Глава 1

Становление и эволюция цифровой вычислительной техники

Эволюция средств автоматизации вычислений

Попытки облегчить, а в идеале автоматизировать процесс вычислений имеют давнюю историю, насчитывающую более 5000 лет. С развитием науки и технологий средства автоматизации вычислений непрерывно совершенствовались. Современное состояние вычислительной техники (ВТ) являет собой результат многолетней эволюции.

В традиционной трактовке эволюцию вычислительной техники представляют как последовательную смену поколений ВТ. Появление термина «поколение» относится к 1964 году, когда фирма IBM выпустила серию компьютеров IBM 360, назвав эту серию «компьютерами третьего поколения». В стандарте ГОСТ 15971-90 дано следующее определение термина:

«Поколение вычислительных машин — это классификационная группа ВМ, объединяющая ВМ по используемой технологии реализации ее устройств, а также по уровню развития функциональных свойств и программного обеспечения и характеризующая определенный период в развитии промышленности средств вычислительной техники».

Одной из идей развития функциональных свойств принято считать концепцию вычислительной машины с хранимой в памяти программой, сформулированную Джоном фон Нейманом. Взяв ее за точку отсчета, историю развития ВТ можно представить в виде трех этапов:

- донеймановского периода;
- эры вычислительных машин и систем с фон-неймановской архитектурой;
- постнеймановской эпохи — эпохи параллельных и распределенных вычислений, где наряду с традиционным подходом все большую роль начинают играть отличные от фон-неймановских принципы организации вычислительного процесса.

Значительно большее распространение, однако, получила привязка поколений к смене технологий. Принято говорить о «механической» эре (нулевое поколение) и последовавших за ней пяти поколениях ВС [28, 152]. Первые четыре поколения традиционно связывают с элементной базой вычислительных систем: электронные лампы, полупроводниковые приборы, интегральные схемы малой степени интеграции (ИМС), большие (БИС), сверхбольшие (СБИС) и ультрабольшие (УБИС)

интегральные микросхемы. Пятое поколение в общепринятой интерпретации ассоциируют не столько с новой элементной базой, сколько с интеллектуальными возможностями ВС. Работы по созданию ВС пятого поколения велись в рамках четырех достаточно независимых программ, осуществлявшихся учеными США, Японии, стран Западной Европы и стран Совета экономической взаимопомощи. Ввиду того, что ни одна из программ не привела к ожидаемым результатам, разговоры о ВС пятого поколения понемногу утихают. Трактовка пятого поколения явно выпадает из «технологического» принципа. С другой стороны, причисление всех ВС на базе сверхбольших интегральных схем (СБИС) к четвертому поколению не отражает принципиальных изменений в архитектуре ВС, произошедших за последние годы. Чтобы в какой-то мере проследить роль таких изменений, воспользуемся несколько отличной трактовкой, предлагаемой в [125]. В работе выделяется шесть поколений ВС. Попытаемся кратко охарактеризовать каждое из них, выделяя наиболее значимые события.

Нулевое поколение (1492–1945)

Для полноты картины упомянем два события, произошедшие до нашей эры: первые счеты — абак, изобретенные в древнем Вавилоне за 3000 лет до н. э., и их более «современный» вариант с косточками на проволоке, появившийся в Китае примерно за 500 лет также до н. э.

«Механическая» эра (нулевое поколение) в эволюции ВТ связана с механическими, а позже — электромеханическими вычислительными устройствами. Основным элементом механических устройств было зубчатое колесо. Начиная с XX века роль базового элемента переходит к электромеханическому реле. Не умаляя значения многих идей «механической» эры, необходимо отметить, что ни одно из созданных устройств нельзя с полным основанием назвать вычислительной машиной в современном ее понимании. Чтобы подчеркнуть это, вместо термина «вычислительная машина» будем использовать такие слова, как «вычислитель», «калькулятор» и т. п.

Хронология основных событий «механической» эры выглядит следующим образом.

1492 год. В одном из своих дневников Леонардо да Винчи приводит рисунок тринадцатиразрядного десятичного суммирующего устройства на основе зубчатых колес.

1623 год. Вильгельм Шиккард (Wilhelm Schickard, 1592–1635), профессор университета Тюбингена, разрабатывает устройство на основе зубчатых колес («считающие часы») для сложения и вычитания шестиразрядных десятичных чисел. Было ли устройство реализовано при жизни изобретателя, достоверно неизвестно, но в1960 году оно было воссоздано и проявило себя вполне работоспособным.

1642 год. Блез Паскаль (Blaise Pascal, 1623–1663) представляет «Паскалин» — первое реально осуществленное и получившее известность механическое цифровое вычислительное устройство. Прототип устройства суммировал и вычитал пятиразрядные десятичные числа. Паскаль изготовил более десяти таких вычислителей, причем последние модели оперировали числами длиной в восемь цифр.

1673 год. Готфрид Вильгельм Лейбниц (Gottfried Wilhelm Leibniz, 1646–1716) создает «пошаговый вычислитель» — десятичное устройство для выполнения всех четырех арифметических операций над 12-разрядными десятичными числами. Результат умножения представлялся 16 цифрами. Помимо зубчатых колес, в устройстве использовался новый элемент — ступенчатый валик.

1786 год. Немецкий военный инженер Иоганн Мюллер (Johann Mueller, 1746–1830) выдвигает идею «разностной машины» — специализированного калькулятора для табулирования логарифмов, вычисляемых разностным методом. Калькулятор, построенный на ступенчатых валиках Лейбница, получился достаточно небольшим (13 см в высоту и 30 см в диаметре), но при этом мог выполнять все четыре арифметических действия над 14-разрядными числами.

1801 год. Жозеф Мария Жаккард (Joseph-Marie Jacquard, 1752–1834) строит ткацкий станок с программным управлением, программа работы которого задается с помощью комплекта перфокарт.

1832 год. Английский математик Чарльз Бэббидж (Charles Babbage, 1792–1871) создает сегмент разностной машины, оперирующий шестиразрядными числами и разностями второго порядка. Разностная машина Бэббиджа по идее аналогична калькулятору Мюллера.

1834 год. Пер Георг Шутц (Per George Scheutz, 1785–1873) из Стокгольма, используя краткое описание проекта Бэббиджа, создает из дерева небольшую разностную машину.

1836 год. Бэббидж разрабатывает проект «аналитической машины». Проект предусматривает три считывателя с перфокарт для ввода программ и данных, память (по Бэббиджу — «склад») на пятьдесят 40-разрядных чисел, два аккумулятора для хранения промежуточных результатов. В программировании машины предусмотрена концепция условного перехода. В проект заложен также и прообраз микропрограммирования — содержание инструкций предполагалось задавать путем позиционирования металлических штырей в цилиндре с отверстиями. По оценкам автора, суммирование должно было занимать 3 с, а умножение и деление — 2–4 мин.

1843 год. Георг Шутц совместно с сыном Эдвардом (Edvard Scheutz, 1821–1881) строят разностную машину с принтером для работы с разностями третьего порядка.

1871 год. Бэббидж создает прототип одного из устройств своей аналитической машины — «мельницу» (так он окрестил то, что сейчас принято называть центральным процессором), а также принтер.

1885 год. Дорр Фельт (Dorr E. Felt, 1862–1930) из Чикаго строит свой «комптометр» — первый калькулятор, где числа вводятся нажатием клавиш.

1890 год. Результаты переписи населения в США обрабатываются с помощью перфокарточного табулятора, созданного Германом Холлеритом (Herman Hollerith, 1860–1929) из Массачусетского технологического института.

1892 год. Вильям Барроуз (William S. Burroughs, 1857–1898) предлагает устройство, схожее с калькулятором Фельта, но более надежное, и от этого события берет старт индустрия офисных калькуляторов.

1937 год. Джорж Стибитц (George Stibitz, 1904–1995) из Bell Telephone Laboratories демонстрирует первый однобитовый двоичный вычислитель на базе электромеханических реле.

1937 год. Алан Тьюринг (Alan M. Turing, 1912–1954) из Кембриджского университета публикует статью, в которой излагает концепцию теоретической упрощенной вычислительной машины, в дальнейшем получившей название машины Тьюринга.

1938 год. Клод Шеннон (Claude E. Shannon, 1916–2001) публикует статью о реализации символической логики на базе реле.

1938 год. Немецкий инженер Конрад Цузе (Konrad Zuse, 1910–1995) строит механический программируемый вычислитель Z1 с памятью на 1000 битов. В последнее время Z1 все чаще называют первым в мире компьютером.

1939 год. Джордж Стибитц и Сэмюэль Вильямс (Samuel Williams, 1911–1977) представили Model I — калькулятор на базе релейной логики, управляемый с помощью модифицированного телетайпа, что позволило подключаться к калькулятору по телефонной линии. Более поздние модификации допускали также определенную степень программирования.

1940 год. Следующая работа Цузе — электромеханическая машина Z2, основу которой составляла релейная логика, хотя память, как и в Z1, была механической.

1941 год. Цузе создает электромеханический программируемый вычислитель Z3. Вычислитель содержит 2600 электромеханических реле. Z3 — это первая попытка реализации принципа программного управления, хотя и не в полном объеме (в общепринятом понимании этот принцип еще не был сформулирован). В частности, не предусматривалась возможность условного перехода. Программа хранилась на перфоленте. Емкость[1] памяти составляла 64 22-битовых слова. Операция умножения занимала 3–5 с.

1943 год. Группа ученых Гарвардского университета во главе с Говардом Айкеном (Howard Aiken, 1900–1973) разрабатывает вычислитель ASCC Mark I (Automatic Sequence-Controlled Calculator Mark I) — первый программно управляемый вычислитель, получивший широкую известность. Длина устройства составила 18 м, а весило оно 5 т. Машина состояла из множества вычислителей, обрабатывающих свои части общей задачи под управлением единого устройства управления. Команды считывались с бумажной перфоленты и выполнялись в порядке считывания. Данные считывались с перфокарт. Вычислитель обрабатывал 23-разрядные числа, при этом сложение занимало 0,3 с, умножение — 4 с, а деление — 10 с.

1945 год. Цузе завершает Z4 — улучшенную версию вычислителя Z3. По архитектуре у Z4 очень много общих черт с современными ВМ: память и процессор

[1] Емкость характеризует количество элементов данных, которое одновременно может храниться в памяти.

представлены отдельными устройствами, процессор может обрабатывать числа с плавающей запятой и, в дополнение к четырем основным арифметическим операциям, способен извлекать квадратный корень. Программа хранится на перфоленте и считывается последовательно.

Не умаляя важности каждого из перечисленных фактов, в качестве важнейшего момента «механической» эпохи все-таки выделим аналитическую машину Чарльза Бэббиджа и связанные с ней идеи.

Первое поколение (1937–1953)

На роль первой в истории электронной вычислительной машины в разные периоды претендовало несколько разработок. Общим у них было использование схем на базе электронно-вакуумных ламп вместо электромеханических реле. Предполагалось, что электронные ключи будут значительно надежнее, поскольку в них отсутствуют движущиеся части, однако технология того времени была настолько несовершенной, что по надежности электронные лампы оказались ненамного лучше, чем реле. Однако у электронных компонентов имелось одно важное преимущество: выполненные на них ключи могли переключаться примерно в тысячу раз быстрее своих электромеханических аналогов.

