа среды распространения.

Стандарт также оговаривает минимальные требования к линии связи, электрические характеристики сигналов, безопасные условия работы приёмника в случаях отказов аппаратуры и другие конфигурационные ограничения, такие как одновременная работа множества приёмников. Основные электрические характеристики сигналов представлены в таблице 4.13.

Таблица 4.13. Основные электрические характеристики сигналов *LVDS*

Характеристика	Обозначение	Минимум	Максимум
Верхний выходной дифференциальный уровень	*Vohd*	247 мВ	454 мВ
Нижний выходной дифференциальный уровень	*Vold*	-454 мВ	-247 мВ
Изменение дифференциального напряжения	Δ *Vod*		50 мВ
Верхний выходной уровень в общем случае	*Vosh*	1,125 В	1,375 В
Нижний выходной уровень в общем случае	*Vosl*	1,125 В	1,375 В
Изменение напряжения в общем случае	Δ *Vos*		50 мВ
Резистор - терминатор	*Rterm*	90 Ом	220 Ом

На рис 4.32 и 4.33 проиллюстрированы соответствующие уровни сигналов.

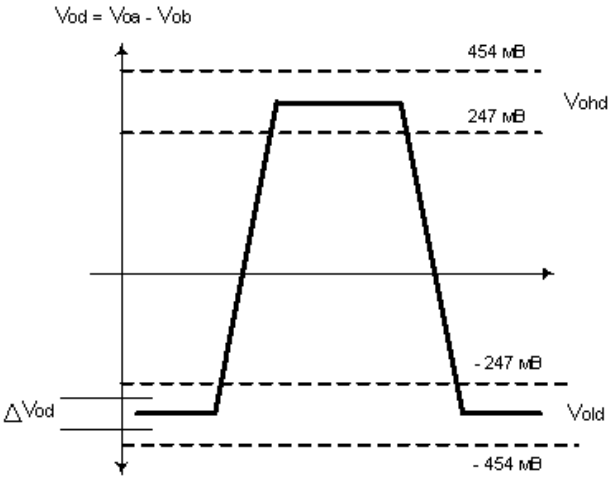


Рис. 4.32. Дифференциальные уровни сигналов *LVDS*.

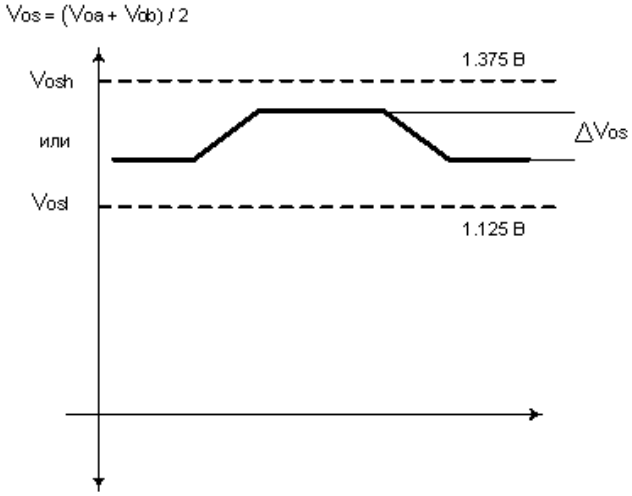


Рис. 4.33. Уровни сигналов *LVDS*.

Описание сигналов параллельного *RapidIO*.

В таблице 4.14 приведены сигналы параллельного интерфейса *RapidIO* 8 *LP-LVDS*

Таблица 4.14. Список сигналов интерфейса *RapidIO* 8 *LP-LVDS*

Название	**Тип**	**Описание**
TCLK0	Выход	Тактовый сигнал передатчика 8-разрядного порта (для старших разрядов, данных 16-разрядного порта). Соединяется с *RCLK*0 приемника
*TCLK0**	Выход	Дифференциальная пара *TCLK*0.
TD[7:0]	Выход	Передаваемые данные (разряды 7:0). Соединяется с *RD*[7:0] приемника.
TD[7:0]*	Выход	Дифференциальная пара *TD*[7:0].
TFRAME	Выход	Сигнал управления передатчика. Соединяется с *RFRAME* приемника.
*TFRAME**	Выход	Дифференциальная пара *TFRAME*
RCLK0	Вход	Тактовый сигнал приемника 8-разрядного порта (для старших разрядов, данных 16-разрядного порта). Соединяется с *TCLK*0 передатчика
*RCLK0**	Вход	Дифференциальная пара *RCLK*0.
RD[7:0]	Вход	Принимаемые данные (разряды 7:0). Соединяется с *TD*[7:0] передатчика.
RD[7:0]*	Вход	Дифференциальная пара *RD*[7:0].
RFRAME	Вход	Сигнал управления приемника. Соединяется с *TFRAME* приемника.
*RFRAME**	Вход	Дифференциальная пара *RFRAME*

На рис. 4.34 показана схема соединения 8-разрядных устройств.

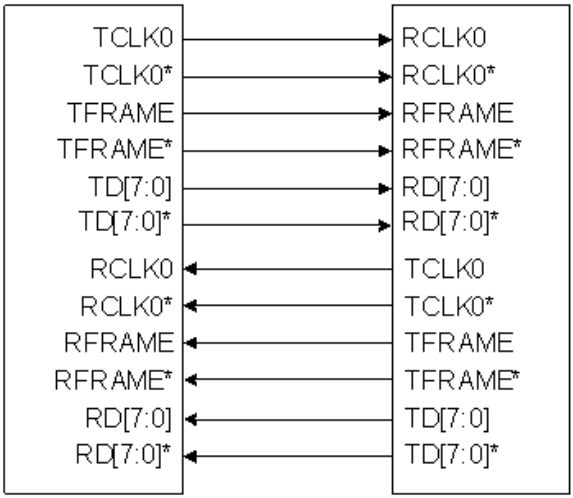


Рис. 4.34. Схема соединения 8-разрядных устройств

Для обеспечения распознавания пакетов и контрольных символов необходимо инициализировать входные порты устройств.

Для этой цели осуществляются два типа процедур инициализации 8/16 *LP-LVDS* входных портов:

- выравнивание входного тактового сигнала и входных данных, для надёжного распознавания входной информации;
- выравнивание пакетов и контрольных символов по 32 бита для правильной организации потока данных.

Непосредственно сама инициализация устройств осуществляется посредством обмена обучающими последовательностями (ОП). Это специальная битовая последовательность может быть легко распознана входной логикой устройства, и проигнорирована, в случае, если устройства уже синхронизированы.

Обучающая последовательность не может быть включена в состав пакета или использована для прерывания пакета. Все транзакции должны быть заблаговременно остановлены. ОП представляет собой 64 – разрядную последовательность для 8 – битового порта и 128 – разрядную последовательность для 16 – битового порта. Формат ОП предусматривает передачу 32 (64) единиц в каждом разряде, чередуемую с передачей 32 (64) нулей в каждом разряде. На рис. 4.35 представлен формат обучающей последовательности.

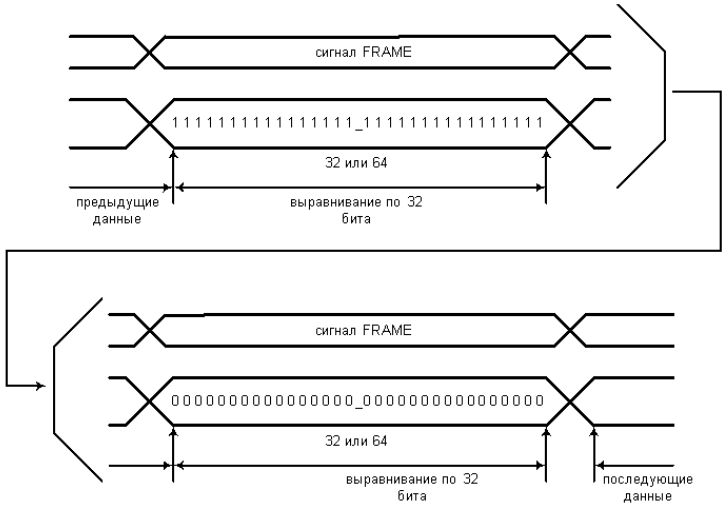


Рис. 4.35. Формат обучающей последовательности.

Обработка ошибок

Одной из основных характеристик интерфейса *RapidIO* является высокая надежность передачи. Ошибки должны детектироваться на каждом участке сети. Обнаружение ошибок производится следующими методами.

Для пакетов:

- весь пакет, за исключением заголовка, защищен контрольной суммой (32 – разрядная *CRC*), дополнительная контрольная сумма считается по первым 80-ти байтам пакета (для ускорения проверки длинных пакетов);
- проверка несоответствия поля «*AckID*» пакета (заложенная в протокол проверка для разрядов, не покрываемых *CRC*);
- контроль ошибок памяти (пакет слишком длинный или слишком короткий);
- таймеры на логическом уровне протокола (превышение времени ожидания ответного пакета).

Для контрольных символов:

• по каналу всегда передается либо пакет, либо управляющий символ (ошибка *S* бита);

• проверка несоответствия поля «*AckID*» контрольного символа;

• таймеры на физическом уровне протокола (превышение времени ожидания подтверждения приема пакета);

• первые 16 бит контрольного символа, инвертируются и включаются во вторые 16 бит контрольного символа.

Алгоритм CRC

С целью повышения надёжности при передаче данных необходимо определить их достоверность, то есть проверить, насколько принятая информация соответствует тому, что хотел передать удаленный узел. Для этого в сообщение закладывают некоторую избыточную информацию, число – являющееся функцией передаваемого сообщения. Эту служебную информацию приято называть контрольной суммой (или последовательностью контроля кадра – *Frame check Sequence, FSC*). Контрольная сумма вычисляется как функция от основной информации, причем необязательно только путем суммирования. Принимающая сторона повторно вычисляет контрольную сумму кадра по известному алгоритму и в случае её совпадения с контрольной суммой, вычисленной передающей стороной, делается вывод о том, что данные были переданы через сеть корректно. В стандарте *RapidIO* применяется метод контроля *CRC*16. Этот метод обладает большой вычислительной сложностью, но и вместе с тем большими диагностическими возможностями. В частности, он обнаруживает все одиночные ошибки, двойные ошибки и ошибки в нечетном числе бит.

Метод обладает также невысокой степенью избыточности. Например, для минимального по длине пакета *RapidIO* размером в байт контрольная информация длинной в 16 бит составляет всего 16 %.

Основная идея алгоритма *CRC* состоит в предоставлении большого двоичного числа, делении его на другое фиксированное двоичное число и использовании остатка этого деления в качестве контрольной суммы.

Особенности практической реализации алгоритма *CRC*

Из выше данного определения *CRC* следует, что алгоритм заключается в делении исходного сообщения на некоторый заданный полином, (в соответствии с правилами CRC арифметики) и остаток от этого деления и будет являться искомым значением.

В стандарте *RapidIO* предлагается следующий вид полинома:

$$G(x) = x^{16} + x^{12} + x^5 + 1$$

Алгоритм имеет следующие особенности:

- ширина *CRC* составляет 16 бит;
- входные данные выровнены по 32 разряда;
- начальное значение *CRC* регистра равно 0*xFFFF*;
- так как по стандарту первые 6 бит не покрываются *CRC*, то эти 6 бит данных маскируются нулями.

Понятно, что деление нельзя организовать обычными методами, поскольку длина входных данных не определенна заранее и может быть очень большой. Поэтому требуется алгоритм, который будет вычислять *CRC* небольшими порциями по мере поступления данных. На рис. 4.36 представлен способ реализации такого алгоритма.

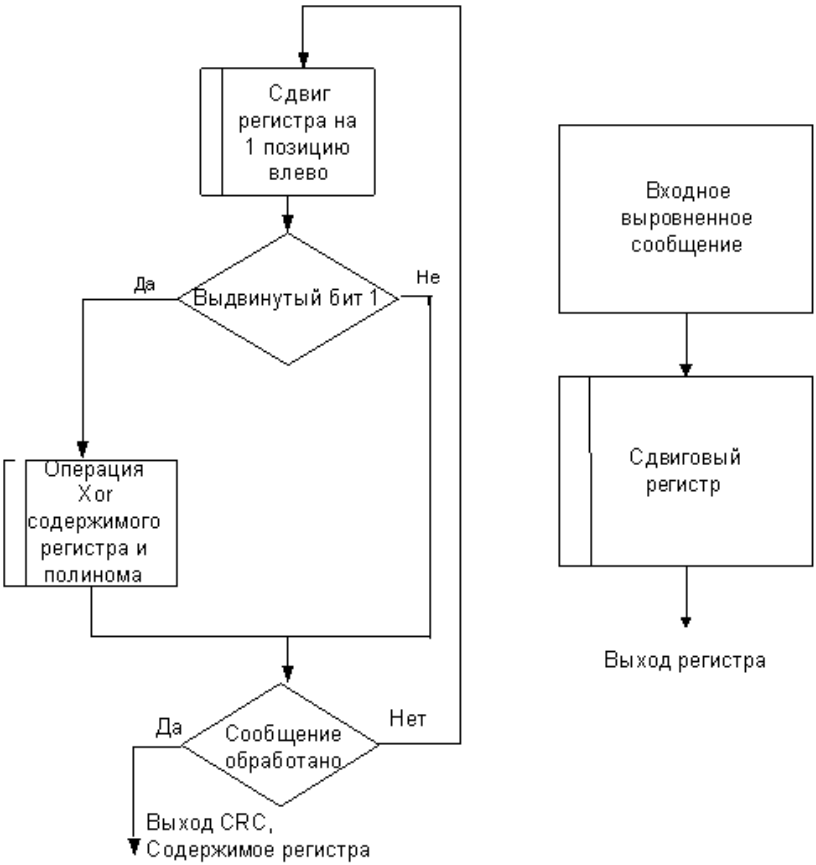


Рис. 4.36. Алгоритм подсчета *CRC*

Данный алгоритм очень прост и непосредственно вытекает из метода деления. Так как данные поступают порциями по 32 бита, то соответственно процедуру деления можно применить одновременно ко всем битам сообщения. Приведённые вычисления справедливы для каждого из 16 разрядов *CRC*.

Общие принципы обработки ошибок

При обнаружении приемником ошибки в пакете или его потере, передатчик автоматически (без участия программы) повторяет посылку ошибочного пакета или контрольного символа. Данная процедура определена стандартом. Таким образом, первичное обнаружение и исправление ошибок в интерфейсе *RapidIO* реализовано полностью аппаратно.

Физический уровень последовательного *RapidIO LP-Serial*.

Так как *Serial RapidIO* определяет только физический уровень, основная часть логики контроллера совпадает с логикой контроллера параллельного *RapidIO*, что существенно упрощает коммутацию между этими интерфейсами и позволяет использовать одни и те же знания.

Спецификация *Serial RapidIO* определяет физическое кодирование пакетов в поток последовательных битов на передающей стороне и распаковку в байты и слова на принимающей стороне. Интерфейс предполагает синхронную дифференциальную передачу данных в каждом направлении. При этом используется кодировка 8*b*/10*b* для стандартов *RapidIO* 1.0 и 2.0. Это означает, что 8 бит данных преобразуются в 10 бит закодированных данных. Код данных включает собственно данные и дополнительную информацию по встраиванию тактового сигнала и повышению надёжности передач (выявление одиночных и некоторых множественных ошибок). В этом коде 8 бит разбиваются на 2 группы: 5 LSB (*abcde*) и 3 MCB (*fgh*). Каждая из групп кодируется по таблице кодом, сбалансированным по «0» и «1», в результате получаются другие данные из 5 и 3 бит, к которым добавляется по одному вычисленному биту (*i* и *j*) текущего дескриптора (RD). В результате получается 10-разрядный кодовый символ (*abcdei* + *fghj*). При декодировании из 1024 возможных блоков выделяются 256 8-разрядных блоков данных и 12 специальных управляющих символов. Код позволяет выделять сигнал синхронизации на приёме и одиночные и некоторые множественные ошибки.

Электрическая спецификация *Serial RapidIO* определяется стандартом 802.3 *XAUI* драйверов с дифференциальными токовыми выходами, позволяющие передавать сигналы на большие расстояния (связь модулей по объединительной плате и между отдельными ЭВМ). Бывают 2 типа передатчиков: на короткие и большие расстояния. Короткие расстояния – это связи микросхем между собой внутри платы или с микросхемой на основной плате и мезонине. При этом снижается потребляемая мощность за счёт небольшого размаха сигналов. На больших расстояниях (передача между модулями в крейте) используется больший размах сигналов, что соответственно приводит к заметному повышению потребляемой мощности, особенно для высоких частот.

Формат пакетов (см. рис. 4.37) при последовательной передаче состоит из 10-разрядного заголовка физического уровня в начале пакета (*ackID* – 5 разрядов, *rsrv* – 2 бита 00, *CRF* – 1 разряд, *prio* – 2 разряда), транспортного заголовка (tt – 2 разряда), логического заголовка (*ftype* – 4 разряда). В конце пакета находится поле *CRC*-16.

	5	2	1	2	2	4	$n \times 16$	16
Предыдущий пакет	*AckID*	*rsrv*	*CRF*	*Prio*	*TT*	*Ftype*	Остальные транспортные и логические поля	*CRC*

Рис. 4.37. Формат последовательного пакета

Как и пакеты при параллельной передаче, пакеты при последовательной передаче должны быть кратными 32 разрядам. Это позволяет существенно упростить логику портов, ширина которых также кратна 32 разрядам. Однако пакеты транспортного и логического уровней кратны 16 разрядам. Поэтому, если общая длина пакета состоит из нечётного числа 16-разрядных полей, в конце пакета помещается 16 нулей для того, чтобы сделать длину пакета кратной 32.

Поле CRC в конце пакета используется для обнаружения ошибок при передачи пакета, причём первые 6 разрядов пакета (*ackID* и *rsrv*) не защищены *CRC*, защита только протоколом связи.

Максимальная длина пакета при последовательной передаче составляет 276 байтов, полезные данные при этом составляют 256

байт. Поэтому, если длина пакета превышает 80 байт (без поля *CRC*), используется второе поле *CRC*-16 в конце пакета, дополняемое при необходимости пустым полем в 16 бит.

На рис. 4.38 показана схема соединения двух 1*X* устройств, работающих по стандарту RapidIO LP-Serial.

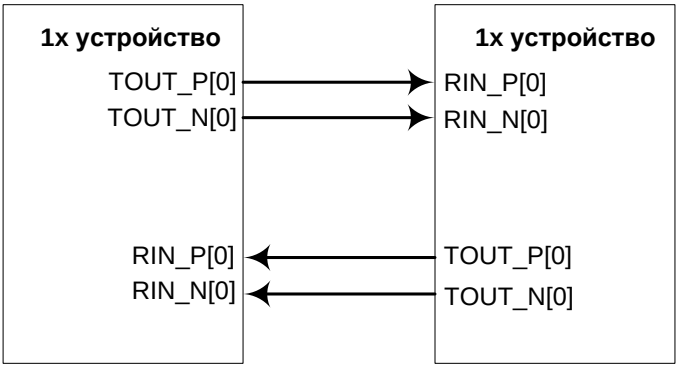


Рис. 4.38. Схема соединения двух 1*X* устройств, работающих по стандарту *RapidIO LP-Serial*.

На рис. 4.39 показана схема соединения двух 4*X* устройств, работающих по стандарту *RapidIO LP-Serial*.

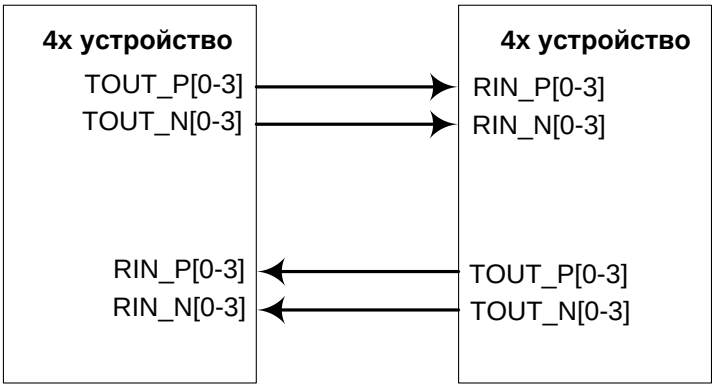


Рис. 4.39. Схема соединения двух 4*X* устройств, работающих по стандарту *RapidIO LP-Serial*.

На рис. 4.40 показана схема соединения 4*X* устройства и 1*X* устройства, работающих по стандарту *RapidIO LP-Serial*.

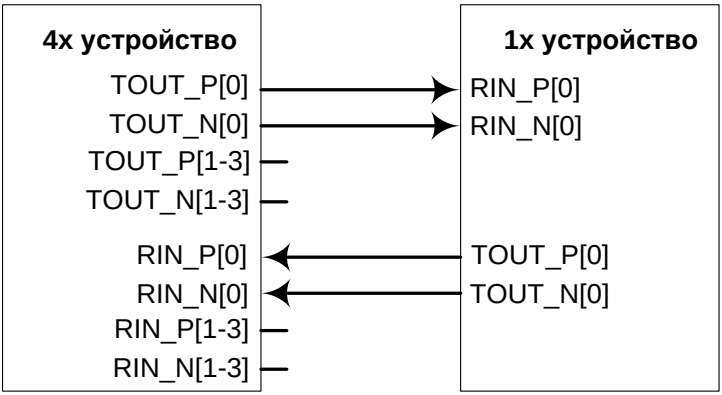


Рис. 4.40. Схема соединения 4*X* устройства и 1*X* устройства, работающих по стандарту *RapidIO LP-Serial*.

Для передачи данных применяется простейший двухуровневый код - *NRZ*. Механизм *RapidIO LP-Serial* передачи данных между устройствами показан на рис. 4.41. Нулевому значению соответствует нижний уровень сигнала, единице - верхний. Переходы электрического сигнала происходят на границе битов. Код *NRZ* не имеет синхронизации. Поэтому стандарт предусматривает использование специальных процедур помехоустойчивого 8*b*/10*b* кодирования, в котором используется десятибитовая основа для передачи восьмибитовых сигналов. Таким образом достигается большая плотность передачи, что позволяет приёмнику надёжно выполнить процедуру синхронизации и восстановления тактового сигнала. Кроме того, этот метод улучшает помехоустойчивость благодаря контролю принимаемых данных на десятибитном интервале. На рис. 4.42 показан процесс 8*b*/10*b* кодирования, передачи, приёма и последующего 10*b*/8*b* декодирования данных. Преобразованный десятибитовый сигнал имеет всего 256 основных + 12 специальных значений для передачи из 1024 возможных комбинаций. Оставшиеся 756 возможных комбинаций не имеют определённой расшифровки и представляют собой ошибочные комбинации. Наличие специальных

символов позволяет применять схему непрерывного обмена большими потоками данных между передатчиком и приёмником.

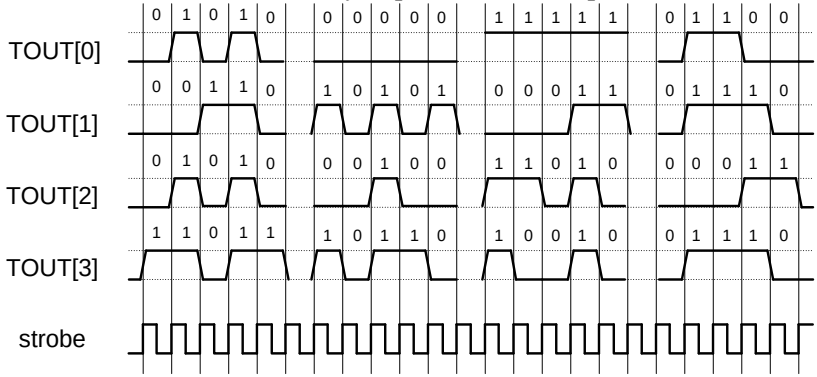


Рис. 4.41. *RapidIO LP-Serial* механизм передачи данных между устройствами.

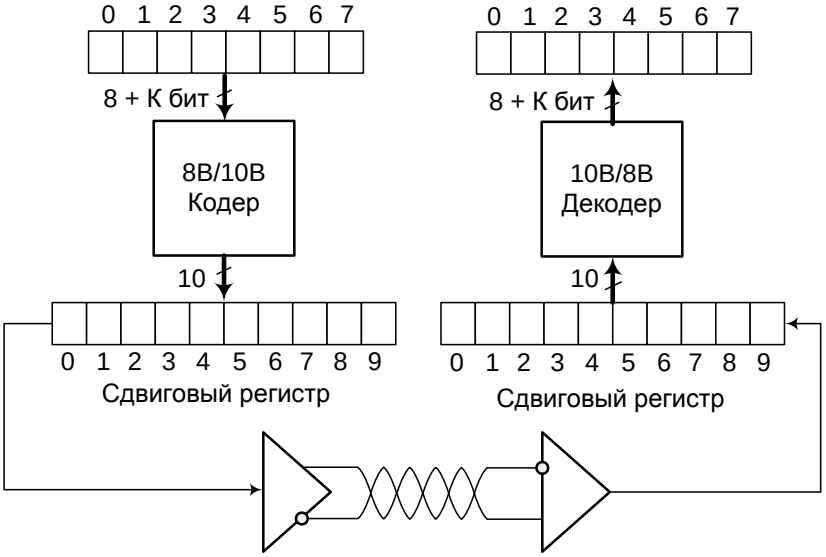


Рис. 4.42. Процесс 8*b*/10*b* кодирования, передачи, приёма и последующего 10*b*/8*b* декодирования данных.

Инициализация портов *RapidIO*, организация передачи пакетов.

После сброса устройству необходимо инициализировать входные *RapidIO LP-Serial* порты устройств с целью обеспечить правильное распознавание пакетов и контрольных символов.

При этом выполняются следующие процедуры:

- выделение тактовых сигналов приёмником из входного потока данных;
- устранение неравномерностей выделенных тактовых сигналов между собой и опорным тактовым сигналом;
- декодирование входных данных и кодирование выходных данных;
- синхронизация и выравнивание сигналов каждой из линий данных для надёжного распознавания входной информации;
- автоматическое определение текущего режима работы 1*X*/4*X* устройства без участия управляющего процессора;
- объединение пакетов и контрольных символов по 32 бита для правильной организации потока данных.

Непосредственно сама инициализация устройств осуществляется посредством обмена специальными служебными символами. Эта битовая последовательность может быть легко распознана входной логикой устройства, и проигнорирована в случае, если устройства уже синхронизированы.

Служебные символы не могут быть включены в состав пакета или использованы для прерывания пакета. Все транзакции должны быть заблаговременно остановлены. На рис. 4.43 показан пример передачи пакетов и контрольных символов по каналу 1*X*.

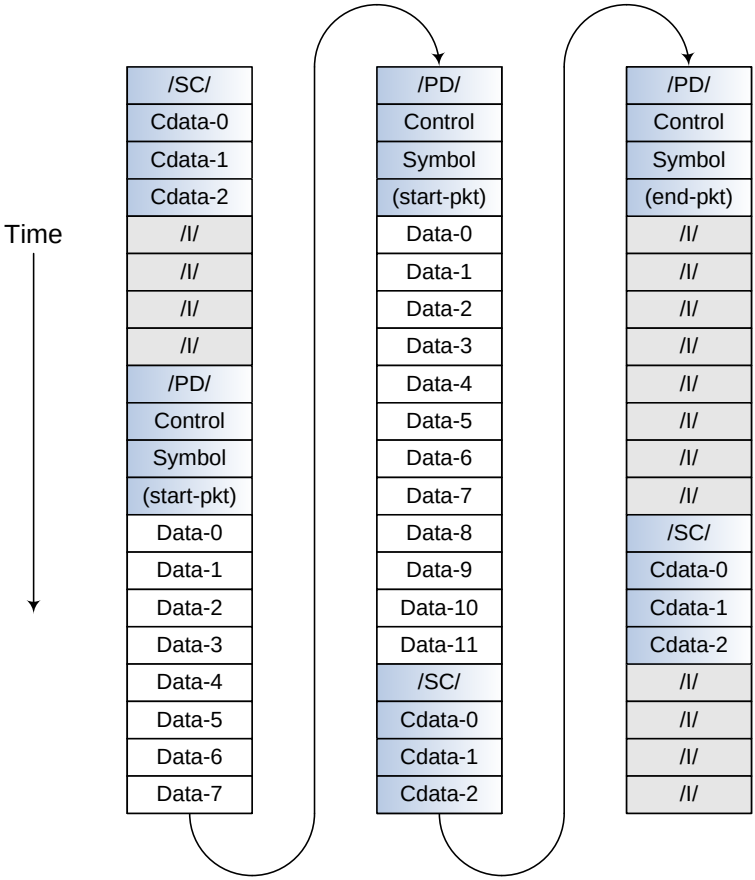
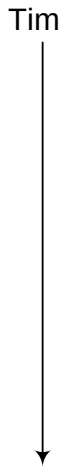


Рис. 4.43. Пример передачи пакетов и контрольных символов по каналу 1*X*.

На рис. 4.44 показан пример передачи пакетов и контрольных символов по каналу 4*X*.



Lane-0	Lane-1	Lane-2	Lane-3
/SC/	Cdata-0	Cdata-1	Cdata-2
/I/	/I/	/I/	/I/
/PD/	Control Symbol (Start-of-packet)		
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
/PD/	Control Symbol (Start-of-packet)		
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
Data-8	Data-9	Data-10	Data-11
/SC/	Cdata-0	Cdata-1	Cdata-2
/PD/	Control Symbol (End-of-packet)		
/I/	/I/	/I/	/I/
/I/	/I/	/I/	/I/
/SC/	Cdata-0	Cdata-1	Cdata-2
/I/	/I/	/I/	/I/

Рис. 4.44. Пример передачи пакетов и контрольных символов по каналу 4*X*.

Транспортный уровень интерфейса предназначен для передачи пакетов из входных блоков интерфейса *RapidIO* в выходные с учетом приоритетов и состояний очередей, при этом предусмотрены функции буферирования и коммутации, обеспечивающей полнодуплексный, неблокирующий режим работы. Каждый входной порт передает только один пакет в определенный момент времени и каждый выходной порт принимает также один пакет. Однако коммутатор способен одновременно передавать пакеты между различными парами передатчик-приемник. Так как несколько входных блоков могут передавать пакеты в один выходной блок, требуется буферизация пакетов во входных блоках. Специальные алгоритмы арбитрации используются для исключения блокировок в очередях. Блок коммутации должен работать в сквозном режиме передачи. Это

означает, что передача пакета из входного блока в выходной начинается до окончания приема пакета входным блоком.

Каждый выходной блок имеет алгоритм арбитрации, которая выбирает один из входных блоков для передачи пакета при нескольких одновременных запросах. Схема арбитра выходного блока показана на рис. 4.45.

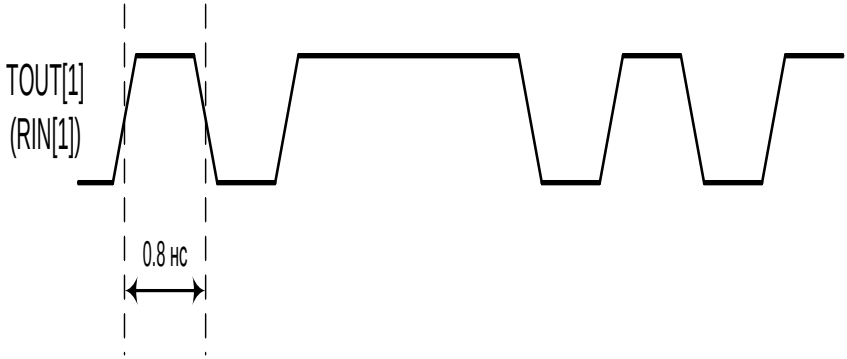


Рис. 4.45. Схема арбитра выходного блока.

Схема арбитрации выходного блока имеет два уровня – четыре арбитра с циклическим приоритетом (по одному на каждый приоритет) и арбитр с фиксированным приоритетом. В арбитрации участвуют только один пакет в выходном блоке, находящиеся в начале очереди. На каждый циклический арбитр приходит запрос от всех восьми портов на передачу пакета с определенным приоритетом. Результат от четырех арбитров поступает на выходной арбитр с фиксированным приоритетом. Стандарт *RapidIO* предусматривает, что пакеты с приоритетом *N* должны быть отправлены раньше пакетов с приоритетом *N*-1. Всегда, когда на арбитрацию с любого порта поступают пакеты с приоритетом 3, они передаются в выходной порт перед пакетами с приоритетом 2. Также пакеты с приоритетом 2 передаются перед пакетами приоритетом 1 и приоритет 1 перед приоритетом 0. Пакет с приоритетом 3 всегда будет отправлен в выходной порт, если есть свободные выходные буфера. Однако пакеты с меньшим приоритетом могут быть не переданы в выходной блок, если число свободных буферов равно или меньше определённых пороговых значений.

Схема арбитрации входного блока включает в себя очередь входных пакетов, которая формируется из входных данных. Пакет в начале очереди является кандидатом на передачу через внутреннюю схему коммутации в требуемый выходной блок. Если в данный момент пакет передать невозможно из-за заполнения выходного блока, производится перестановка и в начало очереди помещается другой пакет. Стандарт *RapidIO* предусматривает несколько режимов арбитрации. В первом режиме в случае перестановки выбирается пакет, ближайший к началу очереди и который может быть передан в требуемый выходной блок. Во втором режиме пакеты с более высоким приоритетом обрабатываются перед пакетами с меньшим приоритетом, если высокоприоритетный пакет может быть передан. В третьем режиме пакеты с более высоким приоритетом обрабатываются перед пакетами с меньшим приоритетом, даже если высокоприоритетный пакет не может быть передан в выходной блок в данный момент.

Схема коммутации транспортного уровня формирует очереди пакетов во входном и выходном блоках. Схема очередей показана на рис. 4.46.

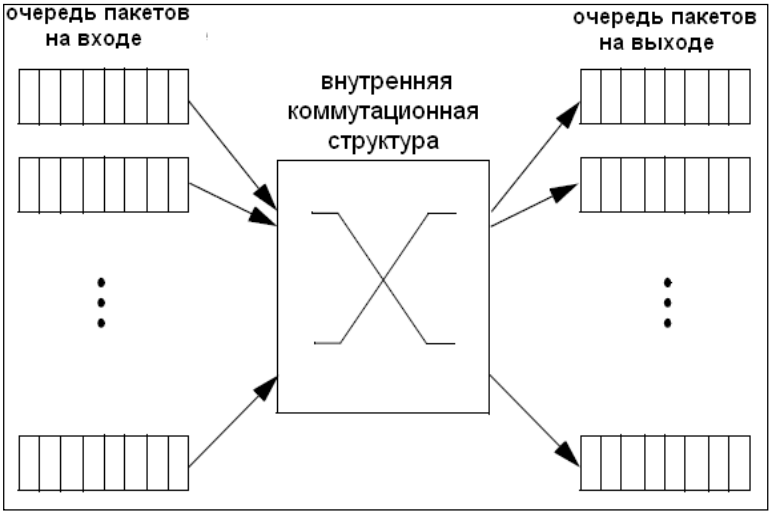


Рис. 4.46. Схема очередей пакетов.

Каждый выходной блок коммутатора может иметь буфер на 8, 16 или 32 пакета. Буферизация необходима для повторной посылки пакетов при ошибках передачи. Внутренняя схема коммутации определяет пакеты, которые будут переданы в выходной блок из одного из входных, на основании информации о заполнении буферов и пороговых значениях для каждого приоритета. Пороги заполнения для каждого выходного порта определены для приоритетов 0, 1 и 2. Пакеты с приоритетом 3 передаются всегда при наличии свободных буферов. Пакеты, пришедшие в выходной блок, передаются по интерфейсу *RapidIO* в соседнее устройство в порядке поступления за исключением случаев повторной передачи. При повторной передаче очередь выходного блока переупорядочивается. В этом случае для отправки выбирается пакет с самым высоким приоритетом, который дольше всех находится в очереди.

Каждый входной блок коммутатора может иметь буфер на 8, 16 или 32 пакета. Также как в выходном блоке, прием пакетов во входной блок ведется с учетом программируемых пороговых значений для каждого порта. Пороговые значения определены для приоритетов 0, 1 и 2. Пакеты с приоритетом 3 принимаются всегда при наличии свободных буферов. Принцип назначения пороговых значений такие же, как и для выходных блоков. Если пакет, пришедший по интерфейсу *RapidIO*, не может быть принят входным блоком, пакет отбрасывается и передающему устройству отправляется контрольный символ «*packet-retry*». Получив его, передатчик выбирает пакет с более высоким приоритетом (если такой имеется) и отправляет его.

Для определения порта, в который должен быть отправлен входящий пакет, используются таблицы маршрутизации. Входной блок использует поле «*DestinationID*» пакета как адрес для выбора одного из элементов в таблице маршрутизации, указывающего, в какой выходной порт необходимо передать пакет. Каждый из портов коммутатора имеет собственную таблицу маршрутизации, которая может быть запрограммирована независимо от остальных. Внутренний блок коммутации должен обеспечивать пересылку пакетов с 16-разрядными полями «*SourceID*» и «*DestinationID*» (64*K* устройств). Возможны два режима работы таблиц маршрутизации: стандартный

и иерархический. Стандартный режим поддерживает номера устройств от 0 до 511. На рис. 4.47 показан пример маршрутизации в системе, когда поле «*DestinationID*» составляет 8 разрядов (стандартный метод адресации).

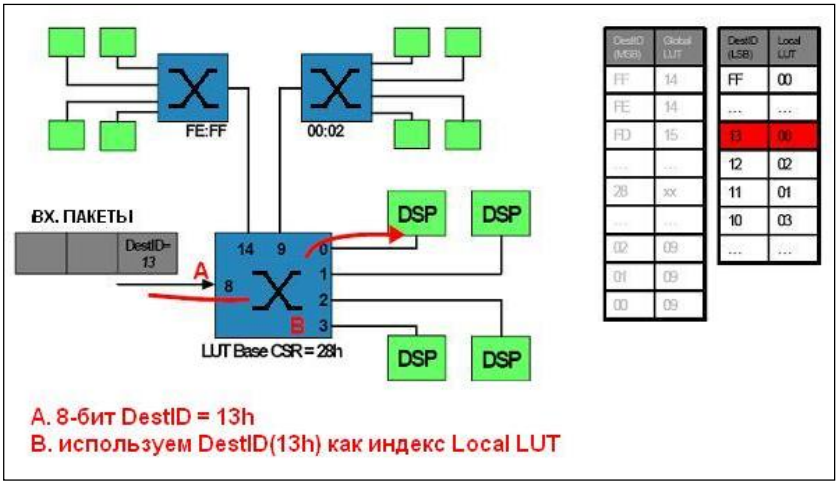


Рис. 4.47. Пример стандартной маршрутизации.

Иерархический режим обеспечивает маршрутизацию всего диапазона номеров от 0 до 65535 с некоторыми ограничениями. Режим работы выбирается для каждого порта индивидуально. Для повышения надежности, в таблицах маршрутизации используется проверка четности. Это исключает ошибочную передачу пакета при одноразрядных ошибках в таблицах. Для пакетов с 8-разрядным полем «*DestinationID*» входной порт использует *ID* как индекс в локальной таблице маршрутизации. Для пакетов с 16-разрядным полем «*DestinationID*» используется весь диапазон номеров от 0 до 65535. На рис. 4.48 представлен пример иерархической маршрутизации, когда старшие разряды поля «*DestinationID*» совпадают с полем *BASE* коммутатора.