Первой электронной вычислительной машиной чаще всего называют специализированный калькулятор ABC (Atanasoff–Berry Computer). Разработан он был в период с 1939 по 1942 год профессором Джоном Атанасовым (John V. Atanasoff, 1903–1995) совместно с аспирантом Клиффордом Берри (Clifford Berry, 1918–1963) и предназначался для решения системы линейных уравнений (до 29 уравнений с 29 переменными). ABC обладал памятью на 50 слов длиной 50 битов, а запоминающими элементами служили конденсаторы с цепями регенерации. В качестве вторичной памяти использовались перфокарты, где отверстия не перфорировались, а прожигались. ABC стал считаться первой электронной ВМ, после того как судебным решением были аннулированы патенты создателей другого электронного калькулятора — ENIAC. Необходимо все же отметить, что ни ABC, ни ENIAC не являются вычислительными машинами в современном понимании этого термина и их правильней классифицировать как калькуляторы.

Вторым претендентом на первенство считается вычислитель Colossus, построенный в 1943 году в Англии в местечке Bletchley Park близ Кембриджа. Изобретателем машины был профессор Макс Ньюмен (Max Newman, 1987–1984), а изготовил ее Томми Флауэрс (Tommy Flowers, 1905–1998). Colossus был создан для расшифровки кодов немецкой шифровальной машины «Лоренц Шлюссель-цузат-40». В состав команды разработчиков входил также Алан Тьюринг. Машина была выполнена в виде восьми стоек высотой 2,3 м, а общая длина ее составляла 5,5 м. В логических схемах машины и в системе оптического считывания информации использовалось 2400 электронных ламп, главным образом тиратронов. Информация считывалась с пяти вращающихся длинных бумажных колец со скоростью 5000 символов/с.

Наконец, третий кандидат на роль первой электронной ВМ — уже упоминавшийся программируемый электронный калькулятор общего назначения ENIAC

(Electronic Numerical Integrator and Computer — электронный цифровой интегратор и вычислитель). Идея калькулятора, выдвинутая в 1942 году Джоном Мочли (John J. Mauchly, 1907–1980) из университета Пенсильвании, была реализована им совместно с Преспером Эккертом (J. Presper Eckert, 1919–1995) в 1946 году. С самого начала ENIAC активно использовался в программе разработки водородной бомбы. Машина эксплуатировалась до 1955 года и применялась для генерирования случайных чисел, предсказания погоды и проектирования аэродинамических труб. ENIAC весил 30 тонн, содержал 18 000 радиоламп, имел размеры 2,5 × 30 м и обеспечивал выполнение 5000 сложений и 360 умножений в секунду. Использовалась десятичная система счисления. Программа задавалась схемой коммутации триггеров на 40 наборных полях. Когда все лампы работали, инженерный персонал мог настроить ENIAC на новую задачу, вручную изменив подключение 6000 проводов. При пробной эксплуатации выяснилось, что надежность машины чрезвычайно низка — поиск неисправностей занимал от нескольких часов до нескольких суток. По своей структуре ENIAC напоминал механические вычислительные машины. Десять триггеров соединялись в кольцо, образуя десятичный счетчик, который исполнял роль счетного колеса механической машины. Десять таких колец плюс два триггера для представления знака числа представляли запоминающий регистр. Всего в ENIAC было 20 таких регистров. Система переноса десятков в накопителях была аналогична предварительному переносу в машине Бэббиджа.

При всей важности каждой из трех рассмотренных разработок основное событие, произошедшее в этот период, связано с именем Джона фон Неймана. Американский математик Джон фон Нейман (John von Neumann, 1903–1957) принял участие в проекте ENIAC в качестве консультанта. Еще до завершения ENIAC Эккерт, Мочли и фон Нейман приступили к новому проекту — EDVAC, главной особенностью которого стала идея *хранимой в памяти программы*.

Технология программирования в рассматриваемый период была еще на зачаточном уровне. Первые программы составлялись в машинных кодах — числах, непосредственно записываемых в память ВМ. Лишь в 50-х годах началось использование языка ассемблера, позволявшего вместо числовой записи команд использовать символьную их нотацию, после чего специальной программой, также называемой ассемблером, эти символьные обозначения транслировались в соответствующие коды.

Несмотря на свою примитивность, машины первого поколения оказались весьма полезными для инженерных целей и в прикладных науках. Так, Атанасофф подсчитал, что решение системы из восьми уравнений с восемью переменными с помощью популярного тогда электромеханического калькулятора Маршана заняло бы восемь часов. В случае же 29 уравнений с 29 переменными, с которыми калькулятор ABC справлялся менее чем за час, устройство с калькулятором Маршана затратило бы 381 час. С первой задачей в рамках проекта водородной бомбы ENIAC справился за 20 с, в противовес 40 часам, которые понадобились бы при использовании механических калькуляторов.

В 1947 году под руководством С. А. Лебедева начаты работы по созданию малой электронной счетной машины (МЭСМ). Эта ВМ была запущена в эксплуатацию в 1951 году и стала первой электронной ВМ в СССР и континентальной Европе.

В 1952 году Эккерт и Мочли создали первую коммерчески успешную машину UNIVAC. Именно с помощью этой ВМ было предсказано, что Эйзенхауэр в результате президентских выборов победит Стивенсона с разрывом в 438 голосов (фактический разрыв составил 442 голоса).

Также в 1952 году в опытную эксплуатацию была запущена вычислительная машина М-1 (И. С. Брук, Н. Я. Матюхин, А. Б. Залкинд). М-1 содержала 730 электронных ламп, оперативную память емкостью 256 25-разрядных слов, рулонный телетайп и обладала производительностью 15–20 операций/с. Впервые была применена двухадресная система команд. Чуть позже группой выпускников МЭИ под руководством И. С. Брука создана машина М-2 с емкостью оперативной памяти 512 34-разрядных слов и быстродействием 2000 операций/с.

В апреле 1953 года в эксплуатацию поступила самая быстродействующая в Европе ВМ БЭСМ (С. А. Лебедев). Ее быстродействие составило 8000–10 000 операций/с. Примерно в то же время выпущена ламповая ВМ «Стрела» (Ю. А. Базилевский, Б. И. Рамееев) с быстродействием 2000 операций/с.

Второе поколение (1954–1962)

Второе поколение характеризуется рядом достижений в элементной базе, структуре и программном обеспечении. Принято считать, что поводом для выделения нового поколения ВМ стали технологические изменения и, главным образом, переход от электронных ламп к полупроводниковым диодам и транзисторам со временем переключения порядка 0,3 мс.

Первой ВМ, выполненной полностью на полупроводниковых диодах и транзисторах, стала TRADIC (TRAnisitor DIgital Computer), построенная в Bell Labs по заказу военно-воздушных сил США как прототип бортовой ВМ. Машина состояла из 700 транзисторов и 10 000 германиевых диодов. За два года эксплуатации TRADIC отказали только 17 полупроводниковых элементов, что говорит о прорыве в области надежности, по сравнению с машинами на электронных лампах. Другой достойной упоминания полностью полупроводниковой ВМ стала TX-0, созданная в 1957 году в Массачусетском технологическом институте.

Со вторым поколением ВМ ассоциируют еще одно принципиальное технологическое усовершенствование — переход от устройств памяти на базе ртутных линий задержки к устройствам на магнитных сердечниках. В запоминающих устройствах (ЗУ) на линиях задержки данные хранились в виде акустической волны, непрерывно циркулирующей по кольцу из линий задержки, а доступ к элементу данных становился возможным лишь в момент прохождения соответствующего участка волны вблизи устройства считывания/записи. Главным преимуществом ЗУ на магнитных сердечниках стал произвольный доступ к данным, когда в любой момент доступен любой элемент данных, причем время доступа не зависит от того, какой это элемент.

Технологический прогресс дополняют важные изменения в архитектуре ВМ. Прежде всего это касается появления в составе процессора ВМ индексных регистров, что позволило упростить доступ к элементам массивов. Раньше, при циклической обработке элементов массива, необходимо было модифицировать код команды, в частности хранящийся в нем адрес элемента массива. Как следствие, в ходе вычислений коды некоторых команд постоянно изменялись, что затрудняло отладку программы. С использованием индексных регистров адрес элемента массива вычисляется как сумма адресной части команды и содержимого индексного регистра. Это позволяет обратиться к любому элементу массива, не затрагивая код команды, а лишь модифицируя содержимое индексного регистра.

Вторым принципиальным изменением в структуре ВМ стало добавление аппаратного блока обработки чисел в формате с плавающей запятой. До этого обработка вещественных чисел производилась с помощью подпрограмм, каждая из которых имитировала выполнение какой-то одной операции с плавающей запятой (сложение, умножение и т. п.), используя для этой цели обычное целочисленное арифметико-логическое устройство.

Третье значимое нововведение в архитектуре ВМ — появление в составе вычислительной машины процессоров ввода/вывода, позволяющих освободить центральный процессор от рутинных операций по управлению вводом/выводом и обеспечивающих более высокую пропускную способность тракта «память — устройства ввода/вывода» (УВВ).

Ко второму поколению относятся и две первые суперЭВМ, разработанные для ускорения численных вычислений в научных приложениях. Термин «суперЭВМ» первоначально применялся по отношению к ВМ, производительность которых на один или более порядков превосходила таковую для прочих вычислительных машин того же поколения. Во втором поколении этому определению отвечали две ВМ (правильнее сказать системы): LARC (Livermore Atomic Research Computer) и IBM 7030. Помимо прочего, в этих ВМ нашли воплощение еще две новинки: совмещение операций процессора с обращением к памяти и простейшие формы параллельной обработки данных.

Заметным событием данного периода стало появление в 1958 году машины М-20. В этой ВМ, в частности, были реализованы: частичное совмещение операций, аппаратные средства поддержки программных циклов, возможность параллельной работы процессора и устройства вывода. Оперативная память емкостью 4096 45-разрядных слов была выполнена на магнитных сердечниках.

Шестидесятые годы XX века стали периодом бурного развития вычислительной техники в СССР. За этот период разработаны и запущены в производство вычислительные машины «Урал-1», «Урал-4», «Урал-11», «Урал-14», БЭСМ-2, М-40, «Минск-1», «Минск-2», «Минск-22», «Минск-32». В 1960 году под руководством В. М. Глушкова и Б. Н. Малиновского разработана первая полупроводниковая управляющая машина «Днепр».

Наконец, нельзя не отметить значительные события в сфере программного обеспечения, а именно создание языков программирования высокого уровня: Фортрана (1956), Алгола (1958) и Кобола (1959).

Третье поколение (1963–1972)

Третье поколение ознаменовалось резким увеличением вычислительной мощности ВМ, ставшим следствием больших успехов в области архитектуры, технологии и программного обеспечения. Основные технологические достижения связаны с переходом от дискретных полупроводниковых элементов к интегральным микросхемам и началом применения полупроводниковых запоминающих устройств, начинающих вытеснять ЗУ на магнитных сердечниках. Существенные изменения произошли и в архитектуре ВМ. Это, прежде всего, микропрограммирование как эффективная техника построения устройств управления сложных процессоров, а также наступление эры конвейеризации и параллельной обработки. В области программного обеспечения определяющими вехами стали первые операционные системы и реализация режима разделения времени.