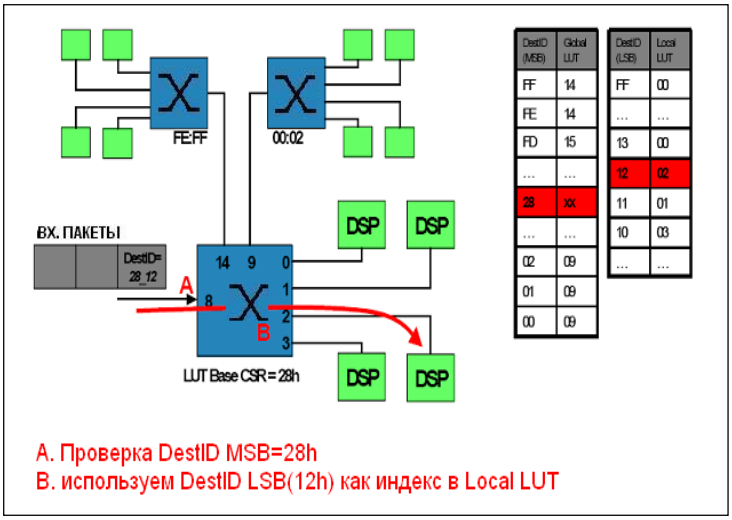


Рис. 4.48. Пример маршрутизации, когда старшие разряды поля "*DestID*" совпадают с полем *BASE* коммутатора.

На рис. 4.49 представлен пример маршрутизации, когда старшие разряды поля «*DestinationID*» не совпадают с полем *BASE*.

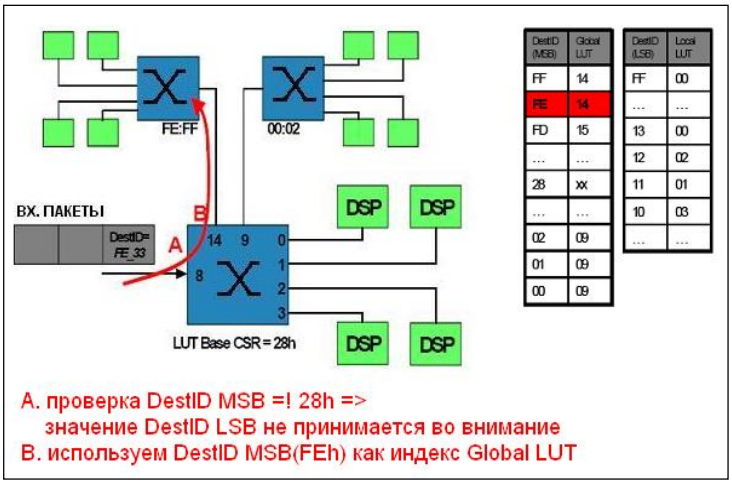


Рис. 4.49. Пример маршрутизации, когда старшие разряды поля «*DestinationID*» не совпадают с полем *BASE*.

Ядро коммутатора может поддерживать работу в смешанном режиме маршрутизации, когда часть портов работает в стандартном режиме, а часть в иерархическом. На рис. 4.50 показан алгоритм нахождения номер порта в зависимости от режима работы.

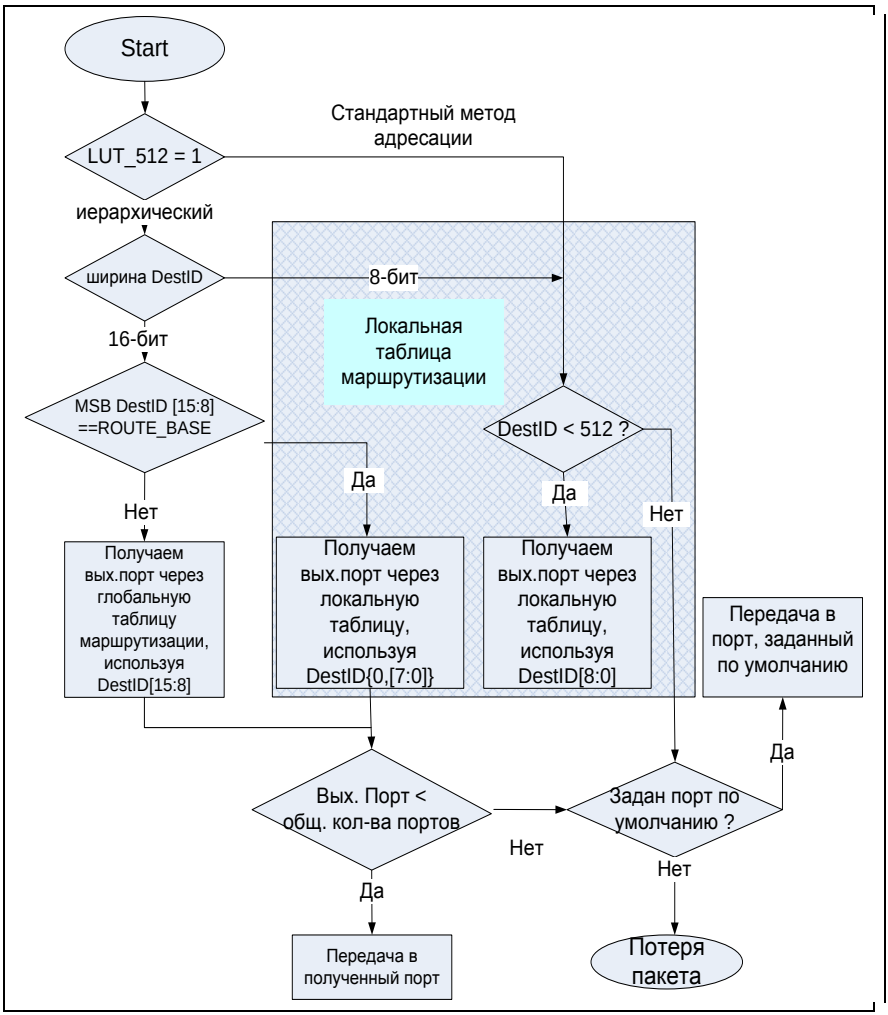


Рис. 4.50. Алгоритм нахождения номера порта в зависимости от режима работы.

По включению питания и после сброса все таблицы маршрутизации должны быть инициализированы, чтобы прописать соответствующие разряды четности. В неиспользуемые элементы таблиц должно быть записано значение 0xFF. Каждый порт блока коммутатора должен иметь свою таблицу маршрутизации. Отдельные таблицы могут быть запрограммированы разными значениями, что позволяет организовать различные пути прохождения пакетов в зависимости от входного порта. Программирование таблицы маршрутизации допускается в любое момент работы системы. Для программирования таблиц могут быть предусмотрены различные наборы регистров.

На логическом уровне протокол ввода/вывода интерфейса *RapidIO* использует операции типа запрос/ответ (*request/response operations*). Задатчик посылает запрос исполнителю, а он в свою очередь отсылает ответ обратно задатчику. Всего возможно два типа реакции исполнителя на запрос:

- *DONE*. Ведомое устройство сообщает об успешном выполнении действия и отсылает необходимые данные;
- *ERROR*. Ведомое устройство сообщает о возникновении ошибки при выполнении действия.

Задержки операций.

В системах, где необходимы пересылки данных между памятями процессоров, задержки на передачу (задержки при прохождении пакета, например, через микросхему коммутатора) становятся определяющим фактором для увеличения производительности многопроцессорных систем. Для вычислительной среды с большой чувствительностью к задержкам параллельный *RapidIO* имеет преимущества перед последовательным интерфейсом за счёт меньшей величины задержки. Не смотря на меньшую ширину данных параллельного *RapidIO* по сравнению с традиционными шинами, стандарт *RapidIO* оказывается существенно более производительным за счёт

отсутствия циклов арбитрации, возможности одновременной передачи по нескольким каналам и большей частоты. Задержки на передачу в *RapidIO* составляют меньше 100 нс.

Глава 5. Этапы разработки высокопроизводительных микросхем

На рынке существует целый ряд компаний, таких как *Synopsys, Mentor Graphics, Cadence*, предлагающих как отдельные САПР, так и целые платформы, покрывающие весь маршрут проектирования микросхем [90,91,92]. На рис. 5.1 представлен типичный маршрут проектирования, представляемый этими компаниями, разбитый на 2 части - *Back End* и *Front End*. Однако данные САПР являются универсальными, охватывают самые различные области применения и не учитывают особенностей проекта и некоторые особенности технологии производства микросхемы. Используя эти особенности можно получить существенно лучшие показатели проекта по всем параметрам: быстродействие, потребление питания, площадь кристалла, режимы эксплуатации. Однако отход от имеющихся маршрутов проектирования приводит к заметному увеличению стоимости проекта, времени проектирования, риска и может доходить до десятков раз при глубокой модернизации маршрутов.

Процесс разработки микросхем можно разбить на следующие основные этапы.

1. Формирование основных характеристик микросхемы (разработка технического задания - ТЗ).

2. Разработка архитектуры микросхемы, создание ее программной модели.

3. Разработка поведенческой модели.

4. Создание тестового программного обеспечения (ПО), включая системное программное обеспечение.

5. Разработка логической модели уровня *RTL*, верификация модели.

6. Уточнение технологии производства микросхем, выбор библиотек, *IP*-блоков (функционально законченных и полностью специфицированных блоков), оптимизация архитектуры микросхемы.

7. Создание принципиальной электрической схемы на уровне транзисторов (логический синтез, разработка заказных и аналоговых узлов).

8. Топологическое проектирование: размещение заказных блоков и библиотечных элементов, построение дерева синхронизаций, трассировка цепей, верификация проекта.

9. Тестирование изготовленных кристаллов и микросхем. Коррекция проекта при необходимости. Проведение испытаний микросхем, окончательное определение параметров микросхем.

Некоторые этапы проектирования из указанных выше могут отсутствовать. Например, если создается микросхема, функционально подобная известной, но с другими характеристиками, отсутствует этап 2, в ряде случаев не требуется этап 3. Оригинальность маршрута проектирования микросхем заключается, прежде всего, в специфике и порядке проектирования на каждом из выделенных этапов, цикличности повторения ряда процедур, включения некоторых процедур для достижения требуемых характеристик, выборе систем проектирования, включая разработку собственных.

Особенности проектирования высокопроизводительных микросхем заключаются в:

- необходимости создании сложных и оригинальных архитектур, требующих создание многоуровневой тестовой системы и системы верификации проекта,
- необходимости проектирования целого ряда функциональных узлов заказных образом (на уровне транзисторов),
- специальным деревом сигналов синхронизации (клоковым), позволяющим обеспечить минимальный временной разброс тактовых сигналов по всей площади кристалла,
- многоитерационными этапами синтеза и топологического проектирования,
- выборе специальных технологических процессов изготовления микросхем, зачастую ухудшающих такие параметры как потребление питания или стоимость микросхемы для достижения высоких частот функционирования.

5.1. Маршрут и методика проектирования микросхем

Рассмотрим поэтапно основные положения маршрута [93] проектирования высокопроизводительных микросхем.

1. Разработка технического задания включает формирование основных характеристик микросхемы в соответствии с потребностями рынка или требованиями заказчика, условий эксплуатации, оценка сложности микросхемы, топологических норм, требуемого тестового программного обеспечения, оценка времени проектирования и маршрута проектирования, необходимость покупки или разработки САПР, требуемые аппаратные, людские ресурсы. Успех создания микросхемы во многом зависит от продуманности маршрута проектирования и жесткому его следованию на протяжении всего проектирования.

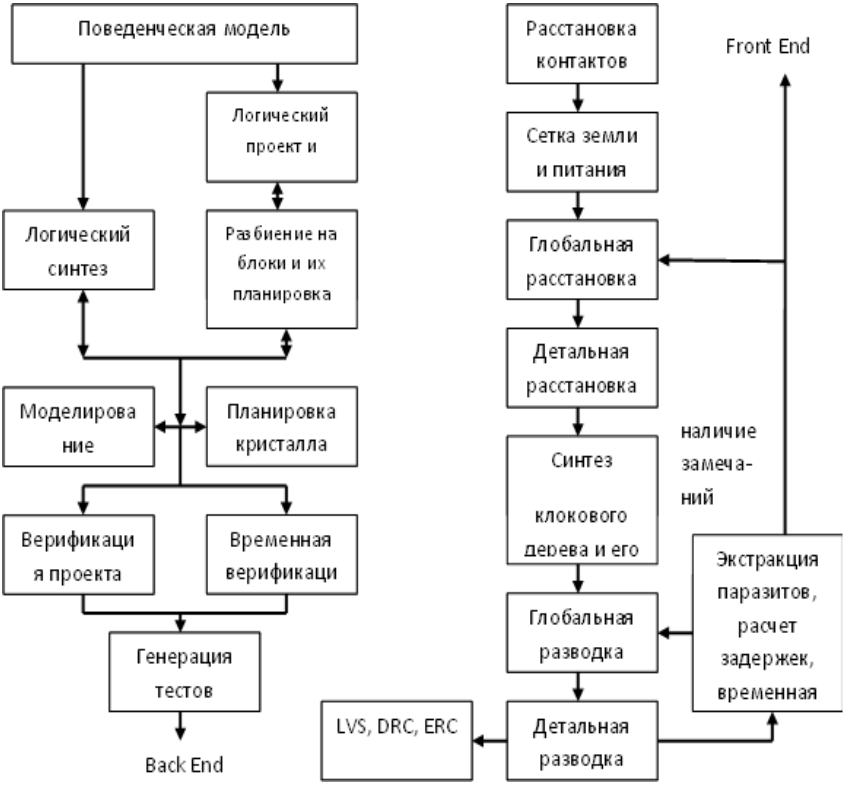


Рисунок 5.1 Стандартный маршрут проектирования микросхем.

В маршруте проектирования должны быть определены не только требуемые программные пакеты проектирования, но и их версии. Изменение маршрута проектирования должно проходить установленным порядком, а не по ходу проектирования микросхем. Как было сказано выше, эффективность маршрута проектирования может быть повышена его ориентацией под выделенный класс задач и используемые технологические нормы. Однако при охвате процесса разработки микросхем различной сложности, как правило, маршрут имеет несколько ветвей. Выбор же конкретной ветви проектирования происходит в процессе проектирования в зависимости от получаемых характеристик проекта. На этом же этапе необходимо определить по возможности ближайшие аналоги разрабатываемой микросхемы с целью использования существующего опыта работ, а также определить в существующих микросхемах наиболее удачные технические решения с целью их дальнейшего развития.

2. Разработка архитектуры микросхемы является сложным итерационным процессом. Для примера, разработка 64-разрядного суперскалярного микропроцессора с известной архитектурой на Западе оценивается суммой порядка 100 млн. долларов, разработка модернизированной архитектуры – сотни миллионов долларов (разработка микропроцессора *Cell* составила 400 млн. долларов [94]), разработка современного высокопроизводительного микропроцессора с новой архитектурой – сотни миллионов - единицы миллиардов долларов (разработка микропроцессора *Itanium* составила несколько миллиардов долларов).

На данном этапе проектирования происходит разработка базовой архитектуры, которая на последующих этапах может корректироваться и оптимизироваться. Оптимизации архитектуры выполняется на нескольких уровнях: оптимизация на поведенческой модели, оптимизация на логической модели, оптимизация на системных и пользовательских программах, оптимизация на этапах создания принципиальной электрической схемы и топологического проектирования.

Поведенческая модель позволяет оптимизировать базовые архитектурные решения. Логическая модель (*RTL*-модель) определяет основные характеристики микросхемы и тем самым позволяет определить критические узлы и несбалансированность архитектурных

решений. Выявление недостатков архитектуры при компьютерном моделировании требуются многие часы функционирования высокопроизводительных компьютеров. Скорость выполнения инструкций на *RTL*-модели до 1000 раз ниже скорости выполнения на поведенческой модели (*C*-модель, покомандная). Такая скорость не позволяет в разумное время проектирования проверить различные варианты архитектурных решений на логической модели. Ускорение оптимизации архитектуры на уровне логической модели можно получить, используя тестовые платы, функционально подобным платам с разрабатываемой микросхемой, где вместо разрабатываемой микросхемы устанавливаются микросхемы *FPGA* (ПЛИС). Такой подход позволяет оперативно не только отлаживать логическую модель, но и проверить и оптимизировать различные архитектурные решения. При этом частота функционирования микросхем *FPGA* ненамного меньше частоты функционирования разрабатываемой микросхемы, а в ряде случаев эти частоты совпадают. Так микросхема контроллера *Ethernet* 10/100 Мб/с, реализованная на *FPGA*, функционировала на той же частоте, что и разрабатываемая микросхема 1890ВГ3Т [10], модель микросхемы 250 МГц микропроцессора 1890ВМ5Ф, реализованная в *FPGA*, функционировала на частоте 24 МГц. Использование программируемых пользователями микросхем в настоящее время является общепризнанным подходом для создания и отладки новых электронных изделий [10].

Эффективная разработка микросхем с новой архитектурой невозможна без моделирования с соответствующим системным и прикладным программным обеспечением. Например, для разработки микросхем графического контроллера 1890ВГ10Т и 1890ВГ14Т [97,98] был создан специализированный стенд, позволяющий выполнять реальные пользовательские программы на текущей версии модели микросхемы, что позволило дополнительно оптимизировать архитектуру микросхем и поднять производительность. За счет оптимизации архитектуры на этом этапе производительность микросхемы 1890ВГ10Т была поднята почти в 2 раза, а микросхемы 1890ВГ14Т в 3 раза. Производительность микросхемы 1890ВГ14Т, изготовленной с технологическими нормами 0,35 мкм соответствует производительности аналогичных микросхем, изготовленных с технологическими нормами 0,25 мкм. На этапе разработки архитектуры

также определяются функциональные узлы, которые должны быть разработаны заказным путем. Перечень заказных узлов может быть уточнен при создании логической модели микросхемы.

На этапах создания принципиальной электрической схемы и топологического проектирования может оказаться, что некоторые цепи имеют слишком большую нагрузку или некоторые операции требуют слишком большое число логических элементов. Оптимизация микроархитектуры на этих этапах позволяет поднять быстродействие до десятков процентов, потребление питания и площадь кристалла могут быть снижены на единицы процентов. Для микропроцессора 1890ВМ5Ф на данном этапе удалось поднять быстродействие почти в полтора раза за счет оптимизации логической модели, позволившей уменьшить число логических элементов на критических функциях и уменьшить длину трасс на кристалле на критических путях. Было проведено несколько десятков таких итерационных процессов для данной микросхемы.

3. Разработка поведенческой модели требуется для решения несколько задач. На первом этапе разработки микросхемы поведенческая модель необходима для оптимизации архитектуры микросхемы. Особую значимость модель имеет для разработки программного обеспечения, которое может вестись параллельной с разработкой микросхемы.

Поведенческую модель можно также использовать для тестирования логической модели микросхемы [95]. Для этого на всем пути проектирования микросхемы происходит постоянное сравнение функционирования логической и поведенческой моделей. Модели проектируются независимыми коллективами разработчиков и, следовательно, вероятность одинаковых ошибок мала. Средствами проектирования [95] возможно на каждом такте процесса моделирования сравнивать состояния всех регистров поведенческой и логической моделей. При наличии различий состояний регистров делается соответствующее оповещение для поиска ошибки. Особую эффективность такая методика показала при создании сложнофункциональных микросхем. При создании суперскалярного 64-разрядного микропроцессора 1890ВМ5Ф такой подход позволил выявить свыше 30 % ошибок, сделанных в логической модели.