В первых ВМ третьего поколения использовались интегральные схемы с малой степенью интеграции (small-scale integrated circuits, SSI), где на одном кристалле размещается порядка 10 транзисторов. Ближе к концу рассматриваемого периода на смену SSI стали приходить интегральные схемы средней степени интеграции (medium-scale integrated circuits, MSI), в которых число транзисторов на кристалле увеличилось на порядок. К этому же времени относится повсеместное применение многослойных печатных плат. Все шире востребуются преимущества параллельной обработки, реализуемые за счет множественных функциональных блоков, совмещения во времени работы центрального процессора и операций ввода/вывода, конвейеризации потоков команд и данных.

В 1964 году Сеймур Крей (Seymour Cray, 1925–1996) построил вычислительную систему CDC 6600, в архитектуру которой впервые был заложен функциональный параллелизм. Благодаря наличию 10 независимых функциональных блоков, способных работать параллельно, и 32 независимых модулей памяти удалось достичь быстродействия в 1 MFLOPS (миллион операций с плавающей запятой в секунду). Пятью годами позже Крей создал CDC 7600 с конвейеризированными функциональными блоками и быстродействием 10 MFLOPS. CDC 7600 называют первой конвейерной вычислительной системой (конвейерным процессором). Революционной вехой в истории ВТ стало создание семейства вычислительных машин IBM 360, архитектура и программное обеспечение которых на долгие годы служили эталоном для последующих больших универсальных ВМ (mainframes). В машинах этого семейства нашли воплощение многие новые для того периода идеи, в частности: предварительная выборка команд, отдельные блоки для операций с фиксированной и плавающей запятой, конвейеризация команд, кэш-память. К третьему поколению ВС относятся также первые параллельные вычислительные системы: SOLOMON корпорации Westinghause и ILLIAC IV — совместная разработка Иллинойского университета и компании Burroughs. Третье поколение ВТ

ознаменовалось также появлением первых конвейерно-векторных ВС: TI-ASC (Texas Instruments Advanced Scientific Computer) и STAR-100 фирмы CBC.

Среди вычислительных машин, разработанных в этот период в СССР, прежде всего необходимо отметить «быстродействующую электронно-счетную машину» — БЭСМ-6 (С. А. Лебедев) с производительностью 1 млн операций/с. Продолжением линии М-20 стали М-220 и М-222 с производительностью до 200 000 операций/с. Оригинальная ВМ для инженерных расчетов «Мир-1» была создана под руководством В. М. Глушкова. В качестве входного языка этой ВМ использован язык программирования высокого уровня «Аналитик», во многом напоминающий язык Алгол.

В сфере программного обеспечения необходимо отметить создание в 1970 году Кеном Томпсоном (Kenneth Thompson) из Bell Labs языка B, прямого предшественника популярного языка программирования C, и появление ранней версии операционной системы UNIX.

Четвертое поколение (1972–1984)

Отсчет четвертого поколения обычно ведут с перехода на интегральные микросхемы большой (large-scale integration, LSI) и сверхбольшой (very large-scale integration, VLSI) степени интеграции. К первым относят схемы, содержащие около 1000 транзисторов на кристалле, в то время как число транзисторов на одном кристалле VLSI имеет порядок 100 000. При таких уровнях интеграции стало возможным уместить в одну микросхему не только центральный процессор, но и вычислительную машину (ЦП, основную память и систему ввода/вывода).

Конец 70-х и начало 80-х годов — это время становления и последующего победного шествия микропроцессоров и микроЭВМ, что, однако, не снижает важности изменений, произошедших в архитектуре других типов вычислительных машин и систем.

Одним из наиболее значимых событий в области архитектуры ВМ стала идея вычислительной машины с сокращенным набором команд (RISC, Redused Instruction Set Computer), выдвинутая в 1975 году и впервые реализованная в 1980 году. В упрощенном изложении суть концепции RISC заключается в сведении набора команд ВМ к наиболее употребительным простейшим командам. Это позволяет упростить схемотехнику процессора и добиться резкого сокращения времени выполнения каждой из «простых» команд. Более сложные команды реализуются как подпрограммы, составленные из быстрых «простых» команд.

В ВМ и ВС четвертого поколения практически уходят со сцены ЗУ на магнитных сердечниках, и основная память строится из полупроводниковых запоминающих устройств (ЗУ). До этого использование полупроводниковых ЗУ ограничивалось лишь регистрами и кэш-памятью.

В сфере высокопроизводительных вычислений доминируют векторные вычислительные системы, более известные как суперЭВМ. Разрабатываются новые параллельные архитектуры, однако подобные работы пока еще носят экспериментальный характер. На замену большим ВМ, работающим в режиме разделения времени,

приходят индивидуальные микроЭВМ и рабочие станции (этим термином обозначают сетевой компьютер, использующий ресурсы сервера).

В области программного обеспечения выделим появление языков программирования сверхвысокого уровня, таких как FP (functional programming — функциональное программирование) и Пролог (Prolog, programming in logic). Эти языки ориентированы на *аппликативный* и *декларативный стили программирования* соответственно, в отличие от Паскаля, С, Фортрана и т. д. — языков *императивного стиля программирования*. При аппликативном стиле вычисления задаются только как вызовы функций. При декларативном стиле программист дает математическое описание того, что должно быть вычислено, а детали того, каким образом это должно быть сделано, возлагаются на компилятор и операционную систему. Такие языки пока используются недостаточно широко, но выглядят многообещающими для ВС с массовым параллелизмом, состоящими из более чем 1000 процессоров. В компиляторах для ВС четвертого поколения начинают применяться сложные методы оптимизации кода.

Два события в области программного обеспечения связаны с Кеном Томпсоном (Kenneth Thompson) и Деннисом Ритчи (Dennis Ritchie) из Bell Labs. Это создание языка программирования С и его использование при написании операционной системы UNIX для машины DEC PDP-11. Такая форма написания операционной системы позволила быстро распространить UNIX на многие ВМ.

Пятое поколение (1984–1990)

Главным поводом для выделения вычислительных систем второй половины 80-х годов в самостоятельное поколение стало стремительное развитие ВС с сотнями процессоров, ставшее побудительным мотивом для прогресса в области параллельных вычислений. Ранее параллелизм вычислений выражался лишь в виде конвейеризации, векторной обработки и распределения работы между небольшим числом процессоров. Вычислительные системы пятого поколения обеспечивают такое распределение задач по множеству процессоров, при котором каждый из процессоров может выполнять задачу отдельного пользователя.

В рамках пятого поколения в архитектуре вычислительных систем сформировались два принципиально различных подхода: архитектура с совместно используемой памятью и архитектура с распределенной памятью.

Характерным примером первого подхода может служить система Sequent Balance 8000, в которой имеется большая основная память, разделяемая 20 процессорами. Помимо этого, каждый процессор оснащен собственной кэш-памятью. Каждый из процессоров может выполнять задачу своего пользователя, но при этом в составе программного обеспечения имеется библиотека подпрограмм, позволяющая программисту привлекать для решения своей задачи более одного процессора. Система широко использовалась для исследования параллельных алгоритмов и техники программирования.

Второе направление развития систем пятого поколения — системы с распределенной памятью, где каждый процессор обладает своим модулем памяти, а связь между

процессорами обеспечивается сетью взаимосвязей. Примером такой ВС может служить система iPSC-1 фирмы Intel, более известная как «гиперкуб». Максимальный вариант системы включал 128 процессоров. Применение распределенной памяти позволило устранить ограничения в пропускной способности тракта «процессор-память», но потенциальным «узким местом» здесь становится сеть взаимосвязей.

Наконец, третье направление в архитектуре вычислительных систем пятого поколения — это ВС, в которых несколько тысяч достаточно простых процессоров работают под управлением единого устройства управления и одновременно производят одну и ту же операцию, но каждый над своими данными. К этому классу можно отнести Connection Machine фирмы Thinking Machines Inc. и MP-1 фирмы MasPar Inc.

В научных вычислениях по-прежнему ведущую роль играют векторные суперЭВМ. Многие производители предлагают более эффективные варианты с несколькими векторными процессорами, но число таких процессоров обычно невелико (от 2 до 8).

RISC-архитектура выходит из стадии экспериментов и становится базовой архитектурой для рабочих станций (workstations).

Знаковой приметой рассматриваемого периода стало стремительное развитие технологий глобальных и локальных компьютерных сетей. Это стимулировало изменения в технологии работы индивидуальных пользователей. В противовес мощным универсальным ВС, работающим в режиме разделения времени, пользователи все более отдают предпочтение подключенным к сети индивидуальным рабочим станциям. Такой подход позволяет для решения небольших задач задействовать индивидуальную машину, а при необходимости в большой вычислительной мощности обратиться к ресурсам подсоединенных к той же сети мощных файл-серверов или суперЭВМ.

Шестое поколение (1990–)

На ранних стадиях эволюции вычислительных средств смена поколений ассоциировалась с революционными технологическими прорывами. Каждое из первых четырех поколений имело четко выраженные отличительные признаки и вполне определенные хронологические рамки. Последующее деление на поколения уже не столь очевидно и может быть понятно лишь при ретроспективном взгляде на развитие вычислительной техники. Пятое и шестое поколения в эволюции ВТ — это отражение нового качества, возникшего в результате последовательного накопления частных достижений, главным образом в архитектуре вычислительных систем и, в несколько меньшей мере, в сфере технологий.

Поводом для начала отсчета нового поколения стали значительные успехи в области параллельных вычислений, связанные с широким распространением вычислительных систем с массовым параллелизмом. Особенности организации таких систем, обозначаемых аббревиатурой MPP (massively parallel processing), будут рассмотрены в последующих главах. Здесь же упрощенно определим их как совокупность большого количества (до нескольких тысяч) взаимодействующих, но

достаточно автономных вычислительных машин. По вычислительной мощности такие системы уже успешно конкурируют с суперЭВМ, которые, как ранее отмечалось, по своей сути являются векторными ВС. Появление вычислительных систем с массовым параллелизмом дало основание говорить о производительности, измеряемой в TFLOPS (1 TFLOPS соответствует 1012 операциям с плавающей запятой в секунду).

Вторая характерная черта шестого поколения — резко возросший уровень рабочих станций. В процессорах новых рабочих станций успешно совмещаются RISC-архитектура, конвейеризация и параллельная обработка. Некоторые рабочие станции по производительности сопоставимы с суперЭВМ четвертого поколения. Впечатляющие характеристики рабочих станций породили интерес к гетерогенным (неоднородным) вычислениям, когда программа, запущенная на одной рабочей станции, может найти в локальной сети не занятые в данный момент другие станции, после чего вычисления распараллеливаются и на эти простаивающие станции.

Наконец, третьей приметой шестого поколения в эволюции ВТ стал взрывной рост глобальных сетей. Этот момент, однако, выходит за рамки данного учебника, поэтому далее комментироваться не будет.

Завершая обсуждение эволюции ВТ, отметим, что верхняя граница шестого поколения хронологически пока не определена, и дальнейшее развитие вычислительной техники может внести в его характеристику новые коррективы. Не исключено также, что последующие события дадут повод говорить и об очередном поколении.

Типы структур вычислительных машин и систем

Достоинства и недостатки архитектуры вычислительных машин и систем изначально зависят от способа соединения компонентов. При самом общем подходе можно говорить о двух основных типах структур вычислительных машин и двух типах структур вычислительных систем.

Структуры вычислительных машин

В настоящее время примерно одинаковое распространение получили два способа построения вычислительных машин: *с непосредственными связями* и *на основе шины*.

Типичным представителем первого способа может служить классическая фоннеймановская ВМ (см. рис. 1.3). В ней между взаимодействующими устройствами (процессор, память, устройство ввода/вывода) имеются непосредственные связи. Особенности связей (число линий в шинах, пропускная способность и т. п.) определяются видом информации, характером и интенсивностью обмена. Достоинством архитектуры с непосредственными связями можно считать возможность развязки «узких мест» путем улучшения структуры и характеристик только определенных связей, что экономически может быть наиболее выгодным решением. У фоннеймановских ВМ таким «узким местом» является канал пересылки данных между ЦП и памятью и «развязать» его достаточно непросто [50]. Кроме того, ВМ с непосредственными связями плохо поддаются реконфигурации.

В варианте с общей шиной все устройства вычислительной машины подключены к магистральной шине, служащей единственным трактом для потоков команд, данных и управления (рис. 1.4). Наличие общей шины существенно упрощает реализацию ВМ, позволяет легко менять состав и конфигурацию машины. Благодаря этим свойствам шинная архитектура получила широкое распространение в мини- и микроЭВМ. Вместе с тем именно с шиной связан и основной недостаток архитектуры: в каждый момент передавать информацию по шине может только одно устройство. Основную нагрузку на шину создают обмены между процессором и памятью, связанные с извлечением из памяти команд и данных и записью в память результатов вычислений. На операции ввода/вывода остается лишь часть пропускной способности шины. Практика показывает, что даже при достаточно быстрой шине для 90 % приложений этих остаточных ресурсов обычно не хватает, особенно в случае ввода или вывода больших массивов данных.

В целом следует признать, что при сохранении фон-неймановской концепции последовательного выполнения команд программы (одна команда в единицу времени) шинная архитектура в чистом ее виде оказывается недостаточно эффективной. Более распространена *архитектура с иерархией шин*, где помимо магистральной шины имеется еще несколько дополнительных шин. Они могут обеспечивать непосредственную связь между устройствами с наиболее интенсивным обменом, например процессором и кэш-памятью. Другой вариант использования дополнительных шин — объединение однотипных устройств ввода/вывода с последующим выходом с дополнительной шины на магистральную. Все эти меры позволяют снизить нагрузку на общую шину и более эффективно расходовать ее пропускную способность.

Структуры вычислительных систем

Понятие «вычислительная система» предполагает наличие множества процессоров или законченных вычислительных машин, при объединении которых используется один из двух подходов.

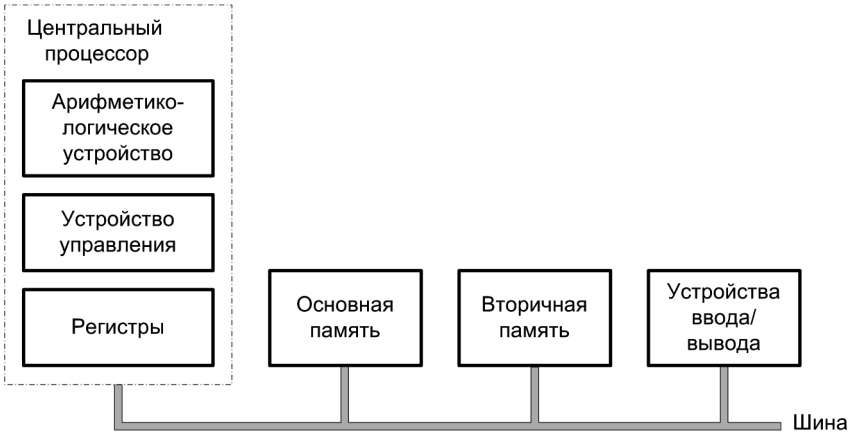


Рис. 1.4. Структура вычислительной машины на базе общей шины

В вычислительных *системах с общей памятью* (рис. 1.5) имеется общая основная память, совместно используемая всеми процессорами системы. Связь процессоров с памятью обеспечивается с помощью коммуникационной сети, чаще всего вырождающейся в общую шину. Таким образом, структура ВС с общей памятью аналогична рассмотренной выше архитектуре с общей шиной, в силу чего ей свойственны те же недостатки. Применительно к вычислительным системам данная схема имеет дополнительное достоинство: обмен информацией между процессорами не связан с дополнительными операциями и обеспечивается за счет доступа к общим областям памяти.

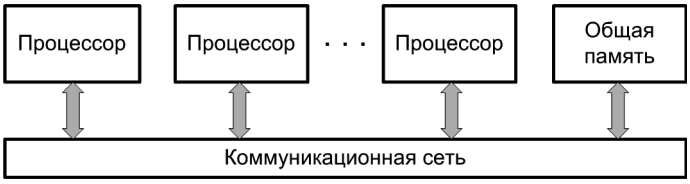


Рис. 1.5. Структура вычислительной системы с общей памятью

Альтернативный вариант организации — *распределенная* система, где общая память вообще отсутствует, а каждый процессор обладает собственной локальной памятью (рис. 1.6). Часто такие системы объединяют отдельные ВМ. Обмен информацией между составляющими системы обеспечивается с помощью коммуникационной сети посредством обмена сообщениями.

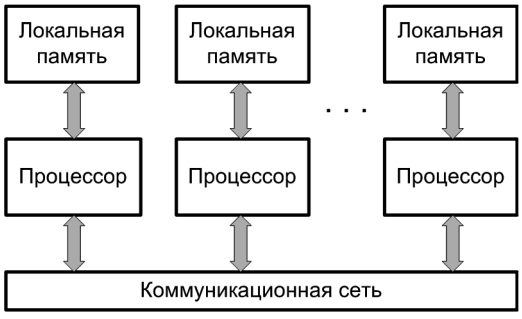


Рис. 1.6. Структура распределенной вычислительной системы

Подобное построение ВС снимает ограничения, свойственные для общей шины, но приводит к дополнительным издержкам на пересылку сообщений между процессорами или машинами.

Основные показатели вычислительных машин

Использование конкретной вычислительной машины имеет смысл, если ее показатели соответствуют показателям, определяемым требованиями к реализации заданных алгоритмов. В качестве основных показателей ВМ обычно рассматривают: емкость памяти, быстродействие и производительность, стоимость и надежность

[20]. Емкость — это наибольшее количество единиц данных, которое может одновременно храниться в памяти. Развернутое пояснение емкости приведено в главе 6.

Быстродействие

Целесообразно рассматривать два вида быстродействия: номинальное и среднее.

Номинальное быстродействие характеризует возможности ВМ при выполнении стандартной операции. В качестве стандартной обычно выбирают короткую операцию сложения. Если обозначить через $\tau_{сл}$ время сложения, то номинальное быстродействие определится из выражения:

$$V_{ном} = \frac{1}{\tau_{сл}} \left[\frac{оп}{с}\right].$$

Среднее быстродействие характеризует скорость вычислений при выполнении эталонного алгоритма или некоторого класса алгоритмов. Величина среднего быстродействия зависит как от параметров ВМ, так и от параметров алгоритма и определяется соотношением:

$$V_{ср} = \frac{N}{T_э},$$

где $T_э$ — время выполнения эталонного алгоритма; N — количество операций, содержащихся в эталонном алгоритме.

Обозначим через n_i число операций i-го типа; l — количество типов операций в алгоритме ($i = 1, 2, ..., l$); τ_i — время выполнения операции i-го типа.

Время выполнения эталонного алгоритма рассчитывается по формуле:

$$T_э = \sum_{i=1}^{l} \tau_i n_i. \qquad (1.1)$$

Подставив (1.1) в выражение для $V_{ср}$, получим

$$V_{ср} = \frac{N}{\sum_{i=1}^{l} \tau_i n_i}. \qquad (1.2)$$

Разделим числитель и знаменатель в (1.2) на N:

$$^{ср} \quad \overline{\qquad\qquad} \qquad (1.3)$$
$$\sum —$$

Обозначив частоту появления операции i-го типа в (1.3) через $q_i = \frac{n_i}{N}$, запишем окончательную формулу для расчета среднего быстродействия:

$$V_{\text{ср}} = \frac{1}{\sum_{i=1}^{l} q_i \tau_i} \left[\frac{\text{оп}}{\text{с}}\right]. \quad (1.4)$$

В выражении (1.4) вектор $\{\tau_1, \tau_2, ..., \tau_l\}$ характеризует систему команд ВМ, а вектор $\{q_1, q_2, ..., q_l\}$, называемый частотным вектором операций, характеризует алгоритм.

Очевидно, что для эффективной реализации алгоритма необходимо стремиться к увеличению $V_{\text{ср}}$. Если $V_{\text{ном}}$ главным образом отталкивается от быстродействия элементной базы, то $V_{\text{ср}}$ очень сильно зависит от оптимальности выбора команд ВМ.

Формула (1.4) позволяет определить среднее быстродействие машины при реализации одного алгоритма. Рассмотрим более общий случай, когда полный алгоритм состоит из нескольких частных, периодически повторяемых алгоритмов. Среднее быстродействие при решении полной задачи рассчитывается по формуле:

$$V_{\text{ср}}^{n} = \frac{1}{\sum_{j=1}^{m}\sum_{i=1}^{l} \beta_j q_{ji} \tau_i} \left[\frac{\text{оп}}{\text{с}}\right], \quad (1.5)$$

где m — количество частных алгоритмов; β_j — частота появления операций j-го частного алгоритма в полном алгоритме; q_{ij} — частота операций i-го типа в j-м частном алгоритме.

Обозначим через N_j и T_j — количество операций и период повторения j-го частного алгоритма; $T_{\max} = \max_j(T_1, ..., T_j, ..., T_m)$ — период повторения полного алгоритма; $\alpha_j = \frac{T_{\max}}{T_j}$ — цикличность включения j-го частного алгоритма в полном алгоритме.

Тогда за время $T_{\max}$ в ВМ будет выполнено $N_{\max} = \sum_{j=1}^{m} \alpha_j N_j$ операций, а частоту появления операций j-го частного алгоритма в полном алгоритме можно определить из выражения:

$$\beta_j = \frac{\alpha_j N_j}{N_{\max}}. \quad (1.6)$$

Для расчета по формулам (1.5, 1.6) необходимо знать параметры ВМ, представленные вектором $\{\tau_1, \tau_2, ..., \tau_l\}$, параметры каждого j-го частного алгоритма — вектор $\{q_{j1}, q_{j2}, ..., q_{jl}\}$ и параметры полного алгоритма — вектор $\{\beta_1, \beta_2, ..., \beta_m\}$.

Производительность ВМ оценивается количеством эталонных алгоритмов, выполняемых в единицу времени:

$$П = \frac{1}{T_э} \left[\frac{\text{задач}}{\text{с}}\right].$$

Производительность при выполнении полного алгоритма оценивается по формуле:

$$П_{\text{п}} = \frac{1}{\sum_{j=1}^{m}\sum_{i=1}^{l}\alpha_j N_j q_{ij}\tau_i} \left[\frac{\text{задач}}{\text{с}}\right]. \quad (1.7)$$

Производительность является более универсальным показателем, чем среднее быстродействие, поскольку в явном виде зависит от порядка прохождения задач через ВМ.

Надежность

Надежность — свойство ВМ правильно выполнять заданные программы (алгоритмы) в условиях воздействия различных случайных факторов. Эти факторы могут мешать достижению цели вычислений, вызывая отказы и сбои.