4. С ростом объема микросхемы трудоемкость разработки тестового ПО растет существенно быстрее трудоемкости разработки аппаратной части, где зависимость близка к линейной при хорошо структурированном проекте, и особенно высока для сложнофункциональных микросхем. Трудоемкость создания тестового ПО для таких микросхем может превышать трудоемкость всей аппаратной разработки. При создании 64-разрядного суперскалярного микропроцессора 1890ВМ5Ф трудоемкость создания тестового ПО превысила в 1,5 раза трудоемкость разработки аппаратной части, несмотря на то, что с аппаратной точки зрения проект микропроцессора достаточно сложный. В данном микропроцессоре целый ряд узлов создавался по методике заказного проектирования (суммарный объем заказных узлов составил около 300 тысяч транзисторов), использовалась динамическая логика, разрабатывались специальные библиотечные элементы, а общий объем микропроцессора составил 27 млн. транзисторов. Для повышения эффективности тестовое ПО должно разрабатываться независимыми коллективами разработчиков [99]. Опыт разработки показал, что менее квалифицированный разработчик, пишущий независимые тесты, может найти больше ошибок, чем более квалифицированный разработчик, разработавший соответствующий аппаратный узел. Современные цифровые микросхемы имеют сотни и тысячи регистров, число состояний и условий функционирования таково, что полное тестирование требует годы работы высокопроизводительной аппаратуры. Именно поэтому в настоящее время избежать ошибок очень сложно. Выделение области применения позволяет сократить число возможных состояний микросхем и максимально полно их протестировать. Другой путь к снижению ошибок - создание минимально возможной функциональности аппаратуры и операционной системы, достаточной для реализации и функционирования сервисов безопасности, позволяющей наиболее полно провести всестороннее тестирование и обеспечить возможность неформального доказательства корректности функционирования.

Тесты, необходимые для разработки микросхемы можно разбить на 4 группы: тесты функционального контроля, сертификационные тесты, тесты для отбраковки кристаллов и микросхем и тесты для

испытаний и периодического контроля. Наиболее сложными являются тесты функционального контроля. Во всех тестах функционального контроля должны быть предусмотрены автоматический контроль выполнения тестов с записью полного состояния микросхемы в случае сбоя, возможность пошагового выполнения тестов, выдача (сохранение на диске) состояний микросхемы в запрашиваемые моменты времени. Можно выделить следующие тесты функционального контроля.

1) Архитектурные тесты, то есть тесты, проверяющие соответствие модели разрабатываемой архитектуре.

2) Тесты всех отдельных функциональных узлов, позволяющие хотя бы один раз изменить состояние каждого разряда регистра или ячейки памяти с единицы на ноль и наоборот и выполнить соответствующую функцию. Например, для памяти должны быть тесты «бегущая» единица, когда во всю память записываются нули, а затем побитно записывается и считывается единица, бегущий ноль (обратный тест), шахматный код (чередуются единицы и нули) и пр. Эти тесты достаточно хорошо проверяют функционирование узла, но они не проверяют взаимодействие узлов между собой. Для эффективного выполнения этой группы тестов необходимо иметь достаточно мелкое разбиение проекта на функциональные узлы. В случае если число состояний требует слишком длительное тестирование, узел должен быть разбит на более мелкие части. Например, если тестирование АЛУ (арифметико-логического устройства) требует времени больше отведенного, должны тестироваться отдельно сумматор, умножитель и т.д.

3) Тесты взаимодействия узлов. Должны быть протестированы соседние функциональные узлы (обратный процесс в рассмотренном выше примере разбиения АЛУ). Если устройство сложное, должно быть создано несколько иерархий взаимодействия узлов. Число иерархий и разбиение на узлы должно определяться отведенным временем тестирования.

4) Тестирование всего устройство в целом. Должны быть созданы тесты, проверяющие взаимодействие всех узлов модели и функционирование микросхемы в целом.

5) Тесты взаимодействия модели микросхемы с внешними устройствами. Должна быть протестирована работа микросхемы с

микросхемами окружения и внешними устройствами. Для возможности данного тестирования должны быть соответствующие модели окружения, часть поведенческих моделей представляют компании, производящие микросхемы или внешние устройства. Например, практически все компании, производящие микросхемы памяти представляют модели на своих сайтах в Интернете. На рисунке 5.2 для примера показано разбиение тестирования микросхемы графического контроллера 1890ВГ10Т. Для наглядности на рисунке не приведено разбиение нижнего уровня модели 1890ВГ10Т.

6) Соответствие логической модели поведенческой. Результаты (состояния регистров) тестирования поведенческой модели и модели уровня RTL должны совпасть.

7) Случайные тесты. Должен быть создан генератор случайного потока инструкций. Генератор случайным образом генерирует последовательность разрешенных инструкций, которые поступают на тестируемое устройство и выполняются им. Данный вид теста показал очень высокую эффективность для сложнофункциональных микросхем, таких как микропроцессор 1890ВМ5Ф. После прохождения всех предыдущих тестов в данном микропроцессоре случайные тесты продолжали выявлять ошибки в логической модели. Тест можно одновременно запускать на десятках и сотнях компьютеров и тем самым многократно ускорять процесс тестирования (для тестирования 1890ВМ5Ф использовался 64-процессорный кластер и несколько десятков отдельных персональных компьютеров, то есть, достигнуто стократное ускорение тестирования).

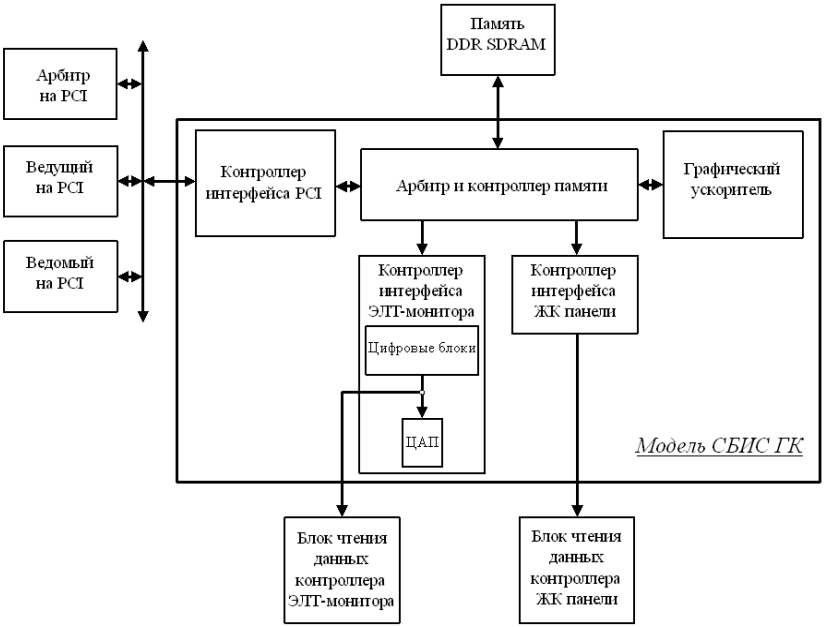


Рис. 5.2. Схема тестирования взаимодействия внутренних узлов микросхемы 1890ВГ10Т и ее взаимодействия с внешними устройствами.

8) Тестирование на аппаратном ускорителе (при наличии). Современные аппаратные ускорители стоят сотни тысяч – единицы миллионов долларов в зависимости от типа и конфигурации. Это удобный инструмент для отладки логической модели с развитыми средствами диагностики, визуализации, поиска ошибок. Однако не всегда удается протестировать модель микросхемы с полным ее окружением из-за отсутствия полноценных моделей окружения. Опыт разработки показал, что данный вид тестирования может рассматриваться как дополнительный, все ошибки, выявляемые данным инструментом, как правило, выявляются другими средствами. Однако ускорители позволяют до нескольких раз сократить время выявления некоторых ошибок.

9) Загрузка на моделях микросхем операционных систем (ОС) типа *Unix* и ОС реального времени для микропроцессоров или соот-

ветствующих драйверов для контроллеров. Выполнение прикладных задач. Процесс загрузки ОС на логической модели, выполняемый самыми высокопроизводительными компьютерами, может составлять от нескольких дней до нескольких недель в зависимости от сложности проекта, что не позволяет проводить эффективную отладку логической модели. Это связано с последовательностью выполнения всех операций. Для распараллеливания этой функции процесс загрузки можно разбить на отдельные кванты времени, каждый из которых выполняется на отдельном компьютере. Начальное состояние микросхемы для каждого кванта времени определяется выполнением соответствующей загрузки на поведенческой модели (это одна из причин, почему должно быть полное соответствие поведенческой и логической моделей). Таким образом, после исправления ошибки можно запустить параллельно работающие компьютеры и время всей загрузки ОС определяется только выделенным квантом времени.

10) Тестирование модели микросхемы на стенде с использованием микросхем *FPGA*. Должен быть создан прототип микросхемы на базе программируемых пользователями микросхемах *FPGA* и проведено тестированием прототипа под операционной системой с реальными пользовательскими программами. Данный вид тестирования может увеличить срок разработки микросхемы до полугода (прежде всего, для простых проектов объемом десятки-сотни тысяч транзисторов), однако для промышленных ЭВМ главным критерием является надежность работы. Тестирование же прототипных плат с микросхемами *FPGA* на прикладных задачах в выделенной области применения приводит к наиболее эффективному выявлению ошибок на заданных задачах и соответственно к повышению надежности функционирования аппаратуры на этих задачах. В рамках данного тестирования должны быть повторено выполнение архитектурных тестов и тестов всех функциональных узлов, загружены операционные системы типа *Linux* и ОС реального времени, исполнены возможные прикладные программы, прежде всего программы заказчика. Современные микросхемы *FPGA* позволяют загружать проекты микросхем объемом до нескольких десятков миллионов транзисторов. Микропроцессор 1890ВМ5Т, имеющий объем 27 млн.

транзисторов, может загрузиться в одну микросхему *FPGA* компании Altera *Stratix II EP*2*S*180F1020. Это очень эффективное тестирование, однако, для каждой микросхемы приходится разрабатывать свои технологические платы – прототипы требуемых плат. На первый взгляд данное тестирование приводит к заметному повышению трудоемкости и увеличению времени проектирования (как отмечалось выше, может достигнуть полугода). Однако данное тестирование позволяет найти ошибки, которые привели бы к необходимости повторного запуска изготовления кристаллов. В конечном итоге создание тестовых плат приводит к сокращению стоимости проектов без заметного увеличения времени проектирования.

11) Тестирование заказных и аналоговых узлов. Достижение соответствия функционирования данных узлов поведенческим моделям.

12) Тестирование модели микросхемы транзисторного уровня после топологического проектирования. Смешанное моделирование при наличии заказных и аналоговых узлов. Эти тесты готовятся на базе ранее подготовленных архитектурных тестов и тестов внутренних узлов, взаимодействия разрабатываемой микросхемы с микросхемами окружения. Основная задача данных тестов состоит в достижении соблюдения всех временных параметров функционирования микросхемы после перехода на транзисторный уровень и появлению паразитных элементов. Для достижения требуемых параметров иногда приходится возвращаться к изменению логической модели микросхемы. Смешанное моделирование не позволяет проводить детальное тестирование в силу большого времени выполнения данного тестирования, однако без такого тестирования сложно избежать ошибок в проекте.

13) Тестирование кристаллов и микросхем. Используется модернизированный (сокращенный) комплект тестов, созданный при разработке модели. Особую значимость приобретает выполнение системного и прикладного программного обеспечения на созданных микросхемах. Весьма эффективным тестированием является тестирование микросхем сторонними организациями. Такое тестирование можно разделить на сертификационное, то есть соответствие стандартам, и на уровне пользовательских задач.

Помимо функциональных тестов разрабатываются специальные тесты для измерения статических и динамических параметров микросхем.

Сертификационные тесты должны максимально возможно поставляться сторонними организациями, имеющими соответствующие права и лицензии. В ряде случаев приходится покупать дополнительное оборудование и программное обеспечения для выполнения данного тестирования.

Тесты для отбраковки кристаллов и микросхем направлены на поиск возможных дефектов при их изготовлении. Различают следующие средства тестирования: проверка массивов логических блоков, проверка массивов памяти, проверка аналоговых блоков. Основная задача средств тестирования — это выявление максимального количества технологических дефектов. При этом необходима минимизация площади элементов тестирования и самого времени тестирования.

При проверке массивов логических блоков преследуются три основные цели: проверка блоков на предмет отсутствия технологических дефектов, проверка правильности функционирования на максимальной частоте работы, проверка правильности функционирования совместно с аналоговыми блоками и блоками памяти.

Разработка проектов с использованием САПР на основе библиотек элементов дало широкие возможности по интеграции средств тестирования. Основой для проверки отсутствия технологических дефектов служат сканирующие цепочки и автоматически сформированные тестовые вектора. Дальнейшим развитием этого метода стала интеграция блоков генерации теста (генераторы предварительно определенных и псевдослучайных последовательностей) и анализа результата (например, сигнатурный регистр) в кристалле. Это позволяет значительно (в 10 – 1000 раз) сократить размер тестовых векторов и время тестирования, а также повысить эффективность. Однако это приводит к увеличению времени разработки тестовых векторов.

Проверка правильности функционирования на максимальной частоте, а также совместно с аналоговыми блоками решается с помощью блоков самотестирования (*LBIST*). Формирование входных воз-

действий на блок производится с использованием специальных генераторов тестовых последовательностей. Анализ выходных сигналов блока делается с помощью, например, сигнатурного регистра. Для отладки микропроцессоров и интеллектуальных контроллеров возможно использование пошагового выполнения инструкций и считывание результата, такой тест может проводиться на максимальной частоте [100].

Для тестирования массивов памяти используются блоки самотестирования памяти (*ABIST*). Такие блоки могут функционировать на максимальной частоте работы памяти. Однако в сложнофункциональных проектах зачастую приходится использовать специфические памяти для повышения производительности системы. Коммерческие САПР не покрывают такое разнообразие. Для решения подобных проблем возможно использование методики и средства построения блоков самотестирования памяти, рассмотренных в работе [100]. Для тестирования аналоговых блоков возможно управление и контроль работы их отдельных узлов с помощью специальных тестовых выводов кристалла [101].

Для упрощения и ускорения тестирования и отбраковки кристаллов и микросхем используется введение в кристалл сканирующих цепочек [102,103]. Такой способ основан на следующем. Все триггеры проекта устанавливаются в определенные значения. Затем, результат срабатывания комбинаторной логики, расположенной между триггерами, защелкивается в эти же триггеры. Далее анализируется состояние этих триггеров. Так проверяется отсутствие технологических дефектов в изготовленной схеме. Для загрузки триггеров и чтения их состояния все эти триггеры объединяются в сдвиговый регистр. В результате для получения тестопригодной схемы необходимо сделать в проекте три изменения. Первое, на входе триггера необходимо устанавливать дополнительный мультиплексор. Это необходимо для переключения входного сигнала: либо «рабочий сигнал» от комбинаторной логики, либо сигнал с выхода предыдущего триггера в сканирующей цепочке. Второе изменение - это разводка в кристалле дополнительных сигналов: сигналов переключения режима, сигналов связи триггеров в цепочку. Третье изменение - организация схемы переключения синхросигналов для прове-

дения тестирования. С точки зрения блоков предельного быстродействия, интеграция дополнительного элемента (мультиплексора) в комбинаторные тракты выглядит наиболее проблемно. Количество каскадов комбинаторных элементов в цепочках обработки данных, как правило, ограничено небольшой величиной (3 – 5 элементов). Максимальный запас, который может быть получен при оптимизации расстановки такого количества элементов гораздо ниже, по сравнению с трактами, где возможна установка десятков элементов для низкочастотной части схемы. Не смотря на то, что в библиотеках элементов присутствуют триггера с интегрированными мультиплексорами, этот мультиплексор дает ухудшение параметра setup триггера, что приводит к снижению максимальной частоты работы. Для решения данной проблемы возможно использование сигнатурного анализа. Реализация этого метода требует разработки дополнительных блоков: управления процессом тестирования, генератора тестовых последовательностей, устройства, осуществляющего сжатие выходных данных конвейера в код, называемый сигнатурой. На рисунке 5.3 представлен пример такого тестирования контроллера интерфейса ЭЛТ-монитора, входящего в графический контроллер 1890ВГ14Т [104]. Тестирование осуществляется следующим образом. Программируются регистры управления контроллером (размер выводимого на экран изображения, глубина представления цвета и частота пиксельного синхросигнала, генерируемого синтезатором). Далее блок управления запускает режим тестирования на время равное выводу на экран одного кадра изображения. В течение этого интервала времени данные, подаваемые в конвейер обработки данных, формируются генератором псевдослучайных чисел (ГПСЧ). А сигнатурный регистр (СР) осуществляет сжатие выходных данных конвейера в сигнатуру. По окончании режима тестирования производится сравнение полученной сигнатуры с эталонным значением для данного режима работы конвейера и для использованной псевдослучайной последовательности данных. Совпадение двух этих результатов говорит о работоспособности конвейера. Дополнительно можно ускорить функционирование всего тракта разработкой специальных триггеров в дополнение к библиотечным для подключения сигналов либо от сигналов тестирующей цепочки, либо от функциональной логики микросхемы.

Проведение испытаний микросхем и окончательное определение их параметров проводятся в соответствии с существующими стандартами.

5. Разработка логической модели уровня *RTL* ведется на языках *Verilog* или *VHDL*. Разработка модели предполагает следование созданным на предыдущих этапах документам – архитектура микросхемы, и техническое задание. Для достижения требований ТЗ в ряде случаев приходится делать итерационный процесс по разработке архитектуры, то есть, если ТЗ выполнить не удается по каким-либо параметрам, приходится возвращаться на этап разработки архитектуры. Другая возможность улучшения показателей микросхемы – написание моделей отдельных узлов на уровне *Gate* (уровне простейших библиотечных элементов). Однако это может привести к увеличению времени проектирования этих узлов в несколько раз. На этом же этапе должны быть разработаны также логические модели всех заказных узлов. При разработке заказных узлов необходимо достичь полного соответствия функционирования этих двух моделей. На этапе создания логической модели возможно также определение дополнительных узлов, которые должны быть разработаны заказным путем. Верификация модели возможно производить с использованием технологических плат на базе микросхем *FPGA* и тестового ПО.

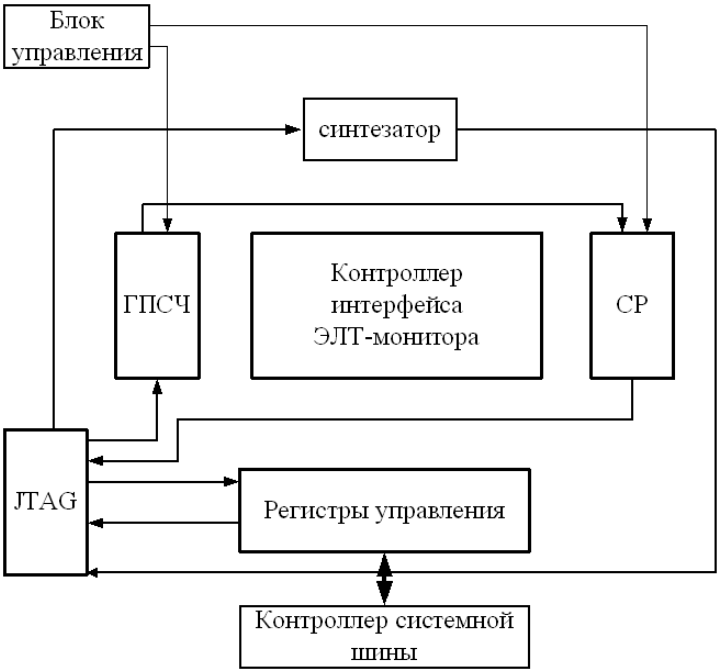


Рис. 5.3. Схема тестирования контроллера интерфейса ЭЛТ-монитора.