Отказ — это событие, заключающееся в нарушении работоспособности ВМ.

Сбой — самоустраняющийся отказ, приводящий к кратковременному нарушению работоспособности.

Обычно надежность определяют с помощью вероятности безотказной работы $P(t)$.

Вероятность безотказной работы показывает вероятность события, состоящего в том, что в пределах заданной наработки (времени t) не возникнет отказ ВМ.

Чаще всего вероятность безотказной работы определяют через интенсивность отказов $\lambda(t)$.

Интенсивность отказов — это среднее число отказов за единицу времени. Как правило, интенсивность отказов задается в расчете на один час работы, то есть имеет размерность [$ч^{-1}$]. Интенсивность отказов зависит от сложности ВМ, характеризуемой количеством комплектующих элементов и соединений, и уровня технологии производства машины.

Обычно принимается, что интенсивность отказов элементов ВМ — постоянная величина ($\lambda = \text{const}$). В этом случае вероятность безотказной работы ВМ определяется экспоненциальным законом

$$P(t) = e^{-\lambda t} \approx 1 - \lambda t.$$

На практике нужно учитывать, что ВМ состоит из элементов различных типов. Для случая, когда отказы различных элементов взаимонезависимы и каждый из них приводит к отказу всей ВМ, можно записать:

$$\lambda = \sum_{i=1}^{m} \lambda_i k_i,$$

где λ_i — интенсивность отказов элементов i-го типа; k_i — количество элементов i-го типа; m — количество типов элементов.

Интенсивность отказов — наиболее удобный показатель надежности простых элементов. Причина состоит в том, что λ-характеристики многих элементов

электроники и компьютерной техники имеют постоянные числовые значения. Кроме того, по λ-характеристикам довольно просто вычислить все показатели надежности ВМ.

Например, важный показатель восстанавливаемой ВМ — *среднее время между соседними отказами* T_0 (другое название — *наработка на отказ*) — определяется по формуле

$$T_0 = \frac{1}{\lambda}.$$

Положим, что персональный компьютер состоит из 200 однотипных элементов с интенсивностью отказов $\lambda_i = 10^{-8}$ 1/ч. Тогда

$$T_0 = \frac{1}{200 \cdot 10^{-8}} = 5 \cdot 10^5 \text{ ч} \approx 57 \text{ лет}.$$

После возникновения отказа ВМ ремонтируют для восстановления работоспособности. Среднее время восстановления $T_в$ также случайная величина. Оно зависит от количества элементов ВМ и техники поиска и устранения отказов.

С позиций надежности эксплуатационные свойства ВМ обычно характеризуют *коэффициентом готовности*

$$K_г = \frac{T_0}{T_0 + T_в}.$$

Коэффициент готовности имеет двоякий смысл:

1. Показывает долю времени, в течение которого ВМ работоспособна, то есть может решать задачи.
2. Демонстрирует вероятность того, что в любой произвольный момент времени ВМ работоспособна.

Стоимость

Стоимость S вычислительной машины равна суммарной стоимости оборудования, входящего в ее состав.

При увеличении стоимости могут решаться следующие задачи:

- повышение производительности;
- повышение надежности (за счет введения резервных элементов);
- повышение точности вычислений.

Конечно, практика решения перечисленных задач требовала установления зависимости каждого результата от стоимости. Например, долго считалось, что повышение производительности ВМ связано со стоимостью законом Гроша (К. Найта)

$$П = kS^2,$$

где k — коэффициент пропорциональности; П — производительность ВМ.

Закон Гроша фактически утверждал, что за счет беспредельного увеличения количества аппаратуры можно добиться беспредельного повышения производительности

ВМ. Исследования доказали ошибочность такого суждения: существует ограничение на предельное количество оборудования в составе одной ВМ.

Критерии эффективности вычислительных машин

Вычислительную машину можно определить множеством показателей, характеризующих отдельные ее свойства. Возникает задача введения меры для оценки степени приспособленности ВМ к выполнению возложенных на нее функций — меры эффективности.

Эффективность определяет степень соответствия ВМ своему назначению. Она измеряется либо количеством затрат, необходимых для получения определенного результата, либо результатом, полученным при определенных затратах. Произвести сравнительный анализ эффективности нескольких ВМ, принять решение на использование конкретной машины позволяет критерий эффективности.

Критерий эффективности — это правило, служащее для сравнительной оценки качества вариантов ВМ. Критерий эффективности можно назвать правилом предпочтения сравниваемых вариантов.

Строятся критерии эффективности на основе частных показателей эффективности (показателей качества). Способ связи между частными показателями определяет вид критерия эффективности.

Способы построения критериев эффективности

Возможны следующие способы построения критериев из частных показателей.

Выделение главного показателя. Из совокупности частных показателей A_1, A_2, ..., A_n выделяется один, например A_1, который принимается за главный. На остальные показатели накладываются ограничения:

$$A_i \leqslant A_{i\text{доп}} \ (i = 2, 3, ..., n),$$

где $A_{i\text{доп}}$ — допустимое значение i-го показателя. Например, если в качестве A_1 выбирается производительность, а на показатели надежности P и стоимости S накладываются ограничения, то критерий эффективности ВМ принимает вид:

$$П_{\text{упр}} \to \max, P \leqslant P_{\text{доп}}, S \leqslant S_{\text{доп}}.$$

Способ последовательных уступок. Все частные показатели нумеруются в порядке их важности: наиболее существенным считается показатель A_1, а наименее важным — A_n. Находится минимальное значение показателя A_1 — $\min A_1$ (если нужно найти максимум, то достаточно изменить знак показателя). Затем делается «уступка» первому показателю ΔA_1, и получается ограничение $\min A_1 + \Delta A_1$.

На втором шаге отыскивается $\min A_2$ при ограничении $A_1 \leq \min A_1 + \Delta A_1$. После этого выбирается «уступка» для A_2: $\min A_2 + \Delta A_2$. На третьем шаге отыскивается $min\ A_3$ при ограничениях $A_1 \leq \min A_1 + \Delta A_1$; $A_2 \leq \min A_2 + \Delta A_2$ и т. д. На последнем шаге ищут $\min A_n$ при ограничениях

$$A_1 \leq \min A_1 + \Delta A_1;$$

$$A_2 \le \min A_2 + \Delta A_2;$$
$$\dots$$
$$A_{n-1} \le \min A_{n-1} + \Delta A_{n-1}.$$

Полученный на этом шаге вариант вычислительной машины и значения ее показателей $A_1, A_2, ..., A_n$ считаются окончательными. Недостатком данного способа (критерия) является неоднозначность выбора ΔA_i.

Отношение частных показателей. В этом случае критерий эффективности получают в виде:

$$K_1 = \frac{A_1, A_2, ..., A_n}{B_1, B_2, ..., B_m} \to \max \tag{1.8}$$

или в виде

$$K_2 = \frac{B_1, B_2, ..., B_m}{A_1, A_2, ..., A_n} \to \min, \tag{1.9}$$

где A_i ($i = 1, 2, ..., n$) — частные показатели, для которых желательно увеличение численных значений, а B_i ($i = 1, 2, ..., m$) — частные показатели, численные значения которых нужно уменьшить. В частном случае критерий может быть представлен в виде:

$$K_3 = \frac{B_1}{A_1} \to min. \tag{1.10}$$

Наиболее популярной формой выражения (1.10) является критерий цены эффективного быстродействия:

$$K_4 = \frac{S}{V_{\text{ср}}} \to \min, \tag{1.11}$$

где S — стоимость, $V_{\text{ср}}$ — среднее быстродействие ВМ. Формула критерия K_4 характеризует аппаратные затраты, приходящиеся на единицу быстродействия.

Аддитивная форма. Критерий эффективности имеет вид:

$$K_5 = \sum_{i=1}^{n} \alpha_i A_i \to \max, \tag{1.12}$$

где $\alpha_1, \alpha_2, ..., \alpha_n$ — положительные и отрицательные весовые коэффициенты частных показателей. Положительные коэффициенты ставятся при тех показателях, которые желательно максимизировать, а отрицательные — при тех, которые желательно минимизировать.

Весовые коэффициенты могут быть определены методом экспертных оценок. Обычно они удовлетворяют условиям:

$$0 \le \alpha_i < 1, \ \sum_{i=1}^{n} \alpha_i = 1. \tag{1.13}$$

Основной недостаток критерия заключается в возможности взаимной компенсации частных показателей.

Мультипликативная форма. Критерий эффективности имеет вид:

$$K_6 = \prod_{i=1}^{n} A_i^{\alpha_i} \rightarrow \max, \tag{1.14}$$

где, в частном случае, коэффициенты α_i полагают равными единице.

От мультипликативной формы можно перейти к аддитивной, используя выражение:

$$\lg K_6 = \sum_{i=1}^{n} \alpha_i \lg A_i. \tag{1.15}$$

Критерий K_6 имеет тот же недостаток, что и критерий K_5.

Максиминная форма. Критерий эффективности описывается выражением:

$$K_7 = \min_i A_i \rightarrow \max. \tag{1.16}$$

Здесь реализована идея равномерного повышения уровня всех показателей за счет максимального «подтягивания» наихудшего из показателей (имеющего минимальное значение).

У максиминного критерия нет того недостатка, который присущ мультипликативному и аддитивному критериям.

Нормализация частных показателей

Частные показатели качества обычно имеют различную физическую природу и различные масштабы измерений, из-за чего их простое сравнение становится практически невозможным. Поэтому появляется задача приведения частных показателей к единому масштабу измерений, то есть их нормализация.

Рассмотрим отдельные способы нормализации.

Использование отклонения частного показателя от максимального.

$$\Delta A_i = A_{\max_i} - A_i. \tag{1.17}$$

В данном случае переходят к отклонениям показателей, однако способ не устраняет различия масштабов отклонений.

Использование безразмерной величины $\overline{A_i}$.

$$\overline{A_i} = \frac{A_{\max_i} - A_i}{A_{\max_i}}, \tag{1.18}$$

$$\overline{A_i} = \frac{A_i}{A_{\max_i}}. \tag{1.19}$$

Формула (1.18) применяется тогда, когда уменьшение A_i приводит к увеличению (улучшению) значения аддитивной формулы критерия. Выражение (1.19)

используется, когда к увеличению значения аддитивной формулы критерия приводит увеличение A_i.

Учет приоритета частных показателей

Необходимость в учете приоритетов возникает в случае, когда частные показатели имеют различную степень важности.

Приоритет частных показателей задается с помощью ряда приоритета I, вектора приоритета $(b_1, ..., b_q, ..., b_n)$ и вектора весовых коэффициентов $(\alpha_1, \alpha_2, ..., \alpha_n)$.

Ряд приоритета представляет собой упорядоченное множество индексов частных показателей $I = (1, 2, ..., n)$. Он отражает чисто качественные отношения доминирования показателей, а именно отношения следующего типа: показатель A_1 важнее показателя A_2, а показатель A_2 важнее показателя A_3 и т. д.

Элемент b_q вектора приоритета показывает, во сколько раз показатель A_q важнее показателя A_{q+1} (здесь A_q — показатель, которому отведен номер q в ряду приоритета). Если A_q и A_{q+1} имеют одинаковый ранг, то $b_q = 1$. Для удобства принимают $b_n = 1$.