6. На этапе уточнения технологии производства, выбора библиотек и *IP*-блоков происходит определение дальнейшего маршрута проектирования и оптимизация микроархитектуры проектируемой микросхемы. Особое место на этом этапе занимает проектирование сложнофункциональных микросхем. Одним из основных вопросов при создании таких микросхем является разработка блоков предельного быстродействия, взаимодействие их с блоками меньшего быстродействия и аналоговыми блоками. В настоящее время для решения задач такого класса применяются два основных подхода:

- использование заказного проектирования при разработке блоков предельного быстродействия;
- разработка специализированных САПР для синтеза и оптимизации проекта на основе специально разработанных библиотек элементов.

В обоих этих подходах используется оптимизация на уровне архитектуры всей системы и архитектуры отдельных блоков. Первый подход отличается тем, что в нем используется весь спектр доступных способов по оптимизации проекта на всех стадиях от разработки архитектуры до получения топологии. Существенными недостатками такого подхода являются высокая стоимость, большое время разработки, сложность быстрого перехода на новую технологию. Второй подход характеризуется разработкой и использованием набора САПР и технологических библиотек элементов для проектирования микросхем выделенного класса. Такой подход позволяет приблизиться к быстродействию полностью заказных схем с проигрышем (в пределах 10-50 %) по току потребления и занимаемой площади. Недостатком такого подхода является то, что необходимы большие вложения в разработку специализированных САПР и технологических библиотек. Однако такие библиотеки и САПР являются универсальными для создания микросхем выделенного класса.

Заказное проектирование широко использовалось при проектировании многими компаниями, в том числе для проектирования микропроцессоров. Принципы и методика заказного проектирования изложены, в частности, в работах [104]. На рис. 5.4 представлена обобщенная методика заказного проектирования.

Эта обобщенная методика состоит из двух основных этапов. Первый - разработка архитектуры микросхемы и микроархитектуры блоков. Второй - это детальная разработка отдельных блоков, сборка блоков и проверка быстродействия на уровне микросхемы. Первый этап характеризуется использованием высокоуровневых средств разработки архитектуры. В то время как на втором этапе используется весь арсенал доступных средств заказного проектирования для разработки и оптимизации блоков с учетом особенностей технологии. Отличительной чертой методики заказного проектирования является то, что после детальной проработки заказных блоков нет возможности вернуться к этапу разработки архитектуры. Это связано с большими затратами времени и средств на разработку заказных блоков.

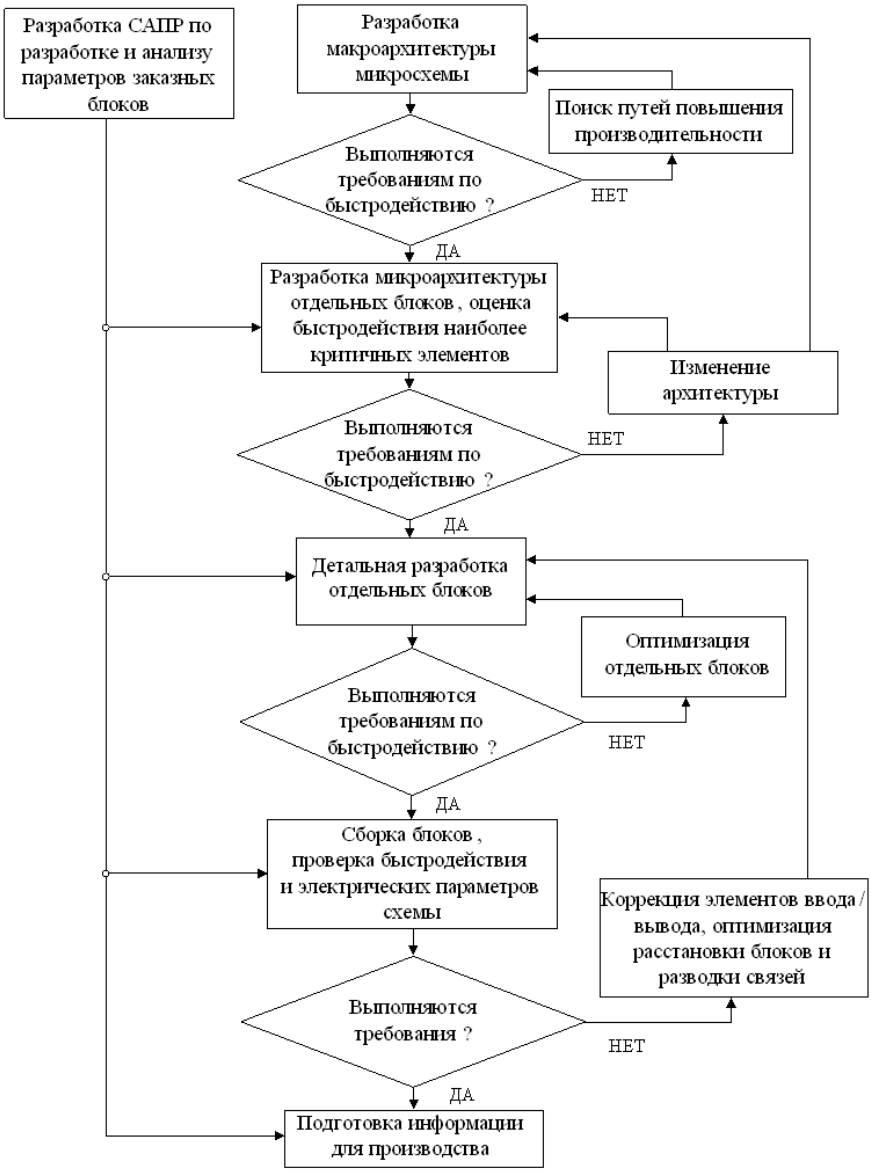


Рис. 5.4 Блок-схема методики заказного проектирования.

При использовании автоматизированного проектирования достижение требуемого быстродействия осуществляется за счет автоматического синтеза и разработки топологии на основе специализированных библиотек элементов и макроблоков, оптимизации на этапах разработки схемы и топологии специализированными САПР. Библиотечные элементы должны оптимизироваться под выделенные задачи. Например, при разработке микропроцессоров 1890ВМ5Ф и 1890ВМ6Я были разработаны в дополнение к существующим триггерам специализированные триггера, оптимизированные для организации селективного отключения блоков. Такой подход используется для проектов, которые невозможно разработать за приемлемое время и средства с использованием средств заказного проектирования. Недостатком такого подхода является то, что необходимы большие вложения в разработку специализированных САПР и библиотек. Однако такие библиотеки и САПР являются универсальными для создания микросхем выделенного класса. На рисунке 5.5 показана обобщенная методика автоматизированного проектирования.

Наиболее оптимальной является методика, в основе которой лежит использование коммерчески доступных САПР совместно с заказным проектированием отдельных блоков. Для достижения эффекта, сопоставимого с применением специализированных САПР и заказного проектирования, возможно выделение блоков разного быстродействия с оценкой возможности автоматического синтеза для каждого блока. Далее необходимо использование методики синтеза, методики сокращения цепочки буферов для большого числа нагрузок, установленных на выходе одного триггера, методики определения наиболее критичных параметров дерева синхронизации и выделения элементов, требующих ручной расстановки.

Выделение блоков предельного быстродействия, которые можно получить средствами синтеза, можно производить на основе коэффициента количества элементов $К_Э$: $K_Э = N_C/N_П$, где N_C - оценочное количество двухвходовых элементов типа И-НЕ/ИЛИ-НЕ, получаемых средством синтеза и необходимых для реализации самой длинной цепочки обработки данных в разрабатываемой модели; $N_П$ - максимальное количество таких же двухвходовых элементов, которое может быть использовано для построения цепочки обработки дан-

ных, работающей на требуемой максимальной частоте передачи сигнала. Данный коэффициент оценивается для всех блоков проекта. Это позволяет разделить проект на блоки, разрабатываемые средствами синтеза и с использованием заказного проектирования.

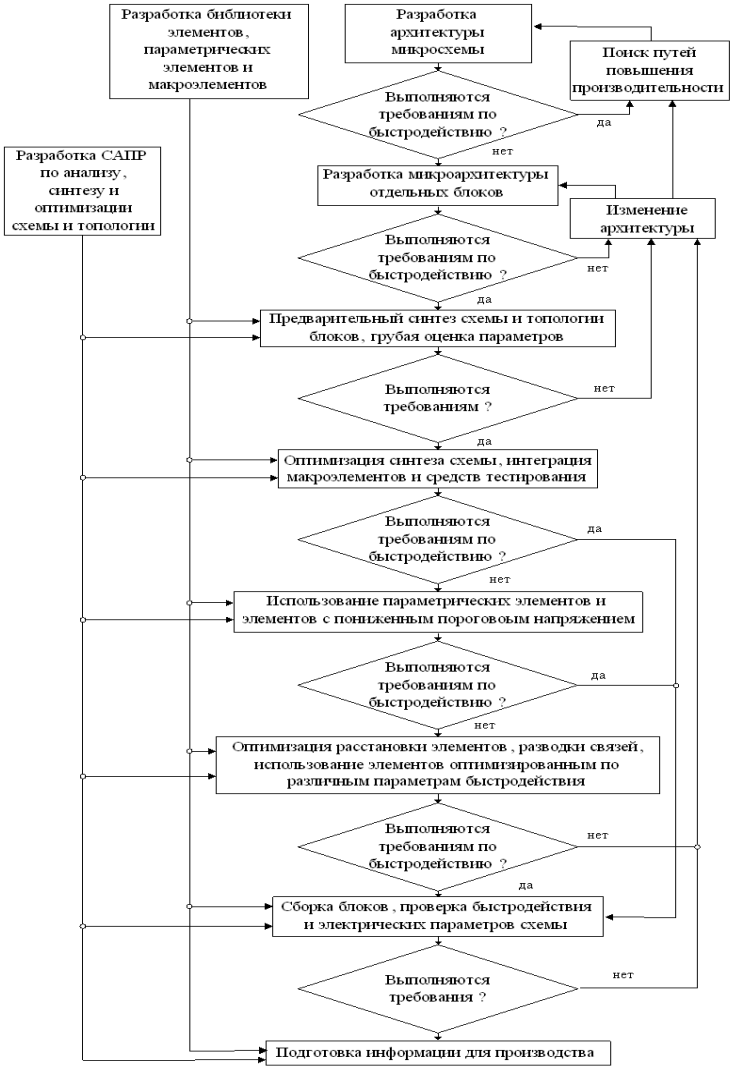


Рис. 5.5. Обобщенная методика автоматизированного проектирования

7. Создание принципиальной электрической схемы на уровне транзисторов является итерационным процессом. Сначала проводится предварительный синтез. Этот этап позволяет получить более точные, по сравнению с этапом разработки архитектуры, оценки производительности, площади и энергопотребления. На этом этапе выделяются высокоскоростные цифровые блоки, требуемое быстродействие которых возможно достичь за счет специализированного маршрута проектирования, такого как: разделение блоков предельного быстродействия на подблоки, ручная расстановка, маршрут синтеза.

Для блоков, имеющих нарушение быстродействия, необходимо провести анализ архитектуры с целью выделения подблоков, имеющих нарушения по быстродействию. На основании этих данных проводится разделение проекта на подблоки. Использование итерационного синтеза подблоков позволяет значительно сократить сроки выбора оптимального решения с целью достижения требуемого быстродействия. Применение специализированного маршрута разработки для подблоков позволяет приблизиться к предельному быстродействию без использования заказного проектирования.

Методика синтеза основана на использовании ряда подходов, позволяющих добиться требуемого быстродействия, которое невозможно получить стандартными способами и в заданные сроки. Требуемое быстродействие достигается, прежде всего, за счет следующих дополнительных этапов работ:

- поиск путей распространения сигналов, не требующих срабатывания за один период синхросигнала;
- выделение подблоков, работающих в режимах, в которых частота синхросигнала снижена в несколько раз по сравнению с максимальной;
- сокращение количества триггеров, находящихся в конце цепочек комбинаторных элементов, нагруженных на один выход триггера;
- сокращению количества элементов в цепочке между триггерами;

• минимизация расфазировки синхросигналов и фронтов синхросигнала.

Сокращения цепочки буферов, установленной на выходе одного триггера, проводится следующим образом. Создается несколько управляющих триггеров и проводится перераспределение нагрузки между ними. Это позволяет при разработке топологии получить оптимальное распределение элементов по площади кристалла и добиться требуемого быстродействия.

Если предпринятые действия не приводят к получению требуемых показателей быстродействия, проводится изменение архитектуры блоков и принципов передачи данных между блоками.

После окончания синтеза микросхемы необходимо провести верификацию списка соединений на соответствие логической модели уровня RTL.

8. Топологическое проектирование разделяется на заказное, к которому можно также отнести и аналоговое, и автоматическое. Если в синтезируемом проекте имеются отдельные заказные узлы, сначала размещаются заказные узлы и затем проводится топологическое проектирование всего кристалла. Этот процесс, как правило, итерационный для достижения требуемых параметров. Трассировка цепей проводится в два этапа: сначала выполняется трассировка шин «земли» и «питания», а затем трассировка сигнальных цепей. Очень важно при создании сетки земли и питания добиться минимальной просадки питания на всём протяжении кристалла. Просадка по питанию приводит к снижению быстродействия микросхемы. Современные технологии корпусирования кристаллов позволяют добиться относительно равномерного распределения напряжения питания по кристаллу. Это так называемая технология *flip-chip*. На верхнем слое металла кристалла наносятся металлические столбики – контакты кристалла. На корпусе микросхемы имеются соответствующие ответные контакты. При корпусировании контакты корпуса совмещаются с контактами кристалла, что позволяет минимизировать паразитные элементы и добиться качественной земли и питания по всему корпусу, а также поднять быстродействие высокоскоростных каналов микросхемы.

Топологическое проектирование аналоговых блоков отличается необходимостью дополнительного моделирования соответствующими средствами САПР и требованиями существенно более качественной земли и питания.

Если в процессе разработки топологии требуемое быстродействие подблоков не достигается, необходимо определить величину расхождения и в зависимости от ее величины предпринять соответствующие действия. Если расхождение составляет единицы процентов, то его, как правило, удается устранить изменением параметров, задаваемых при топологическом проектировании критичных подблоков. Если расхождение составляет порядка 10%, необходимо определить перечень критичных цепей и определить длину цепочек и нагрузочные характеристики буферов в этих цепочках. Дальше необходимо провести оптимизацию этих цепочек. Эффективным способом оптимизации является подгонка нагрузочных способностей выходных буферных элементов под реальные нагрузки, что позволяет в ряде случаев также сократить число элементов в цепочке.

Следующим шагом является ручная расстановка следующих элементов:

- подблоков формирования сигналов выходных шин;
- подблоков приема данных от входных шин.
- элементов, используемых для формирования синхросигналов с кратными частотами.

Для таких блоков используются следующие рекомендации по расстановке. Для подблоков, формирующих выходные сигналы шин, элементы формирования выходных разрядов необходимо устанавливать рядом с площадочными элементами соответствующих разрядов. Для подблоков, принимающих входные сигналы шин, элементы приема входных разрядов необходимо устанавливать также рядом с площадочными элементами соответствующих разрядов. Для шин, у которых критичным параметром является задержка от входного синхросигнала до выходных данных и не используется ФАПЧ (фазовая автоподстройка частоты), необходимо строить отдельное дерево синхросигнала для выходных триггеров. Элементы, формирующие синхросигналы с кратными частотами, необходимо

устанавливать рядом с источником синхросигнала и на минимальном расстоянии друг от друга.

Это позволяет:

- сократить задержку переключения выходных сигналов и минимизировать ее разброс;
- повысить запас по параметрам *setup* и *hold* для входных данных относительно фронта синхросигнала, минимизировать его разброс;
- минимизировать разброс дерева синхронизации триггеров, используемых для приема входных данных с шин, и для формирования выходных данных шин.
- минимизировать расфазировку синхросигналов с кратными частотами.

Если принимаемые шаги не приводят к достижению требуемого быстродействия, происходит возврат на этап синтеза или этап создания логической модели.

После проведения топологического проектирования необходимо выполнить следующие действия:

- верифицировать топологию на соответствие проектно-конструкторским нормам;
- экстрагировать паразитные элементы;
- верифицировать принципиальную схему с учетом реальных задержек в топологии.

Как показывает практический опыт, перед отправкой на завод топологии кристаллов важны не только проверки *DRC* (*Design Rule Check*), *LVS* (*Layout Versus Schematic*), *ANTENNA*, плотность металлов, обычно предоставляемые заводом-изготовителем, но и проверка на закоротки – так называемая *ERC* (*Electrical Rule Check*) проверка. Причем, по своему составу, ни одна из перечисленных выше проверок не перекрывает *ERC*. Эта проверка особенно актуальна, когда в одном кристалле присутствуют нескольких шин «питания» и «земля». Обычно, такая ситуация бывает в проектах, где есть заказные аналоговые блоки, которым необходимы не зашумленные «земля» и «питание», и цифровые на библиотечных элементах, либо в проектах содержащих высоковольтные и низковольтные транзи-

сторы. Также положительный результат эта проверка дает в больших цифровых проектах, где невозможен надежный визуальный контроль.

Дополнительные проверки можно проводить с использованием четырех *ERC* проверок:

1) Проверка наличия закороток между «питанием» и «землей» по металлам на уровне элементов. Проводимость объявляется только по металлическим межсоединениям элементов.

2) Проверка наличия закороток между «питаниями» и «землями» по металлам и диффузиям на транзисторном уровне. Проводимость объявлена по металлам и диффузиям.

3) Проверка наличия закороток между «питаниями» и «землями» по *N*-карману и подложке. Проводимость объявлена по металлам, диффузиям, *N*-карману и подложке. При этой проверке часто встречаются следующие ошибки: разрыв по металлу между контактной площадкой «питания» и *N*-карманом, разрыв по металлу между контактной площадкой «земли» и истоками n-транзисторов.

4) В последней проверке объявлены транзисторы, диоды, резисторы, емкости. Присутствует проводимость по всем проводящим слоям, включая все слои поликремния. Проводятся проверки на:

- неподсоединенные затворы, стоки, истоки, подложки транзисторов к схеме.
- неподсоединенные узлы к «земле» и «питанию» через один или более транзистор, резистор, диод.

Несмотря на простоту проверок, они позволяют в ряде случаев найти дополнительные ошибки.

Стоимость изготовления кристаллов при небольших партиях (6 - 24 пластин) определяется в основном стоимостью фотошаблона. Для технологических норм 0, 13 – 0,18 мкм стоимость комплекта фотошаблонов составляет $80 – 150 тыс. При наличии в проекте кристалла цифровых микросхем небольшого количества ошибок исправление этих ошибок можно произвести дополнительным использованием нескольких ячеек, что позволяет избежать или уменьшить число изготовлений комплекта фотошаблонов.

Существует два способа исправления ошибок:

- исправление ошибок в кристалле микросхем за счет изменения топологий только отдельных фотошаблонов металлов и сохранения всех остальных фотошаблонов;

- исправление ошибок специальным прибором *FIB* – источником ионного пучка шириной до нескольких нанометров, позволяющего делать операции распыления или нанесения диэлектриков или металлов на кремневых пластинах.

Однако возможности таких операций относительно небольшие без наличия на кристалле специализированных логических элементов.

Для реализации указанных возможностей возможно использование специальной библиотеки элементов для отладки кристаллов микросхем, позволяющей эффективно исправлять ошибки обоими способами, указанными выше.

Эффективным комплектом библиотека являются следующие элементы:

1. Программируемая ячейка, содержащая *D* – триггер и ПАЛ с 4 входами, подобная ячейкам в *FPGA* компании *Altera*. ПАЛ программируется разводкой в верхних слоях металлов в полуавтоматическом режиме в соответствии с требуемой функцией.
2. Буфер клокового дерева.
3. Буфер.
4. Ячейка 2И-НЕ.
5. Ячейка 2ИЛИ-НЕ.
6. Инвертор.
7. *D*-триггер
8. Цепочка элементов 2И-НЕ и «ИЛИ-НЕ».

В проектах должны быть предусмотрены возможности топологической коррекции кристаллов микросхем с использованием данной библиотеки. Для этого должны предусматриваться области соединения с данными элементами или разрезки цепочек элементов с использованием *FIB* (пятачки запрета топологии, с диаметром, соответствующим ширине ионного пучка и возможностью различения нижних слоев металлов). Целесообразно предусматривать возможность вывода некоторых точек соединений элементов на верхний слой металла.

Размещения элементов библиотеки на кристалле должно зависеть от его размера, технологии, числа транзисторов, типа функциональных узлов (память, логика, аналоговые элементы и пр.).

Число таких элементов определяется размером кристалла и обычно составляет порядка нескольких сотен штук. Первоначально затворы транзисторов таких ячеек подсоединяют к шинам «земли» или «питания». Если необходимо сделать небольшую коррекцию электрической схемы после изготовления микросхемы, то можно задействовать «ремонтные» элементы, сделав изменения только в слоях металлов и тем самым сократить число изменяемых масок. С учетом того, что стоимость изготовления кристаллов при небольших партиях (6-24 пластин) для технологических норм ниже 0,18 мкм определяется в основном стоимостью фотошаблонов, данный способ позволяет существенно сократить материальные затраты на изготовление исправленной версии СБИС. Такой подход позволяет также ускорить подготовку документации для изготовления кристаллов и их изготовление от 1 до 2 месяцев.

9. Тестирование изготовленных кристаллов и микросхем разделяется на этапы исследовательский и производственный. Исследовательский этап направлен на поиск возможных ошибок, производственный - на выявление дефектов при изготовлении. На исследовательском этапе используется комплект тестов, созданный при разработке модели. Особую значимость приобретает выполнение системного и прикладного программного обеспечения. Весьма эффективным тестированием является тестирование микросхем сторонними организациями. Такое тестирование можно разделить на сертификационное, то есть соответствие стандартам, и на уровне пользовательских задач. Для производственного тестирования во все проекты, как правило, вводятся средства тестирования через порт JTAG, сканирующие цепочки и встраиваемые средства самотестирования. Проведение испытаний микросхем и окончательное определение параметров микросхем проводятся в соответствии с существующими стандартами.

5.2. Разработка заказных узлов

Наиболее простой и хорошо формализованной является методика проектирования на основе стандартных библиотек (cell based design). Правда, характеристики кристаллов по частоте, площади и потребляемой мощности в этом случае получаются довольно низкими. Например, максимальная рабочая частота таких микросхем как микропроцессоры, изготовленных по технологии 0,18 мкм, составляет порядка 200 МГц [106]. Такой подход и получаемые при этом рабочие частоты вполне оправданы при проектировании таких микросхем, как периферийные контроллеры.

Метод полностью заказного проектирования (full custom design) характеризуется тем, что:

- транзисторы являются основным элементом для проектирования схемы;

- каждый транзистор в схеме может иметь (и часто имеет) индивидуальный размер, оптимизированный для выполнения конкретной функции.

В высокоскоростных цифровых блоках с большой разрядностью (сумматорах, умножителях и т.д.), особенно при применении таких логических стилей проектирования, как “домино", динамического, динамического дифференциального с уменьшенной амплитудой сигнала, важна прецизионная оптимизация нагрузочной способности каскадов, что и является причиной проектирования схем на транзисторном уровне. При этом одновременно, в одном проекте, может применяться как простая статическая логика, так и различные виды динамической логики, дифференциальная логика, логика на проходных транзисторах и логика с малым перепадом [107]. Использование метода полностью заказного проектирования позволяет достигать почти в два раза большего быстродействия [108], существенно экономить потребляемую мощность и в зависимости от количества уровней металлизации в два-три раза уменьшить площадь схемы.

Два существенных ограничения препятствуют повсеместному внедрению полностью заказного проектирования кристаллов:

1) Высокие затраты на проектирование. При отсутствии формальных методов и при очень большом числе варьируемых параметров требуется большое количество высококвалифицированных специалистов с опытом проектирования заказных схем и топологии для быстродействующих микросхем;

2) Метод полностью заказного проектирования предъявляет очень высокие требования к системам верификации. Это относится как к системе логической верификации проекта, поскольку одни и те же логические функции могут быть реализованы множеством различных транзисторных схем, так и к системе верификации физического дизайна, поскольку динамические схемы, логика с малым перепадом существенно повышают требования к качеству физического дизайна. Для контроля специфических ограничений необходимы оригинальные программные средства. Однако разработка оригинальных программных средств требует длительного периода времени и больших материальных средств (средства автоматизации должны быть ориентированы на конкретное технологическое производство).

Заказные блоки и элементы библиотеки проходят этапы верификации и характеризации, после чего передаются на следующий этап сборки кристалла [109]. Причем базовые библиотечные элементы используются для автоматического и ручного синтеза блоков, так как известно, что цифровые схемы в глубоком субмикроне значительную часть быстродействия (в технологии 0,18 мкм до 40%) теряют на разводке.

После сборки всего кристалла проводится верификация физического дизайна (*DRC, ERC, LVS* проверки) [110], экстракция паразитных *RC*-цепей в межсоединениях [111]. На основе моделирования с учётом *RC*-цепей выявляются критические цепи и элементы, у которых нарушены временные соотношения, вручную вносятся исправления и процесс повторяется несколько раз до достижения требуемых параметров проекта. Такой подход имеет ряд преимуществ с точки зрения применяемых программных средств автоматизированного проектирования:

1) Могут быть использованы независимые программные средства различных производителей систем автоматического проектирования. Причем, каждая программа может работать с собственной базой данных и использовать различные стандарты представления информации. Интерфейс между программами осуществляется файлами, представленными в стандартных для отрасли форматах;

2) Этот подход позволяет применять средства проектирования, адекватные проектируемым блокам. Если требуется полностью заказное проектирование для обеспечения параметров блока, то применяются средства заказного проектирования, если блок разрабатывается с использованием стандартных библиотек, то применяют набор соответствующих программных средств.

В частности, в микропроцессоре 1890ВМ5Ф [112] были выделены наиболее критичные блоки по быстродействию: АЛУ (арифметико-логическое устройство), блок *TLB* (ассоциативная память для трансляции виртуальных адресов в физические), регистровые файлы, блок вычисления адресов. Далее было произведено разбиение данных блоков на отдельные блоки и показано, что полностью заказными необходимо делать только блок *TLB* и регистровые файлы, остальные блоки можно реализовать средствами САПР при условии создания быстродействующих заказных сумматоров с различным числом разрядов. Было показано, что при синтезе блоков целочисленного АЛУ в автоматическом режиме средствами САПР рабочая частота получалась около 200 МГц. Время синтеза занимало от 0,5 до 1,5 часов при различных вариациях исходной информации (логическая модель на языке *Verilog*). Примерно такое же время занимает создание топологии. Однако для достижения указанной частоты необходимо прогнать данный процесс несколько раз, выявляя критичные узлы и дорабатывая соответствующим образом модель. При полностью заказном проектировании время создания такого блока (с количеством транзисторов в блоке 70-80 тысяч) занимает более 8 человеко-месяцев, причём большая часть уходит на проектирование топологии. Рабочая частота возрастает до 400 МГц в диапазоне температур -60 - +125° С. При использовании заказных сумматоров и дальнейшем синтезе блока АЛУ частота становилась равной 300 МГц, сроки разработки при этом уменьшались в 3 раза. Тем самым возможно достичь существенного сокращения сроков разработки и требуемых ресурсов при относительно невысокой потери быстродействия. Сокращение критичных по быстродействию цепей в микропроцессоре можно также достичь разработкой специализированных динамических триггеров для организации функции блокирования клоков, требуемой для сокращения потребления питания микросхемы. Таким образом, для поднятия частоты всего проекта в

целом библиотека стандартных элементов должна дополняется динамическими логическими элементами.

При проектировании заказным образом в стиле «домино» необходимо очень тщательно, в отличие от статического проектирования, организовывать конвейер. Типичная структура «домино» конвейера показана на рис.5.6 и рис.5.7. «Домино» конвейер начинается с входных триггеров, которые используются для преобразования входных статичных логических сигналов в монотонные сигналы, необходимые для «домино» логики. Для сохранения результатов работы динамических каскадов в конце первого полутакта используются промежуточные триггеры типа «защелка». Выходные триггеры удерживают выходные сигналы «домино» конвейера в неизменном состоянии в течение следующего полутакта, что необходимо для правильной работы последующих блоков.

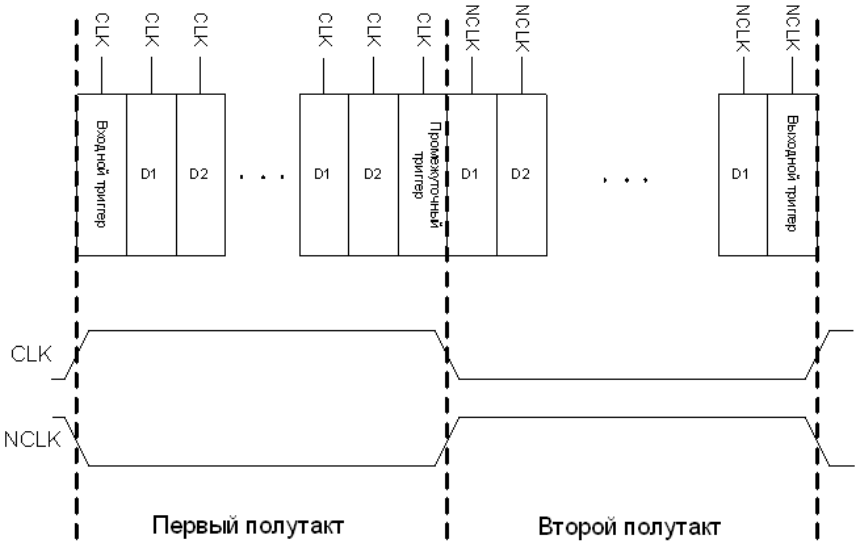


Рис. 5.6. «Домино» конвейер с идеальными тактовыми сигналами

Как видно из рисунка 5.6, одна часть каскадов находится в фазе предзаряда, в то время как другая - в фазе срабатывания и наоборот. Такой режим работы «домино» конвейера достигается за счёт введения дополнительного тактового сигнала *NCLK*, находящегося в противофазе с тактовым сигналом *CLK*. Добавление дополнительного

тактового сигнала увеличивает сложность дизайна, но позволяет существенно увеличить производительность. На рис. 5.6 изображен идеальный случай, когда сигналы *CLK* и *NCLK* находятся строго в противофазе относительно друг друга. В течение первого полутакта идет фаза срабатывания, за которую нужно успеть защелкнуть сигналы на выходах промежуточного триггера до наступления фазы предзаряда. Такой режим работы создает жесткие границы по времени для прихода данных.

В действительности, идеальных тактовых сигналов не существует (см. рис. 5.7) - возникает их расфазировка, что обусловлено такими факторами как: существование неравномерной нагрузки в цепях тактовых сигналов, варьирование уровней напряжений питания и т.д. Обычно фаза срабатывания длится половину периода тактового сигнала, но с появлением расфазировки она уменьшается и в наихудшем случае $T - 2 \cdot t_{skew}$, где t_{skew} - время расфазировки двух тактовых сигналов.

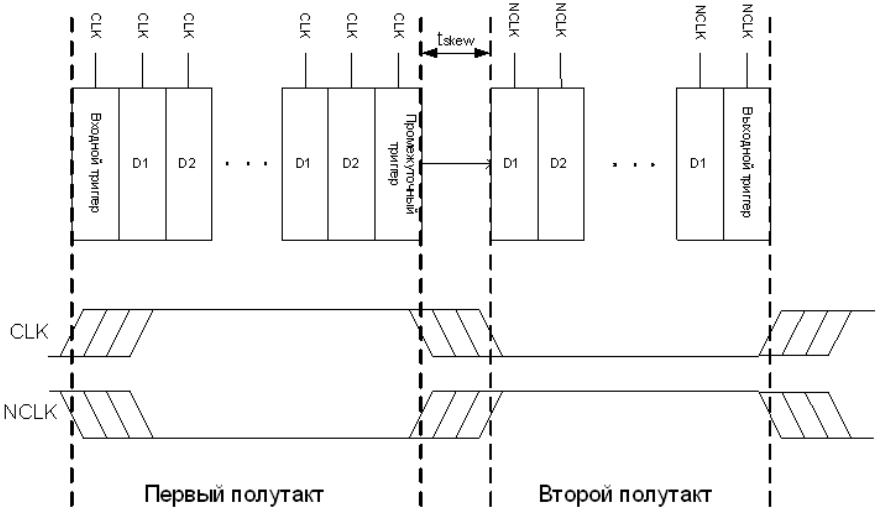


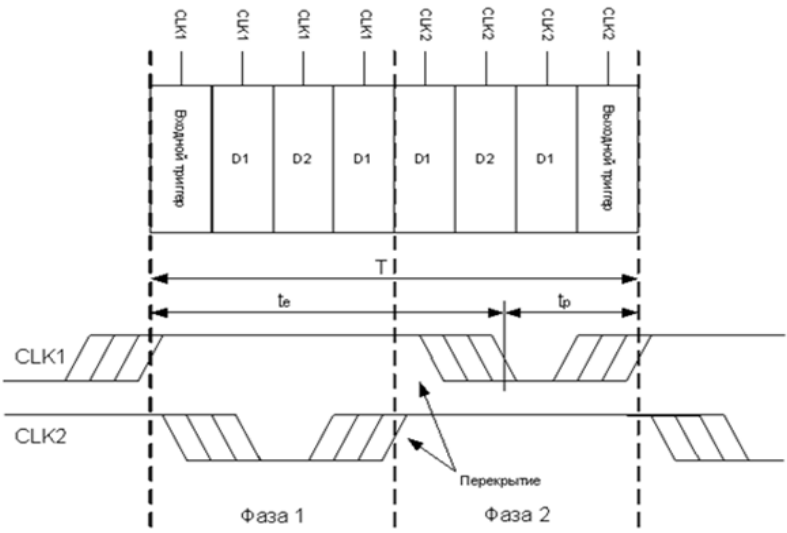
Рис. 5.7. «Домино» конвейер с неидеальными тактовыми сигналами

Необходимо добавить, что длительность фазы срабатывания при такой конфигурации конвейера еще больше уменьшается из-за задержек, добавляемых триггерами. Время фазы срабатывания определяется временем работы динамической логики в каждой из двух

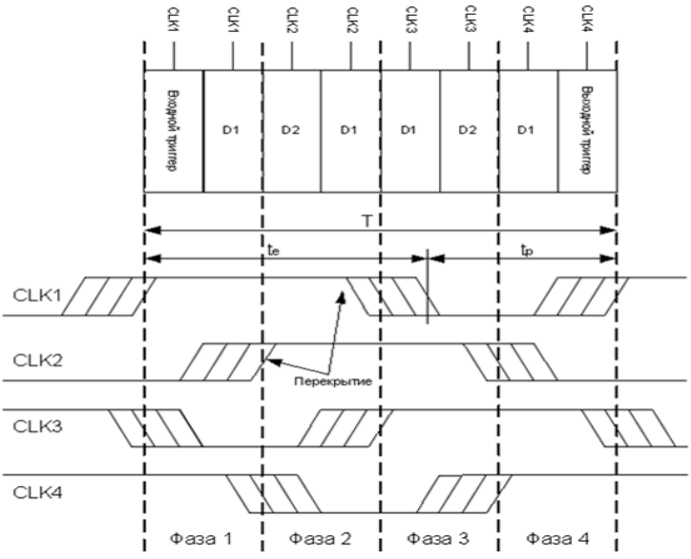
секций, наибольшая из них определяет итоговую тактовую частоту устройства. Также важно заметить, что динамические каскады должны успеть полностью перезарядиться в течение фазы предзаряда, до того, как начнется следующая фаза срабатывания.

Чтобы смягчить требования к временным соотношениям применяется метод, когда два и более тактовых сигналов работают с перекрытием [113]. После того как результат сработавшего динамического каскада будет принят следующим каскадом, цепи предыдущего можно переводить в фазу предзаряда.

На рис.5.8 изображен пример двух и четырех перекрывающихся тактовых сигналов. В такой схеме тактирования необходимо рассчитать приход фронтов тактовых сигналов таким образом, чтобы результат срабатывания каскада успел прийти на вход следующего каскада до наступления фазы предзаряда. Также необходимо учитывать время расфазировки между тактовыми сигналами. Расфазировка в такой схеме тактирования никогда не будет влиять на работу критических путей, так как все каскады успеют сработать до наступления фазы предзаряда. Более того, промежуточные триггеры уже больше не будут нужны, что дополнительно увеличит быстродействие всего конвейера.



а



б

Рис. 5.8. Пример двух- и четырехфазных перекрывающихся тактовых сигналов.

Рассмотрим *N* перекрывающихся тактовых сигналов с периодом *T*. Вся работа динамического конвейера разделится на N фаз. Передние фронты тактовых сигналов в каждой из последующих фаз будут приходить через время, равное *T/N*. Время срабатывания должно занимать более 50 % длительности периода тактового сигнала при двух перекрывающихся тактовых сигналах, но с тремя и более тактовыми сигналами её длительность может быть и *T*/2. Обозначим время, необходимое для срабатывания как t_e. Время t_p, необходимое для предзаряда, определяется следующим образом (см. рис.5.9 а): каскад *A* должен успеть полностью перезарядиться и изменить состояние на своем выходе с «1» на «0» за время t_{prech}. Наихудший случай расфазировки t_{skew} будет наблюдаться при раннем приходе заднего фронта сигнала *CLK*1 и при запаздывании заднего фронта сигнала *CLK*2. Так можно определить нижнюю границу величины t_p, а именно $t_p \geq t_{prech} + t_{skew}$.

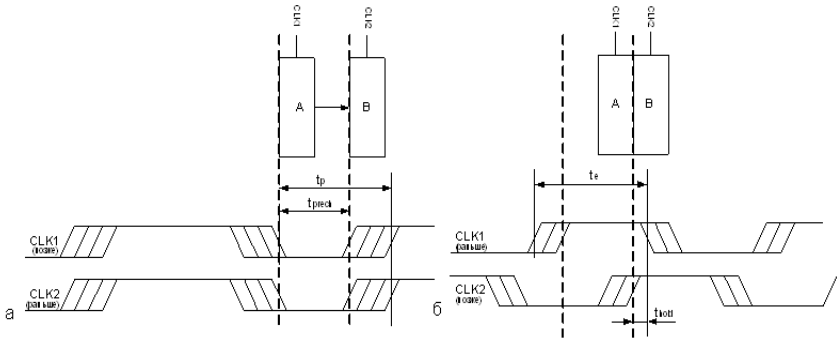


Рис.5.9. Временных ограничений на срабатывание (а) и предзаряд (б)

Подобным образом (см. рис.5.9 б) определяется время t_e, необходимое для срабатывания. Тактовый сигнал *CLK*1 должен остаться в «1» пока результаты работы каскада *A* не воспримутся следующим каскадом B за время t_{hold} (минимальное перекрытие для *CLK*1 и *CLK*2). Минимальное время перекрытия с учетом расфазировки будет $t_{hold} + t_{skew}$. Добавляя полученное выражение ко времени *T/N*

между фазами можно определить нижнюю границу величины t_e, а именно $t_e \geq t/N + t_{skew} + t_{hold}$. Используя последние два выражения для t_p и t_e, можно найти отношение между длительностью тактового цикла и максимально допустимой расфазировкой: $t_{skew-\max} = 0{,}5[T(N-1)/N - t_{hold} - t_{prech}]$

Для больших *N* и *T* максимально возможное значение расфазировки может составлять *T*/2. Малые значения *N* уменьшают это значение из-за уменьшения перекрытий тактовых сигналов. Наиболее предпочтительно выбирать *N*=2 или *N*=4 из-за хорошего компромисса между устойчивостью конвейера к расфазировкам и относительной легкости генерирования необходимых тактовых сигналов проводящего канала от шины питания или «земли» на выход. Каскад, обозначенный на рис. 5.10 как *D*1, представляет собой стандартную ячейку «домино» логики. Добавление дополнительного транзистора для разряда N_1 уменьшает быстродействие каскада и увеличивает нагрузку на тактовый сигнал; чтобы ее немного снизить, в каскаде *D*2 используется видоизмененная «домино» логика без транзистора N_1, что позволяет уменьшить нагрузку на тактовый сигнал и увеличить скорость срабатывания логической функции.