Компоненты векторов приоритета и весовых коэффициентов связаны между собой следующим отношением:

$$b_q = \frac{\alpha_q}{\alpha_{q+1}}.$$

Зависимость, позволяющая по известным значениям b_i определить величину α_q, имеет вид:

$$\alpha_q = \frac{\prod_{i=q}^{n} b_i}{\sum_{q=1}^{n} \prod_{i=q}^{n} b_i}. \qquad (1.20)$$

Знание весовых коэффициентов позволяет учесть приоритет частных показателей.

Перспективы совершенствования архитектуры ВМ и ВС

Совершенствование архитектуры вычислительных машин и систем началось с момента появления первых ВМ и не прекращается по сей день. Каждое изменение в архитектуре направлено на абсолютное повышение производительности или, по крайней мере, на более эффективное решение задач определенного класса. Эволюцию архитектур определяют самые различные факторы, главные из которых показаны на рис. 1.7. Не умаляя роли ни одного из них, следует признать, что наиболее очевидные успехи в области средств вычислительной техники все же связаны с технологическими достижениями. Характер и степень влияния прочих факторов подробно описаны в [93] и в данном учебнике не рассматриваются.

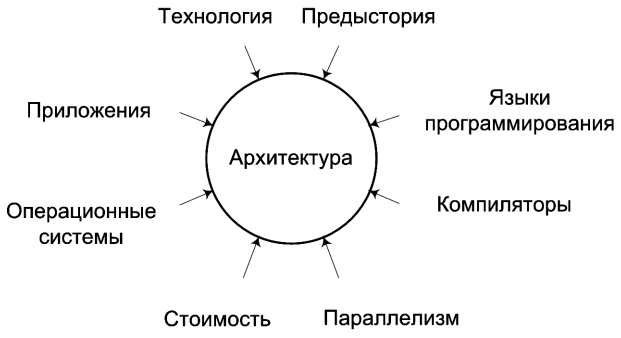


Рис. 1.7. Факторы, определяющие развитие архитектуры вычислительных систем

С каждым новым технологическим успехом многие из архитектурных идей переходят на уровень практической реализации. Очевидно, что процесс этот будет продолжаться и в дальнейшем, однако возникает вопрос: «Насколько быстро?» Косвенный ответ можно получить, проанализировав тенденции совершенствования технологий, главным образом полупроводниковых.

Тенденции развития больших интегральных схем

На современном уровне вычислительной техники подавляющее большинство устройств ВМ и ВС реализуется на базе полупроводниковых технологий в виде *сверхбольших интегральных микросхем* (СБИС). Каждое нововведение в области архитектуры ВМ и ВС, как правило, связано с необходимостью усложнения схемы процессора или его составляющих и требует размещения на кристалле СБИС все большего числа логических или запоминающих элементов. Задача может быть решена двумя путями: увеличением размеров кристалла и уменьшением площади, занимаемой на кристалле элементарным транзистором, с одновременным повышением плотности упаковки таких транзисторов на кристалле.

Пока основные успехи в плане увеличения емкости СБИС связаны с уменьшением размеров элементарных транзисторов и плотности их размещения на кристалле. Здесь тенденции эволюции СБИС хорошо описываются эмпирическим законом Мура[1] [120]. В 1965 году Мур заметил, что число транзисторов, которое удается разместить на кристалле микросхемы, удваивается каждые 12 месяцев. Он предсказал, что эта тенденция сохранится в 70-е годы, а начиная с 80-х темп роста начнет спадать. В 1995 году Мур уточнил свое предсказание, сделав прогноз, что удвоение числа транзисторов далее будет происходить каждые 24 месяца.

Современные технологии производства сверхбольших интегральных микросхем позволяют разместить на одном кристалле логические схемы всех компонентов процессора. В настоящее время процессоры всех вычислительных машин реализуются в виде одной или нескольких СБИС. Более того, во многих многопроцессорных ВС

[1] Гордон Мур (Gordon E. Moore) один из основателей фирмы Intel.

используются СБИС, где на одном кристалле располагаются сразу несколько процессоров (обычно не очень сложных).

Каждый успех создателей процессорных СБИС немедленно положительно отражается на характеристиках ВМ и ВС. Совершенствование процессорных СБИС ведется по двум направлениям: увеличение количества логических элементов, которое может быть размещено на кристалле, повышение быстродействия этих логических элементов. Увеличение быстродействия ведет к наращиванию производительности процессоров даже без изменения их архитектуры, а в совокупности с повышением плотности упаковки логических элементов открывает возможности для реализации ранее недоступных архитектурных решений.

В целом для процессорных СБИС можно сделать следующие выводы [133]:

- плотность упаковки логических схем процессорных СБИС каждые два года будет возрастать вдвое;
- удвоение внутренней тактовой частоты процессорных СБИС происходит в среднем каждые два года.

По мере повышения возможностей вычислительных средств растут и «аппетиты» программных приложений относительно емкости основной памяти. Эту ситуацию отражает так называемый закон Паркинсона: «Программное обеспечение увеличивается в размерах до тех пор, пока не заполнит всю доступную на данный момент память». В цифрах тенденция возрастания требований к емкости памяти выглядит так: увеличение в полтора раза каждые два года. Основная память современных ВМ и ВС формируется из СБИС полупроводниковых запоминающих устройств, главным образом динамических ОЗУ. Естественные требования к таким СБИС: высокие плотность упаковки запоминающих элементов и быстродействие, низкая стоимость.

Для СБИС памяти также подтверждается справедливость закона Мура и предсказанное им уменьшение темпов повышения плотности упаковки. В целом можно предсказать, что *число запоминающих элементов на кристалле будет возрастать в два раза каждые полтора года*.

С быстродействием СБИС памяти дело обстоит хуже. Высокая скорость процессоров уже давно находится в противоречии с относительной медлительностью запоминающих устройств основной памяти. Проблема постоянно усугубляется несоответствием темпов роста тактовой частоты процессоров и быстродействия памяти, и особых перспектив в этом плане пока не видно. Общая тенденция: *на двукратное уменьшение длительности цикла динамического ЗУ уходит примерно 15 лет*.

В плане снижения стоимости СБИС памяти перспективы весьма обнадеживающие. В течение достаточно длительного времени *стоимость в пересчете на один бит снижается примерно на 25–40 % в год*.

Перспективные направления исследований в области архитектуры вычислительных машин и систем

Основные направления исследований в области архитектуры ВМ и ВС можно условно разделить на две группы: эволюционные и революционные. К первой группе

следует отнести исследования, целью которых является совершенствование методов реализации уже достаточно известных идей. Изыскания, условно названные революционными, направлены на создание совершенно новых архитектур, принципиально отличных от уже ставшей традиционной фон-неймановской архитектуры.

Большинство из исследований, относимых к эволюционным, связано с совершенствованием архитектуры микропроцессоров (МП). В принципе, кардинально новых архитектурных подходов в микропроцессорах сравнительно мало. Основные идеи, лежащие в основе современных МП, были выдвинуты много лет тому назад, но из-за несовершенства технологии и высокой стоимости реализации нашли применение только в больших универсальных ВМ (мэйнфреймах) и суперЭВМ. Наиболее значимые из изменений в архитектуре МП связаны с повышением уровня параллелизма на уровне команд (возможности одновременного выполнения нескольких команд). Здесь в первую очередь следует упомянуть конвейеризацию, суперскалярную обработку и архитектуру с командными словами сверхбольшой длины (VLIW). После успешного переноса на МП глобальных архитектурных подходов «больших» систем основные усилия исследователей теперь направлены на частные архитектурные изменения. Примерами таких эволюционных архитектурных изменений могут служить: усовершенствованные методы предсказания переходов в конвейере команд, повышение частоты успешных обращений к кэш-памяти за счет усложненных способов буферизации и т. п.

Наблюдаемые нами достижения в области вычислительных средств широкого применения пока обусловлены именно «эволюционными» исследованиями. Однако уже сейчас очевидно, что, оставаясь в рамках традиционных архитектур, мы довольно скоро натолкнемся на технологические ограничения. Один из путей преодоления технологического барьера лежит в области нетрадиционных подходов. Исследования, проводимые в этом направлении, по нашей классификации отнесены к «революционным». Справедливость такого утверждения подтверждается первыми образцами ВС с нетрадиционной архитектурой.

Оценивая перспективы эволюционного и революционного развития вычислительной техники, можно утверждать, что на ближайшее время наибольшего прогресса можно ожидать на пути использования идей параллелизма на всех его уровнях и создания эффективной иерархии запоминающих устройств.

Глава 7

Организация шин

Интерфейс IrDA

Инфракрасный интерфейс IrDA (Infra red Data Assotiation) позволяет соединяться с периферийным оборудованием без кабеля при помощи ИК-излучения с длиной волны 880 нм. Порт IrDA позволяет устанавливать связь на коротком расстоянии до одного метра в режиме точка-точка. В цели IrDA входили низкое потребление и экономичность, поэтому интерфейс IrDA использует узкий ИК-диапазон с малой мощностью потребления, что позволяет создать недорогую аппаратуру.

Устройство инфракрасного интерфейса подразделяется на два основных блока: преобразователь и блок кодирования/декодирования (рис. 7.21).

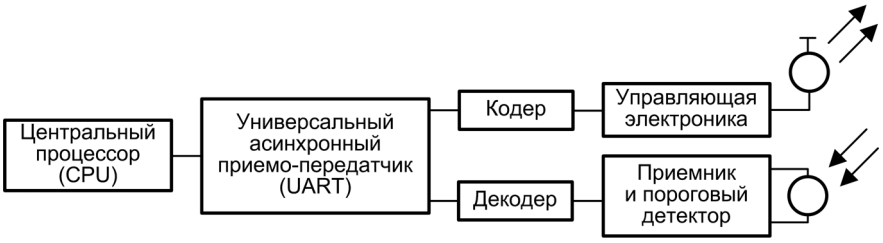


Рис. 7.21. Схема организации инфракрасной связи

Блоки обмениваются данными на уровне электрических сигналов. Перед передачей данные пакуются в кадры, состоящие из 10 битов, где байт данных обрамляется одним старт-битом в начале и одним стоп-битом в конце кадра. Порт IrDA использует универсальный асинхронный приемопередатчик UART (Universal Asynchronous Receiver Transmitter) и работает со скоростью передачи данных 2400–115 200 бит/с.

Перед началом передачи устройство выясняет, нет ли в ближайшей окрестности активности в ИК-диапазоне (не ведется ли какая-либо передача в пределах его досягаемости). Если такая активность обнаружена, то программе, выдающей запрос, посылается соответствующее сообщение, а передача откладывается. Если оба соединяющихся устройства являются компьютерами, то любое из них может быть ведущим. Выбор зависит от того, какое из устройств первым проявит инициативу. В процессе установления связи два устройства «договариваются» о максимальной скорости, с которой они оба могут работать, и ряде других параметров обмена. Все первичные передачи, выполняемые до фазы переговоров, по умолчанию ведутся на скорости 9,6 Кбит/с. Любая станция, не принимающая в данный момент времени участия в обмене, перед началом передачи должна прослушивать канал не менее

500 мс, чтобы убедиться в отсутствии трафика в ИК-диапазоне. С другой стороны, станция, участвующая в обмене, должна вести передачу не более 500 мс.