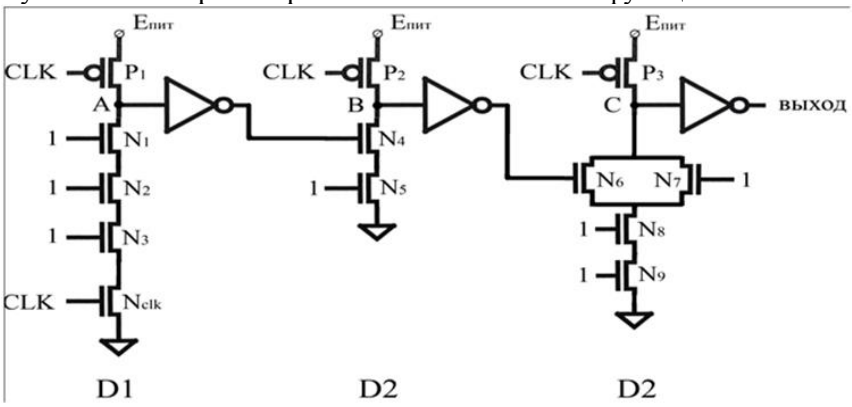


Рис. 5.10. Последовательное включение каскадов *D*2

По этим же причинам всегда необходимо ставить каскад D1 после триггерных элементов, последние могут удерживать сигналы своих выходов в неизменном состоянии в течение фазы предзаряда.

Использование динамической логики (ДЛ) для реализации наиболее критичных узлов микропроцессора, дает возможность достичь максимального быстродействия, однако она обладает высокой чувствительность к шумам и высоким энергопотреблением. Снизить потребление, сохранив при этом достигнутое быстродействие, можно за счет использования полудинамической схемотехники. На рисунке 5.11 показан переход от полностью динамического дифференциального сумматора [114] к полудинамическому.

Рис.5.11. Структурная схема быстродействующего сумматора

Функциональный узел «конечная сумма» требует прямой и инверсный сигналы, следовательно, все предшествующие ему блоки должны быть дифференциальными, так как ДЛ не может обеспечить инверсию внутри конвейерного полутакта. Если конечный каскад реализовать на статической логике, то остальные узлы можно перевести в разряд однофазной динамики, при этом не критичный по быстродействию блок «условная сумма» реализуется на обычной КМОП логике. Результат табл.5.1 показывает, что при одинаковом быстродействии средняя мощность снизилась в 2,5 раза.

Таблица.5.1. Сравнительные характеристики 64-х разрядных сумматоров после схемотехнической реализации (условия моделирования: $T=25^0$ С, $E_{пит}$=1,8 В; технология 0,18 мкм).

Архитектура	Входная емкость такт, пФ	Максимальная средняя мощность, мВт/ГГц	Средняя мощность, мВт/ГГц	Количество транзисторов	Время срабатывания, пс
KSR4S4	7,51	118	113	6366	405
LFR2S4	1,69	88	44	4513	396

Применение полудинамической схемотехники привело к потере динамической совместимости по выходу заказного блока, сигналы потеряли монотонность характерную для ДЛ. Статический характер выходного сигнала может привести к снижению быстродействия при использовании этого блока на более высоком уровне иерархии.

Рис.5.12. Структурная схема полутактового конвейера

На рис. 5.12 показана часто встречаемая структурная схема полутактового конвейера микропроцессора. Примером такой структуры может служить система выборки операндов между регистровым файлом, КЭШ памятью данных и АЛУ микропроцессора.

Для достижения максимального быстродействия весь критический путь конвейера (БЛОКN + ЛОГИКА) должен быть реализован на ДЛ. Но для этого все остальные некритичные узлы (БЛОК1, БЛОК2 и т.д.) должны быть динамически совместимы по выходу, иначе «ЛОГИКА» будет функционировать неправильно. Реализация всей схемы на ДЛ приведет к неоправданно резкому увеличению по-

требляемой мощности. С другой стороны, реализация узла «ЛОГИКА» на основе статической логики приведет к падению быстродействия конвейера.

Решить данную проблему можно за счет использования псевдодинамической логики (ПДЛ). С одной стороны, она дает возможность использовать динамически не совместимые блоки (реализованные на КМОП, КПЛ и т.д.), с другой стороны, сохраняет достигнутое быстродействие. Данный вид логики совместим как с динамически монотонными, так и со статическими сигналами, при этом время срабатывания логической ячейки на ПДЛ всего на 10...15% хуже ДЛ, что видно из табл.5.2.

Таблица.5.2. Сравнительный анализ различных типов логики в нормированном виде.

Тип ячейки	Логический стиль	Мощность (*P*) (нормир.)	Задержка (D) (нормир.)	*P*×*D* (нормир.)
Буфер	КМОП	1	1	1
	КПЛ	-	-	-
	Домино	5,2	0,43	2,24
	ПДомино	10.9	0,65	7,09
И	КМОП	1	1	1
	КПЛ	1,5	1,22	1,83
	Домино	5,6	0,57	3,19
	ПДомино	11,8	0,70	8,26
ИЛИ	КМОП	1	1	1
	КПЛ	1,5	0,96	1,44
	Домино	5,7	0,36	2,05
	ПДомино	11.8	0,52	6,14
Мультиплексор	КМОП	1	1	1
	КПЛ	0,45	0,57	0,26
	Домино	4,7	0,41	1,93
	ПДомино	10	0,51	5,1
Исключающее ИЛИ	КМОП	1	1	1
	КПЛ	0,46	0,56	0,26
	Домино	4,3	0,41	1,76
	ПДомино	9,2	0,50	4,6

Однако ПДЛ имеет существенный недостаток, который надо учитывать при утверждении технического задания. Средняя потребляемая мощность ячеек на ПДЛ с уменьшением частоты не изменяется. Это приводит к нерациональному расходу энергии при использовании данного конвейера на пониженных частотах.

В рассматриваемом случае, когда жертвуя малым, а именно повышенным потреблением блока «ЛОГИКА», реализованным на псевдодинамической логике, мы выигрываем в большом, в значительном уменьшении потребления конвейера за счет реализации некритичных блоков на статической и полудинамической логике, при этом сохраняя необходимое быстродействие в критическом пути.

Как было указано выше, существенного ускорения проектирования АЛУ и ряда других устройств можно достичь, разработав быстродействующий сумматор заказным образом с последующим синтезом всей схемы средствами САПР. Быстродействующий сумматор (БС) реализуется по схеме с ускоренным переносом (*carry look-ahead* (*CLA*)) семейства *Kogge-Stone,* на основе *Ling* уравнений. Общая структурная схема БС показана на рис.5.13.

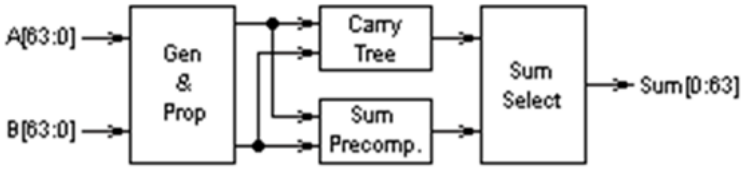


Рис. 5.13. Структурная схема быстродействующего сумматора

Первый каскад необходим для формирования опорных сигналов *Gen&Prop*, что дает возможность в параллель рассчитать сигналы переноса и предварительной суммы. Последний каскад формирует результат суммирования. В табл.5.3 представлены пять различных архитектур сумматоров, наиболее часто применяемых в современных микропроцессорах.

Radix-2 – формирование переноса за $2^n = K$, где n – количество каскадов в дереве переноса, K – количество разрядов сумматора.

Radix-4 – формирование переноса за $4^n=K$. При использовании R4 по сравнению с *R*2, сигнал переноса формируется за 3 сложных каскада, вместо 6-ти простых, отсюда вытекает ряд преимуществ:

- повышение быстродействия;
- значительно уменьшается число межсоединений, а, следовательно, уменьшается паразитная составляющая при реализации топологии;
- сокращение числа транзисторов на 25 %;
- уменьшение потребления;
- уменьшение нагрузки на тактовый сигнал.

Недостатком R4 является усиление эффекта деления заряда (*charge share*), что накладывает дополнительные требования при схемотехнической реализации для соблюдения характеристик помехоустойчивости.

Таблица.5.3. Сравнительные характеристики 64-х разрядных сумматоров после схемотехнической реализации на динамической логике (Условия моделирования: $T=25^0$ C, $E_{пит}$=1,8 В, технология 0,18 мкм).

Архитектура	Аббревиатура	Входная емкость, пФ	Средняя мощность, мВт/ГГц	Количество транзисторов	Время срабатывания, пс
Radix-2	R2	15,1	168	12394	417
Radix-2 Sparseness=4	R2S4	9,05	106	7570	423
Radix-4	R4	9,65	134	9142	402
Radix-4 Sparseness=2	R4S2	8,07	95	6814	399
Radix-4 Sparseness=4	R4S4	7,51	90	6366	405

Поскольку особенностью архитектуры сумматора на динамической логике является необходимость формирования дифференциального сигнала переноса, использование *R*2 или *R*4 является нецелесообразным. Необходимо перераспределить нагрузку блока переноса так, чтобы уравновесить её по критичности с блоком предварительной суммы, который строится на однофазной динамической логике.

Sparseness=2 или *Sparseness*=4 – формирование сигнала переноса в каждом 2-ом или 4-ом разряде, соответственно. Преимуществом использования *S*2 или *S*4 совместно с *Radix* является:

- уменьшение нагрузки на первый каскад сумматора;
- сокращение числа транзисторов на 20 %;
- уменьшение потребления;
- уменьшение нагрузки на тактовый сигнал.

К недостаткам *Radix-Sparseness* архитектуры можно отнести более высокие требования к выполнению топологии сумматора, поскольку сигналы, формирующие предварительную сумму сравнимы по критичности с сигналами переноса.

Из всех представленных в табл.5.3 архитектур, наиболее приемлемой для 64-х разрядных сумматоров является *R*4*S*4, так как топология сумматора с наименьшим количеством транзисторов, а, следовательно, и площадью, имеет минимальные паразитные *RC*-цепи. Таким образом, относительное увеличение времени срабатывания блока после физической реализации будет наименьшим. Это видно из табл. 5.4. Задержка сумматора с учетом экстракции паразитных параметров, выполненного по технологии 0,18 мкм объёмного кремния, увеличилась на 22 %. Также немаловажно отметить, что *R*4*S*4 имеет самое низкое потребление и минимальную ёмкость нагрузки локального дерева синхронизации.

Необходимо также уточнить результаты табл.5.4 и сделанных выводов для других применений. Архитектура *R*4*S*4 имеет преимущество только для сумматора с однофазным выходом (прямой или инверсный) работающим в 64-м режиме. Если, например, необходимо спроектировать сумматор с парафазным выходом, то характеристики табл.5.4 изменятся и оптимальной будет архитектура *R*4*S*2. А для 32-ого режима наилучшей будет архитектура *R*2*S*4.

Данные в табл. 5.4 демонстрируют сравнительный анализ нескольких работ последних лет. Самый быстродействующий из представленных сумматоров выполнен по технологии 0,18 мкм (*Cu*-КНИ) и имеет время срабатывания 440 пс, что на 12 % лучше, чем в работе 7. Однако очевидное преимущество КНИ, делает более выгодным использование сумматора 7.

Таблица 5.4. Сравнительные характеристики 64-х сумматоров

Дизайн	Статья	Епит, В	Технология	Кол-во тран-ов	Площадь, мкм × мкм	Время срабат., пс
[1]	2002 *JSSC*	2,5	0,25 мкм	-	800 × 150 (0,12 мм^2)	900
[2]	2001 *JSSC*	1,5	0,18 мкм (Cu-КНИ)	-	0,357 мм^2	440
[3]	2000 *VLSI Sym.*	1,8	0,225 мкм (*Cu*)	12846	190 × 806 (0,153 мм^2)	470
[4]	1999 *JSSC*	2,5	0,5 мкм	-	1600 × 275 (0,44 мм^2)	1500
[5]	1996 *ISSCC*	3,3	0,5 мкм	6924	96 × 2560 (0,246 мм^2)	930
[6]	1995 *CICC*	3,3	0,5 мкм	3044	1305 × 207 (0,27 мм^2)	2600
[7]	Электроника, микро- и наноэлектроника, 2006	1,8	0,18 мкм	6366	210 × 346 (0,073 мм^2)	490

В заключении необходимо отметить, что использование динамической логики является весьма трудоёмким и рискованным способом проектирования. Именно поэтому такое проектирование, как правило, допускается в редких, скорее даже исключительных, случаях. Прежде всего, такая логика используется для проектирования специальной памяти, такой как регистровые файлы и блоки трансляции виртуальных адресов в физические (*TLB*).

Заключение.

Создание высокопроизводительных микропроцессорных систем можно рассматривать как отдельную область науки в силу сложности задачи и необходимости решения многих задач в смежных областях. Невозможно создать действительно мощный микропроцессор без решения технологических, схемотехнических, конструктивных, инженерных, алгоритмических задач, создания программного обеспечения, встречной оптимизации ПО и аппаратуры.

Невозможно создать многопроцессорную систему без развития сетевых технологий, коммуникационных систем, новых сред передач высокочастотных сигналов. Создание высокопроизводительного вычислительного комплекса дополнительно требует решения ряда инженерных задач. Все эти направления должны быть поддержаны стандартами и соответствующими технологиями. В книге отражены только классификация, основные подходы по созданию и характеристики микропроцессоров и коммуникационных систем. В реальной работе для создания высокопроизводительных микропроцессорных систем приходится решать существенно большее разнообразие проблем. Это отвод тепла от микросхем и модулей, источники питания, кабельная продукция, среды передач, измерения, аттестация, программное обеспечение, алгоритмы, технология производства кристаллов и корпусов и многое другое. Ряд вопросов снимается при чётком следовании международным стандартам. Однако для создания вычислительных систем с уникальными характеристиками приходится искать новые пути развития практически по всем направлениям. Именно этим и интересна выделенная область науки.

Необходимость развития естественных наук, машиностроения, медицины, ядерной энергетики и др. областей требует перехода на микропроцессоры с терафлопсной и суперЭВМ с эксафлопсной производительностью. Для достижения таких характеристик созданные к настоящему моменту базовые принципы проектирования скорее всего придётся кардинально изменять. Главные проблемы – низкие надёжности и высокое энергопотребление микропроцессоров и суперЭВМ в целом. Проектирование суперЭВМ должно уже основываться на том, что сбои микросхем регулярны, и сама система должна поддерживать свою работоспособность, в отличие от старых систем, когда нештатные ситуации ложились на плечи обслуживающего персонала и программистов. По всей видимости, придётся переходить и на иную систему логики (возможно только для отдельных функциональных узлов). В книге рассмотрены, в частности, адиабатические схемы, обратимая и самосинхронная логики. Большой потенциал имеет самосинхронная логика при создании многоядерных микропроцессоров и выделении отдельных функционально законченных узлов. В этом случае удастся добиться основных преимуществ самосинхронной логики: 100% самопроверяемость, то

есть повышении надёжности микросхем и снижение потребления питания за счёт исключения клокового дерева, требующего до 30% потребляемой мощности высокопроизводительных синхронных микропроцессоров. Поднимется также производительность за счёт исключения необходимого в синхронных схемах времени предустановки (*setup*) и сохранения (*hold*) сигналов по отношению к клоковому сигналу. Однако этого можно достичь только при разбиении проекта на отдельные функционально законченные блоки. Наивысшие показатели производительности может дать потоковая архитектура процессоров на выделенных классах задач, таких как обработка изображений, задача горения и пр. Самосинхронная логика наиболее естественно реализует принципы потоковой машины и позволяет дать максимальное увеличение производительности. Создание гибридных микропроцессоров и/или процессорных модулей, содержащих универсальные микропроцессоры и потоковые процессоры, позволяет добиться гибкости программирования и высоких показателей производительности, зависящих от задачи.

При переходе на предельные нормы изготовления существенно усложняются маршруты и методики проектирования кристаллов, сужаются области применения заказного проектирования. В этом случае динамическая логика используется только в крайних случаях и, как правило, только для специальных структур памяти (например, регистровый файл). Необходимо уже в архитектуре микропроцессоров учитывать тот факт, что характеристики транзисторов разные в разных углах кристалла, то есть нельзя создавать блоки больших размеров (больше 1-3 мм).

Для предельных норм изготовления микросхем любая внешняя атака ионизирующих частиц способна приводить к сбою, поэтому конструкция СБИС должна обеспечивать минимизацию вероятности подобного сбоя и возможность исправления (вылечивания) произошедших ошибок. Соответственно и схемотехника, и топология, и конструкция, и маршрут проектирования суб-нм СБИС должны решать задачу не обороны, а исправления и излечения локальных потерь данных.

При проектировании суперЭВМ эксафлопсного класса необходимо учитывать то, что сбой — это не единичное событие, а событие

с вероятностью, приближающейся к 1, соответственно и проектирование аппаратно-программного комплекса должно быть организовано на постоянный контроль и исправление возникающих ошибок.

Эффективное решение ряда задач решается за счёт специализации и разделения труда как внутри компании, так и между отдельными компаниями. Реально позволить вести разработку по всем направлениям, необходимым для создания высокопроизводительных вычислительных систем, могут только такие компании как *Intel* и *IBM*. И главное преимущество этих компаний в наличии у них уникальных технологий производства микросхем. В частности, компания *IBM* планирует существенное повышение производительности многопроцессорных систем за счёт создания гибридных КМОП КНИ технологий микросхем и технологий оптических сред передачи (A_3B_5), то есть будут созданы микропроцессоры со встроенными оптическими излучателями и приёмниками, что позволит существенно сэкономить потребление энергии и поднять частоты передачи данных. Однако средние компании вполне могут составить конкуренцию большим компаниям, например, при выделении (специализации) областей применения, оптимизации под них аппаратного и программного обеспечения. Так что всё в Ваших руках.

Литература.

1. Tom R. Halfhill Netlogic Broadens XLP Family. Multithreading and Four-Way Issue with One to Eight CPU Cores. // Microprocessor Report. – 07.2010. – С.*1- 11.* *www.MPRonline.com*

2. www.caviumnetworks.com Multi-Core MIPS64 Processors OCTEON II CN63XX Multi-Core MIPS64 Processors.

3. *Patterson, David A., Hennessy, John L. Computer Architecture: A Quantitative Approach. Fourth edition. - The Morgan Kaufmann Series in Computer Architecture and Design. Kindle Locations 1789-1801. Elsevier Science (reference). Kindle Edition,* 2011.-676с.

4. Д. Адамов Учёт особенностей микроэлектронных нанотехнологий при проектировании СБИС. // Электроника: Наука, Технология, Бизнес. – 2007. - №7. – С.98-105. // Электроника: Наука, Технология, Бизнес. – 2007. - №8. – С.114-118.

5. Yeap G. Practical low power digital VLSI Design // Kluwer Academic Publishers, 1998.

6. *Abdollahi A., Fallah F., Pedram M. Leakage Current Reduction in CMOS VLSI Circuits by Input Vector Control // Proc. ISLPED*, 2002.

7. *Chandrakasan A., Sheng S., Broersen R. Low-power CMOS Digital Design // IEEE Journal of Solid-State Circuits, v.27(4)*, С.473-484.

8. Hoon-Sang Jin, Myung-Soo Jang, Jin-Suk Song, Jin-Yong Lee, Taek-Soo Kim, Jeong-Taek Kong Dynamic power estimation using the probabilistic contribution measure (PCM) // ISLPED '99: Proceedings of the 1999 international symposium on Low power electronics and design.

9. *Kougia S., Chatzigeorgiou A., Nikolaidis S. Evaluating Power Efficient Data-Reuse Decisions for Embedded Multimedia Applications: An Analytical Approach // Journal of Circuits, Systems and Computers, February*, 2004, *v.13, n.1*.

10. *Chatzigeorgiou A. and Stephanides G. Energy Issues in Software Design of Embedded Systems // 2nd WSEAS International Conference on Applied Informatics, Rethymnon, Crete, Greece, July* 7-14, 2002.

11. Борошко С.И., Иванов С.Г., Ивлев А.А., Ильягуев В.Н., Калашников О.А., Осипенко П.Н., Пекач Ю.В. Способы уменьшения токов потребления КМОП СБИС // Тезисы доклада на 7-й конференции «Радиационная стойкость электронных систем. Стойкость-2004», - М.: СПЭЛС, 2004, выпуск 7 С.77-78.

12. *Ishihara T. and Yasuura H. "Voltage scheduling problem for dynamically variable voltage processors" // Proc. Int'l Symposium on Low Power Electronics and Design, August, 1998*, С.197—202.

13. *T.Kuroda et al.. Variable supply-voltage scheme for low-power high-speed CMOS digital design // JSSC, vol.33*, С.*454-462, Mar.1998.*

14. Intel Centrino Mobile Technology Performance Brife // March, 2003. www.intel.com/performance

15. The Mobile AMD Athlon Processor // AMD, Inc, 2002, www.amd.com

16. *Transmeta Gets More Efficeon // Microprocessor Report, October*, 2003, v.17(10).

17. *Suessmith B., Paap III P. "PowerPC 603 microprocessor power management", Communications of the ACM, June, 1994, nr.6*, С.43—46.

18. *Gary S., Dietz C., Eno J., Gerosa G., Park S., and Sanchez H. The PowerPC TM 603 microprocessor: a low-power design for portable applications // Proceedings of the IEEE International Computer Society Conference (COMPCON 94), San Francisco, CA, February, 1994*, C.307—315.

19. *T. Kuroda Low power CMOS Design. A Design and System Perspective //IWFIPT, Keio University*, 2001.

20. *T.Kuroda. Low-power CMOS digital design with dual embedded adaptive power supplies //JSSC, vol.35, no.4*, C.652-655, *April* 2000.

21. J.*Kuo, S.Lin. Low-voltage SOI CMOS VLSI devces and Circuits //John Wiley & Sons, Inc, New York*, 2001.

22. Осипенко П.Н. Сравнительный анализ микропроцессорных схем, реализованных на КНИ и объёмном кремнии // V Всероссийское совещание-семинар по проблеме «Создание сверхбольших интегральных схем на основе структур «кремний-на-изоляторе» со скрытым диэлектрическим слоем» КНИ-2003, Секция прикладных проблем при Президиуме РАН, Москва, 2003.

23. *Tiwari V. et. al. "Reducing Power in High-performance Microprocessors" // Design Automation Conference, San Francisco, June*, 1998.

24. *Shelar R., Narayanan H., Desai M. Orthogonal Partitioning and Gated Clock Architecture for Low Power Realization of FSMs // IEEE Int. ASIC/SOC conf, September*, 2000, C.266-270

25. *Darren Jons. How to successfully use Gated Clocking in ASIC design // MIPS Technology Inc*, 2002.

26. R. Landauer, "Irreversibility and heat generation in the computing process," IBM J. Res. Develop., vol. 5, pp. 183–191, 1961.

27. *E. Fredkin and T. Toffoli, "Conservative logic", Int. J. of Theor. Phys., vol. 21*, C. 219-253, 1982.

28. *M. Haghparast and K. Navi.* A Novel Fault Tolerant Reversible Gate For Nanotechnology Based Systems // American Journal of Applied Sciences 5 (5): 519-523, 2008.

29. *A.I. Maimistov. Reversible logic elements as a new field of application of optical solutions // Quantum Electronics (1995). Volume: 25, Issue*: 10, C.1009-1013.

30. *N. Margolus. Feynman and Computation, chapter Crystalline Computation. Perseus Books*, 1999.

31. *T. Toffoli. Reversible computing. Tech memo MIT/LCS/TM-151, MIT Lab for Comp. Sci*, 1980.

32. P. *Kerntopf. A comparison of logical e±ciency of reversible and conventional gates. In International Workshop on Logic Synthesis*, C.261-269, 2000.

33. *A. Peres. Reversible logic and quantum computers. Physical Review A, 32:3266-3276*, 1985.

34. *N. Margolus. Feynman and Computation, chapter Crystalline Computation. Perseus Books*, 1999.

35. *A. De Vos. Towards reversible digital computers. In European Conference on Circuit Theory and Design, pages 923{931, Budapest, Hungary*, 1997.

36. *V. Shende, K. Prasad, L. Markov, P. Hayes / Synthesis of Reversible Logic Circuits // IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 22, NO. 6, JUNE* 2003, C. 710-722.

37. http://nthur.lib.nthu.edu.tw/bitstream/987654321/13644/1/2030238030013.pdf

38. *H. Thapliyal and M. B. Srinivas, "A beginning in the reversible logic synthesis of sequential circuits," in Proc. of MAPLD*, 2005.

39. *Bruce, J.W.; Thornton, M.A.; Shivakumaraiah, L.; Kokate, P.S. / Efficient adder circuits based on a conservative reversible logic gate // IEEE Computer Society Annual Symposium on VLSI*, 2002. *Proceedings*. C. 74-79.

40. http://ww1.ucmss.com/books/LFS/CSREA2006/CDE5277.pdf

41. *K. Kozo, S. Tsutomu, M. Jun. On Magnetic Bubble Logic Circuits // IEEE Transactions on Computer Society. Vol. C-25, Issue* 3, C. 247 – 253.

42. *S. Nakata. Recent Progress in Adiabatic Circuits // Recent Patents on Electrical Engineering* 2009, 2, C.40-44.

43. *Merkle, R. C. Reversible electronic logic using switches. // Nanotechnology, Volume 4, Issue 1*, C. 21-40 (1993).

44. http://lyle.smu.edu/~mitch/ftp_dir/pubs/feinstein_thesis.pdf

45. *B. Liu. Reconfigurable double gate carbon nanotube field effect transistor based nanoelectronic architecture // Asia and South Pacific Design Automation Conference*, 2009. *ASP-DAC* 2009. C.19-22.

46. *Varshavsky V., Kishinevsky M., Marakhovsky V. et al. Self-timed Control of Concurrent Processes, Ed. by V.Varshavsky. Kluver Academic Publishers*, 1990. – 245 с.

47. *Muller D.E. Asynchronous logics and application to information processing / In: H. Aiken, W.F. Main (Eds.), Switching Theory in Space Technology, Stanford University Press, Stanford, CA*, 1963 – С.289-297.

48. *Ad Peeters. The Asynchronous Bibliography. BIBTEX database file async.bib. May 05, 2004 /http://www.win.tue.nl/async-bib/doc/async.html/*

49. Плеханов Л.П. Разработка самосинхронных схем: функциональный подход // В сб. "Проблемы разработки перспективных микро- и наноэлектронных систем 2010", под общ. ред. А.Л. Стемпковского – М.: ИППМ РАН, 2010 – С. 424–429.

50. Плеханов Л.П. Основы самосинхронных электронных схем. – М.: БИНОМ. Лаборатория знаний, 2013 – 208 с.

51. Соколов И.А., Степченков Ю.А., Дьяченко Ю.Г. Самосинхронный триггер с однофазным информационным входом. Патент № 2405246 от 27.11.2010

52. *Sokolov I.A., Stepchenkov Y.A., Dyachenko Y.G. Self-timed RS-trigger with the enhanced noise immunity. Inter. Application number PCT/RU*2010/000279.

53. Велихов Е.П., Бетелин В.Б., Кушниренко А.Г. Промышленность, инновации, образование и наука в России. - М.: Наука, 2009. -141 с.

54. Rick Stevens and Andy White. A DOE Laboratory plan for providing exascale applications and technologies for critical DOE mission needs http://computing.ornl.gov/workshops/SCIDAC2010/r_stevens.pdf

55. International Exascale Software Project www.exascale.org

56. *The Defense Advanced Research Projects Agency DARPA* [Электронный ресурс]. Режим доступа: www.darpa.mil

57. PRACE www.prace-project.eu

58. Бетелин В.Б., Баранов С.В., Бобков С.Г., Краснюк А.А., Осипенко П.Н., Стенин В.Я., Черкасов И.Г., Чумаков А.И., Яненко А.В. Перспективы использования субмикронных КМОП СБИС в сбоеустойчивой аппаратуре, работающей под воздействием атмосферных нейтронов. //Микроэлектроника, 2009, том 38, №1, с.1-5.

59. Бобков С. Г., Сидоров Е.А. Коммуникационные технологии в высокопроизводительных вычислительных системах. // Средства разработки высокопроизводительных вычислительных систем. Коммуникационные технологии. Системы памяти. – Сб. статей под ред. чл.-корр. РАН В.Б. Бетелина. НИИСИ РАН. Москва, 2001 г. – с. 4-63

60. Корнеев В. В. Параллельные вычислительные системы.- М.: Нолидж,1999. - 320 с.

61. Андреев А., Воеводин В., Жуматин С. Кластеры и суперкомпьютеры - близнецы или братья? Открытые системы No.5-6, 2000.

62. *HP 9000 V2600 Enterprise Server. Architectural Overview White Paper*.

63. Шадский А. Семейство компьютеров Ultra компании Sun Microsystems. JetInfo. No. 23-24,1997

64. Кузьминский М. Архитектура S2MP - свежий взгляд на cc-NUMA. Открытые ситемы No. 2, 1997.

65. Кузьминский М. Векторно - параллельные суперкомпьютеры NEC. Открытые системы No. 3, 1999

66. *S. Garg, R. Godley, R. Griffiths, A. Pfiffer, T. Prickett, D. Robboy, S. Smith, T.M. Stallcup, S. Zeisset. Achiving Large Scale Parallelism Through Operating System Resourse Management on the Intel TFLOPS Supercomputer. Intel Technology Journal, Quarter 1*, 1998

67. QSW SuperCluster Architecture Overview. Quadrics Supercomputer Worlds Ltd. (http://www.quadrics.com)

68. *A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, V. Sunderam. PVM - Parallel Virtual Machine: A User Guide and Tutorial for Networked Parallel Computing. The MIT Press. Cambridge Massachusets. London, England*. 1994

69. *Message Passing Interface Forum. MPI: A message-passing interface standard*. 1995

70. *OpenMP: A Proposed Industry Standard API for Shared Memory Programming. October* 1997

71. *QSW SuperCluster Architecture Overview. Quadrics Supercomputer Worlds Ltd*.

72. Робачевский А.М. Операционная система UNIX. - СПб.: БХВ Санкт-Петербург, 1999

73. *C. Schimmel. UNIX Systems for Modern Architectures: Symmetric Multiprocessing and Caching for Kernel Programmers. Addison-Wesley Professional Computing Series*, 1994

74. *Virtual Interface Architecture Specification. Version 1.0. Compaq Computer Corp., Intel Corporation, Microsoft Corporation.December* 16, 1997

75. Скотт М. Ядро Linux в комментариях: Пер. с англ./Скотт Максвелл.- К.: Издательство Диасофт, 2000

76. *Intel Virtual Interface (VI) Architecture Developer's Guide. Revision 1.0. Intel Corporation. September* 9, 1998

77. *cLAN for Windows. Software User's Guide. PartNumber: CLAN-D004-001, Revision 4.1. Giganet Incorporated. August*, 2000

78. *ServerNet - A High Bandwidth, Low Latency Cluster Interconnection. Compaq White Paper ECG097/0998. September*, 1998

79. *The GM Message Passing System. Myricom, Inc*. 1999

80. *W. Gropp, E. Lusk, N. Doss, A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard*.

81. *W. Gropp, E. Lusk. MPICH Abstract Device Interface. Version 3. Reference Manual, Mathematics and Computer Science Division. Argonne National Laboratory. Draft of November 16*, 2000

82. *InfiniBand Cavalry to the Rescue. By Infiniband Trade Assiciation. Buses and Boards, September/October* 2000.

83. *Myrinet-on-VME Protocol Specification Draft Standart. VITA 26-199x. Draft 1.1, 31 August* 1998.

84. *RapidIO specification Revision 2.1, Complete Spec. Stack, 2009* (*www.rapidio.org/specs/current*).

85. Олифер В.Г., Олифер Н.С. Компьютерные сети. Принципы, технологии, протоколы. – СПб: Изд. дом "Питер", 2000. - 672с.

86. Новиков Ю. В., Карпенко Д. Г. Аппаратура локальных сетей: функции, выбор, разработка / Под общей редакцией Ю.В. Новикова. – М.: ЭКОМ, 1998. – 288с.

87. Басиладзе С.Г. Интерфейсы магистрально-модульных многопроцессорных систем - М.: Энергоатомиздат, 1992. - 256 с.

88. *USB in a NutShell // http://www.beyondlogic.org/usbnutshell/usb1.shtml, - December 10th*, 2011

89. *Sam Fuller RapidIO. The Embedded System Interconnect // John Wiley & Sons, Ltd* / - 2005. – 382 с.

90. Евтушенко Н., Немудров В., Сырцов И. Методология проектирования систем на кристалле, основные принципы, методы, программные средства // Электроника. – 2003. - № 6, С.12-16.

91. *Galaxy Design Platform // www.synopsys.com/products/solutions/galaxy_platform.html*, - 2006.

92. *INCISIVE FUNCTIONAL VERIFICATION PLATFORM, Data Sheet, // www.cadence.com/brochures/incisive_platform.pdf*, 2005.

93. Бобков С.Г. Методика проектирования микросхем для компьютеров серии «Багет» // Информационные технологии, № 3, 2008, с.2-7.

94. *Introduction to the Cell multiprocessor/ J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, D. Shippy // IBM J. RES. & DEV. – 2005. - VOL. 49. - NO.* 4/5. – С.21-34.

95. Бобков С.Г. Маршрут проектирования микросхем серий 1890 и 1990. Часть 1 // Электронные компоненты. – 2008. - №5. - С.98-104.

Бобков С.Г. Маршрут проектирования микросхем серий 1890 и 1990.Часть 2 // Электронные компоненты. – 2008. - №8. - С.126-131.

96. Шагурин И., Шалтырев В, Волов А. «Большие» FPGA как элементная база для реализации систем на кристалле // Электронные компоненты. – 2008. - №1. - С. 38-42.

97. Бобков С.Г., Евлампиев Б.Е. Схема тестирования высокочастотных блоков однокристального графического контроллера. Научная сессия МИФИ-2003 // Сборник научных трудов в 14 томах. Том 1. Автоматика. Электроника. Электронные измерительные системы. М.: МИФИ, 2003. - С.173-174.

98. Бобков С.Г., Евлампиев Б.Е. Разработка методик проектирования быстродействующих СБИС со сложной структурой класса микросхем графического контроллера // Информационные технологии и вычислительные системы. – 2005. - №1. - С.88-104.

99. Бобков С.Г. Методика тестирования микросхем для компьютеров серии «Багет». // Программные продукты и системы. – 2007. - №3. – С.2-5.

100. Бобков С.Г. Евлампиев Б.Е., Сидоров А.Ю. Блок самотестирования внутренней памяти // Проблемы разработки перспективных

микроэлектронных систем. Сб. научн. трудов под общ. ред. А.Л. Стемпковского. – М.: ИППМ РАН, 2005. - С.222-228.

101. *Advanced microprocessor test strategy and methodology/ Huott, W V, Koprowski, T J, Robbins, B J, Kusko, M P, Et al. // IBM Journal of Research and Development. – 1997. - Volume 41. - Number* 4/5. – С.1-22.

102. *Full Hold-Scan Systems in Microprocessors: Cost/Benefit Analysis/ Kuppuswamy R., DesRosier P., Feltham D., Sheikh R., Thadikaran P. // Intel Technology Journal. – 2004. - Volume 08. - Issue* 01. – С.63-72.

103. Бобков С.Г., Грузинова Е.В., Сидоров А.Ю. Применение интерфейса JTAG при тестировании системного контроллера // Электроника, микро- и наноэлектроника. Сб. научн. трудов под ред. В.Я. Стенина. – М.: МИФИ, 2003. - С.198-203.

104. Бобков С.Г., Евлампиев Б.Е. Схема тестирования высокочастотных блоков однокристального графического контроллера. Научная сессия МИФИ-2003 // Сборник научных трудов в 14 томах. Том 1. Автоматика. Электроника. Электронные измерительные системы. М.: МИФИ, 2003. - С.173-174.

105. *High-Performance Microprocessor Design/ P.E. Gronowski, W.J. Bowhill, R.P. Preston, M.K. Gowan, R.L. Allmon // IEEE JOURNAL OF SOLID-STATE CIRCUITS.* – 1998. - *VOL. 33. - NO.* 5. – С. 676–685.

106. Агафонов А.Е., Бобков С.Г., Токарев В.Е. Структура «Домино» конвейера // Электроника, микро- и наноэлектроника. Сб. научн. трудов под ред. В.Я. Стенина. – М.: МИФИ, 2006. - С.102-109.

107. Бобков С.Г., Ларионов А.В. Применение псевдодинамической логики в микропроцессорной технике // Электроника, микро- и наноэлектроника. Сб. научн. трудов под ред. В.Я. Стенина. – М.: МИФИ, 2007. - С.157-160.

108. Агафонов А.Е., Бобков С.Г., Токарев В.Е. 64-разрядное однотактовое целочисленной 400 МГц АЛУ // Электроника, микро- и наноэлектроника. Сб. научн. трудов под ред. В.Я. Стенина. – М.: МИФИ, 2007. - С.167-170.

109. Бобков С.Г., Токарев В.Е. Проектирование заказных блоков с учетом экстракции RC паразитных параметров // Электроника,

микро- и наноэлектроника. Сб. научн. трудов под ред. В.Я. Стенина. – М.: МИФИ, 2005. - С.125-132.

110. Бобков С.Г., Токарев В.Е. Особенности проведения входного контроля кристаллов перед изготовлением // Научная сессия МИФИ-2004. Сб. науч. трудов. В 15т. М.: МИФИ, 2004. - Т. 1. - С. 189-190.

111. Бобков С.Г., Токарев В.Е. Проектирование заказных блоков с учетом экстракции RC паразитных параметров // Проблемы разработки перспективных микроэлектронных систем. Сб. научн. трудов под общ. ред. А.Л. Стемпковского. – М.: ИППМ РАН, 2005. - С.51-52.

112. Бобков С.Г., Аряшев С.И., Барских М.Е., Бычков К.С., Зубковский П.С. Микропроцессор гибридный. Патент № 2359315 от 20.06.2009.

113. David Harris, Mark A. Horowitz. Skew-Tolerant Domino Circuits // IEEE Journal of SSC. - 1997. - Vol. 32. - № 11.- PP.1702-1711.

114. Бобков С.Г., Ларионов А.В., Токарев В.Е. Сравнительный анализ архитектур быстродействующих сумматоров // Электроника, микро- и наноэлектроника. Сб. научн. трудов под ред. В.Я. Стенина. – М.: МИФИ, 2006. - С.117-120.