При всех своих достоинствах вариант «точка — точка» не всегда является лучшим вариантом реализации последовательного интерфейса, особенно при подключении высокоскоростных внешних устройств. Для поддержки внешних запоминающих и мультимедийных устройств предпочтение все чаще отдается многоточечным (Multipoint) интерфейсам, таким, например, как Thunderbolt.

Интерфейс Thunderbolt

Громкое имя Thunderbolt (переводится как «удар молнии») получил интерфейс, представляющий эффективную технологию ввода-вывода для обслуживания дисплеев высокого разрешения и высокоскоростных внешних устройств через единственный порт. Версия Thunderbolt 1 (2011 г.), уже применяемая во всех компьютерах Mac фирмы Apple, обеспечивает скорость передачи информации до 10 Гбит/с одновременно в обоих направлениях. Это в 2 раза быстрее, чем в USB 3, и в 12 раз быстрее, чем в FireWire 800. В версии Thunderbold 2 (2014 г.) скорость передачи данных увеличена до 20 Гбит/с.

Интерфейс является результатом совместной разработки компаний Intel и Apple. В Thunderbolt включены адаптеры для двух видов протоколов. Первым из них является протокол PCI Express, ориентированный на передачу данных. Второй протокол, DisplayPort, поддерживает пересылку информации о видео и звуке.

ПРИМЕЧАНИЕ

DisplayPort — стандарт сигнального интерфейса для цифровых дисплеев, рассматривается как наиболее современный интерфейс соединения аудио- и видеоаппаратуры (компьютера с дисплеем или компьютера и систем домашнего кинотеатра). Технология, реализованная в DisplayPort, позволяет передавать одновременно как видео-, так и аудиосигналы.

Технология Thunderbolt поддерживает доступ к высокоскоростным периферийным устройствам (внешним ЗУ, массивам RAID, мультимедийным устройствам) и дисплеям высокого разрешения через простой порт и кабель. Разъем Thunderbolt — это разъем Mini DisplayPort, через который помимо пересылки информации обеспечивается также энергопитание подключенных периферийных устройств (до 10 Вт). К кабелю можно подключать цепочки различных (даже очень медленных) устройств без ущерба для общей пропускной способности интерфейса. Дело в том, что шина «молнии» поддерживает разные скорости Thunderbolt-устройств. Так, Thunderbolt 2 позволяет одновременно подключать до трех современных мониторов с разрешением 4K (3840 × 2160 пикселов), работа которых сопряжена с пересылкой больших объемов информации на очень высоких скоростях.

Помимо высокой скорости ввода-вывода большим преимуществом интерфейса Thunderbolt считается поддержка передачи данных, видео, аудио и питания всего через один порт и кабель. Это избавляет от лишних проводов USB, опутывающих компьютер при работе с многочисленными периферийными устройствами.

Пользователь может подключить до семи устройств к каждому из портов Thunderbolt, связав их в одну цепочку (daisy-chain), которая реализует схему последовательного подключения. Такая топология предполагает, что каждое устройство из цепочки должно иметь два порта Thunderbolt.

Кроме основных адаптеров Thunderbolt разработаны и выпускаются переходники под популярные интерфейсы (FireWire, USB, SATA и др.), что существенно расширяет сферу использования Thunderbolt при сопряжении с разнообразными периферийными устройствами.

Новый интерфейс практически непосредственно подключается к шине PCI Express компьютера, поэтому у него есть такой недостаток, как слабая защита от внешних несанкционированных действий. Поскольку подключенное устройство Thunderbolt имеет неограниченный доступ к памяти компьютера, то, оказавшись в руках злодея, может нанести существенный ущерб.

Центральное звено интерфейса — микросхема контроллера Thunderbolt. Работает контроллер как миниатюрный роутер, быстро переключающийся в ходе передачи между двумя двунаправленными каналами данных. Эти каналы организованы в одном кабеле. Максимальная длина проводного кабеля — 3 м. Она может быть увеличена до нескольких десятков метров с помощью оптического кабеля, но при этом теряется возможность подачи электрического питания к подключаемому устройству. Заметим, что стоимость Thunderbolt возрастает из-за использования активных кабелей, в которые встроены микросхемы приемопередатчиков. Другими словами, пользователь платит не только за микросхемы-контроллеры в самих компьютерных устройствах, но и за электронику в кабеле.

Принцип работы интерфейса Thunderbolt иллюстрирует рис. 7.22.

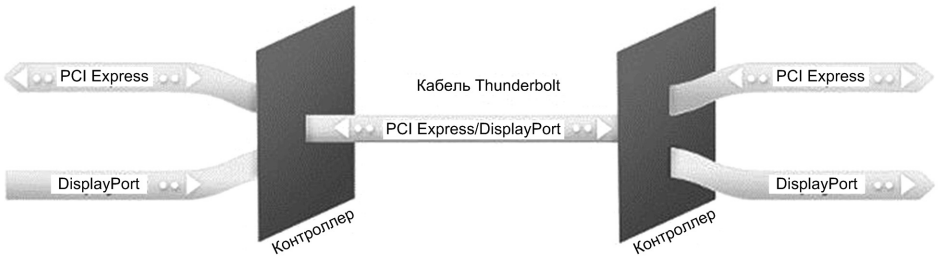


Рис. 7.22. Потоки информации через интерфейс Thunderbolt

Интерфейс осуществляет двунаправленную пересылку информации между ВМ и периферийными устройствами по двум каналам (канал PCI Express для данных, а канал DisplayPort для видео- и аудиоинформации), используя технику мультиплексирования на основе коммутатора-контроллера. Поскольку эти каналы разграничены между собой, передача информации по единому кабелю выполняется без задержек. Канал DisplayPort одновременно управляет дисплеем с разрешением более 1080 пикселов на строку и 8 каналами звука, что делает его идеальным при работе с видео- и аудиоприложениями. Поскольку Thunderbolt-устройства для

операционной системы выглядят как устройства PCI Express и DisplayPort, они могут использовать стандартные драйверы.

Динамика функционирования интерфейса Thunderbolt поддерживается четырьмя уровнями протоколов:

- уровнем кабеля и разъема;
- физическим уровнем;
- транспортным уровнем;
- прикладным уровнем.

Уровень кабеля и разъема обеспечивает доступ к среде передачи. Он определяет физические и электрические параметры порта разъема.

Протокол физического уровня отвечает за техническое обслуживание линии связи для обеспечения высокоэффективной передачи данных. К числу его задач относятся, например, обнаружение «горячего» подключения, а также обнаружение и кодирование данных.

ПРИМЕЧАНИЕ

«Горячее» подключение к портам означает, что пользователи могут добавлять устройства, не перезагружая свой компьютер.

Физический уровень был разработан так, чтобы при минимальных накладных расходах обеспечивать дуплексную связь со скоростью 10 Гбит/с для транспортного и прикладного уровней.

ПРИМЕЧАНИЕ

Некоторые системы связи позволяют передавать данные одновременно в обоих направлениях (используя разные частотные диапазоны для каждого из направлений). Соединения, обладающие такой возможностью, называются *дуплексными*. Скажем, двухколейная железная дорога, по которой поезда могут перемещаться во встречных направлениях, является своеобразной дуплексной линией связи.

Транспортный уровень является ключевым в работе Thunderbolt — именно он задает привлекательность интерфейса в качестве высокоскоростной технологии ввода-вывода. Перечислим некоторые из его характеристик.

- Высокопроизводительная архитектура коммутации с низким энергопотреблением.
- Поддержка высокоэффективного формата пакета с низкой себестоимостью и гибким качеством обслуживания, что позволяет мультиплексировать «пульсирующие» транзакции PCI Express с коммуникациями DisplayPort в одной линии связи. Транспортный уровень способен гибко распределять полосу пропускания линии связи с использованием приоритетов и механизмов резервирования полосы пропускания.
- Использование пакета небольшого размера для достижения малой задержки.
- Управление потоком на основе распределения кредитов, обеспечивающее потребность лишь в буферах малого размера.

- Симметричная архитектура, которая поддерживает гибкие топологии (звезду, дерево, схему цепочки и т. д.) и реализует одноранговое соединение устройств (с помощью программного обеспечения).
- Протокол синхронизации времени в сети, что позволяет всем подключенным устройствам синхронизировать свое время с точностью 8 нс.

Прикладной уровень содержит протоколы ввода-вывода, которые отображаются на транспортный уровень. Повторим, что изначально Thunderbolt обеспечивает полную поддержку протоколов PCI Express и DisplayPort. Эта функция реализуется адаптером протокола, который отвечает за эффективную инкапсуляцию отображаемой информации протокола в пакеты транспортного уровня. Отображенные пакеты протокола направляются между устройством-источником и устройством-приемником по пути, который может пролегать через несколько контроллеров Thunderbolt. В устройстве-приемнике адаптер протокола воссоздает отображенный протокол таким образом, что он ничем не отличается от исходного протокола, полученного устройством-источником. Благодаря выполняемому отображению протокола доступные интерфейсу устройства появляются в операционной среде целевого компьютера как устройства PCI Express или DisplayPort, тем самым позволяя использовать стандартные драйверы.

В целом эксперты утверждают, что интерфейс Thunderbolt демонстрирует хороший баланс производительности, простоты и гибкости, открывающий широкие возможности для разработчиков компьютеров.

Глава 12

Топология вычислительных систем

Сама идея многопроцессорной вычислительной системы предполагает обмен данными между компонентами этой ВС. Коммуникационная система ВС представляет собой сеть, *узлы* которой связаны трактами передачи данных — *каналами*. В роли узлов сети могут выступать процессоры, банки памяти, устройства ввода/вывода, коммутаторы либо несколько перечисленных элементов, объединенных в группу. Организация внутренних коммуникаций вычислительной системы называется *топологией*.

Топологию сети определяет множество узлов N, объединенных множеством каналов C. Связь между узлами обычно реализуется по двухточечной схеме (point-to-point). Любые два узла, связанные каналом связи, называют смежными узлами или соседями. Каждый канал $c = (x, y) \in C$ соединяет один узел-источник (source node) x с одним узлом-получателем (recipient node) y, где $x, y \in N$. Часто пары узлов соединяют два канала — по одному в каждом направлении. Канал характеризуют шириной w_c — числом сигнальных линий; частотой f_c — скоростью передачи битов по каждой сигнальной линии; задержкой t_c — временем пересылки бита из узла x в узел y. Для большинства каналов задержка находится в прямой зависимости от физической длины линии связи l_c и скорости распространения сигнала v: $l_c = v\, t_c$. Полоса пропускания канала b_c определяется выражением $b_c = w_c f_c$.

Классификация коммуникационных сетей

Классификацию сетей обычно производят по следующим признакам:

- стратегия синхронизации;
- стратегия коммутации;
- стратегия управления;
- топология.

Классификация по стратегии синхронизации

Две возможных стратегии синхронизации операций в сети — это синхронная и асинхронная. В синхронных сетях все действия жестко согласованы во времени,

что обеспечивается за счет единого генератора тактовых импульсов (ГТИ), сигналы которого одновременно транслируются во все узлы. В асинхронных сетях единого генератора нет, а функции синхронизации распределены по всей системе, причем в разных частях сети часто используются локальные ГТИ. Синхронные сети по сравнению с асинхронными обычно медленнее, но в них легче предотвращаются конфликтные ситуации.

Классификация по стратегии коммутации

По стратегии коммутации различают *сети с коммутацией соединений* и *сети с коммутацией пакетов*. Как в первом, так и во втором варианте информация пересылается в виде пакета. *Пакет* представляет собой группу битов, для обозначения которой применяют также термин *сообщение*.

В сетях с коммутацией соединений еще до начала пересылки сообщения формируется тракт от узла-источника до узла-получателя (путем соответствующей установки коммутирующих элементов сети). Этот тракт сохраняется вплоть до доставки пакета в пункт назначения. Пересылка сообщений между определенной парой узлов производится всегда по одному и тому же маршруту.

Сети с коммутацией пакетов предполагают, что сообщение самостоятельно находит свой путь к месту назначения. В отличие от сетей с коммутацией соединений, маршрут от исходного пункта к пункту назначения каждый раз может быть иным. Пакет последовательно проходит через узлы сети. Очередной узел запоминает принятый пакет в своем буфере временного хранения, анализирует его и решает, что с ним делать дальше. В зависимости от загруженности сети принимается решение о возможности немедленной пересылки пакета к следующему узлу и о дальнейшем маршруте следования пакета на пути к цели. Если все возможные тракты для перемещения пакета к очередному узлу заняты, в буфере узла формируется очередь пакетов, которая «рассасывается» по мере освобождения линий связи между узлами (при насыщении очереди, согласно одной из стратегий маршрутизации, может произойти так называемый «сброс хвоста» (tail drop) — отказ от вновь поступающих пакетов). Хотя сети с коммутацией пакетов по сравнению с сетями на базе коммутации соединений более эффективно используют ресурсы сети, время доставки сообщения в них сильно зависит от переменных задержек в перемещении пакетов.

Классификация по стратегии управления

Коммуникационные сети можно также классифицировать по тому, как в них организовано управление. В некоторых сетях, особенно с коммутацией соединений, принято *централизованное управление* (рис. 12.1). Процессоры посылают запрос на обслуживание в единый контроллер сети, который производит арбитраж запросов с учетом заданных приоритетов и устанавливает нужный маршрут. К данному типу следует отнести сети с шинной топологией. Процессорные матрицы также строятся как сети с централизованным управлением, которое осуществляется сигналами от центрального процессора. Приведенная схема применима и к сетям с коммутацией

пакетов. Здесь маршрут определяется адресом узла назначения, хранящимся в заголовке пакета. Большинство из серийно выпускаемых ВС имеют именно этот тип управления.

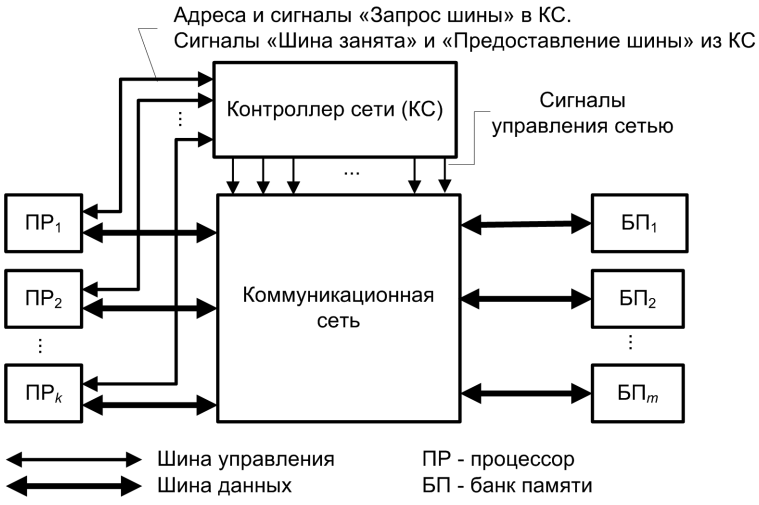


Рис. 12.1. Структура сети с централизованным управлением

В схемах с *децентрализованным управлением* функции управления распределены по узлам сети. Вариант с централизацией проще реализуется, но расширение сети в этом случае связано со значительными трудностями. Децентрализованные сети в плане подключения дополнительных узлов значительно гибче, однако взаимодействие узлов в таких сетях существенно сложнее.

В ряде сетей связь между узлами обеспечивается посредством множества коммутаторов, но существуют также сети с одним коммутатором. Наличие большого числа коммутаторов ведет к увеличению времени передачи сообщения, но позволяет использовать простые переключающие элементы. Подобные сети обычно строятся как многоступенчатые.

Классификация по топологии

Топологию коммуникационной сети (организацию внутренних коммуникаций многопроцессорной вычислительной системы) можно рассматривать как функцию отображения множества процессоров (ПР) и банков памяти (БП) на то же самое множество ПР и БП. Иными словами, топология описывает, каким образом процессоры и банки памяти могут быть соединены с другими процессорами и банками памяти.

В основу системы классификации обычно кладут разбиение всех возможных топологий на *статические* и *динамические*. В сетях со статическими топологиями структура связей фиксирована. В сетях с динамической топологией конфигурация соединений в процессе вычислений может быть оперативно изменена (с помощью программных средств).

Узел в сети может быть терминальным, то есть источником или приемником данных, коммутатором, пересылающим информацию с входного порта на выходной, или совмещать обе роли. В сетях *с непосредственными связями* (direct networks) каждый узел одновременно является как терминальным узлом, так и коммутатором, и сообщения пересылаются между терминальными узлами напрямую. В сетях *с косвенными связями* (indirect networks) узел может быть либо терминальным, либо коммутатором, но не одновременно, поэтому сообщения передаются опосредовано, с помощью выделенных коммутирующих узлов. (В дальнейшем условимся называть оба варианта «прямыми» и «косвенными» сетями, а терминальный узел будем называть «терминалом».) Существуют также такие топологии, которые нельзя однозначно причислить ни к прямым, ни к косвенным. Любую прямую сеть можно изобразить в виде косвенной, разделив каждый узел на два — терминальный узел и узел коммутации. Современные прямые сети реализуются именно таким образом — коммутатор отделяется от терминального узла и помещается в выделенный маршрутизатор. Основное преимущество прямых сетей в том, что коммутатор может использовать ресурсы терминальной части своего узла. Это становится существенным, если учесть, что, как правило, последний включает в себя вычислительную машину или процессор.

Схема классификации коммуникационных сетей на основе их топологии показана на рис. 12.2.

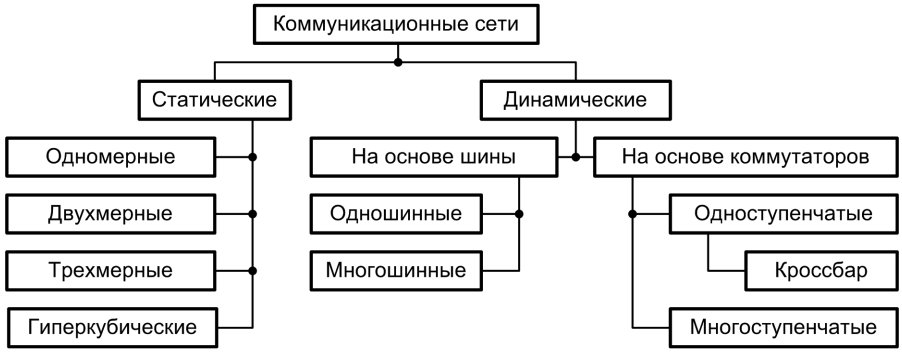


Рис. 12.2. Классификация коммуникационных сетей по топологии

Метрики сетевых соединений

Чтобы охарактеризовать сеть, обычно используют следующие параметры:

- размер сети;
- число связей;
- диаметр сети;
- степень узла;
- пропускная способность сети;
- задержка сети;

- связность сети;
- ширина бисекции сети;
- полоса бисекции сети.

Размер сети (N) численно равен количеству узлов, объединяемых сетью.

Число связей (I) — это суммарное количество каналов между всеми узлами сети. Иногда этот параметр называют *стоимостью сети*. В плане стоимости лучшей следует признать ту сеть, которая требует меньшего числа связей.

Диаметр сети (D), называемый также *коммуникационным расстоянием*, определяет минимальный путь, по которому проходит сообщение между двумя наиболее удаленными друг от друга узлами сети. *Путь* в сети — это упорядоченное множество каналов $P = \{c_1, c_2, ..., c_n\}$, по которым данные от узла-источника, последовательно переходя от одного промежуточного узла к другому, поступают на узел-получатель. Для обозначения отрезка пути между парой смежных узлов применяют термин *переход* (в живой речи также «транзит» и «хоп»). Минимальный путь от узла x до узла y — это путь с минимальным числом переходов. Если обозначить число переходов в минимальном пути от узла x до узла y через $H(x, y)$, то диаметр сети D — это наибольшее значение $H(x, y)$ среди всех возможных комбинаций x и y. Так, в цепочке из четырех узлов наибольшее число переходов будет между крайними узлами, и «диаметр» такой цепочки равен трем. С возрастанием диаметра сети увеличивается общее время прохождения сообщения, поэтому разработчики ВС стремятся по возможности обходиться меньшим диаметром.

Степень узла (d). Каждый узел сети x связан с прочими узлами множеством каналов $C_x = C_{Ix} \cup C_{Ox}$, где C_{Ix} — множество входных каналов, а C_{Ox} — множество выходных каналов. Степень узла x представляет собой сумму числа входных и выходных каналов узла, то есть она равна числу узлов сети, с которыми данный узел связан напрямую. Например, в сети, организованной в виде матрицы, где каждый узел связан только с ближайшими соседями (слева, справа, сверху и снизу), степень узла равна четырем. Увеличение степени узлов ведет к усложнению коммутационных устройств сети и, как следствие, к дополнительным задержкам в передаче сообщений. С другой стороны, повышение степени узлов позволяет реализовать топологии, имеющие меньший диаметр сети, и тем самым сократить время прохождения сообщения. Разработчики ВС обычно отдают предпочтение таким топологиям, где степень всех узлов одинакова, что позволяет строить сети по модульному принципу.

Пропускная способность сети (W) характеризуется количеством информации, которое может быть передано по сети в единицу времени. Обычно измеряется в мегабайтах в секунду или гигабайтах в секунду без учета издержек на передачу избыточной информации, например битов паритета.

Задержка сети (T) — это время, требуемое на прохождение сообщения через сеть. В сетях, где время передачи сообщений зависит от маршрута, говорят о минимальной, средней и максимальной задержках сети.

Связность сети (Q) можно определить как минимальное число параллельных трактов между любой парой узлов. Связность сети характеризует устойчивость

сети к повреждениям, то есть ее способность обеспечивать функционирование ВС при отказе компонентов сети.

Ширина бисекции сети (B). Для начала определим понятие *среза сети* $C(N_1, N_2)$ как множество каналов, разрыв которых разделяет множество узлов сети N на два непересекающихся подмножества узлов N_1 и N_2. Каждый элемент $C(N_1, N_2)$ — это канал, соединяющий узел из N_1 с узлом из N_2. Бисекция сети — это срез сети, разделяющий ее примерно пополам, то есть так, что $|N_2| \leqslant |N_1| \leqslant |N_2| + 1$. Ширину бисекции B характеризуют минимальным числом каналов, разрываемых при всех возможных бисекциях сети:

$$B = \min_{bisection} \left|C(N_1,\ N_2)\right|.$$

Ширина бисекции позволяет оценить число сообщений, которые могут быть переданы по сети одновременно, при условии что это не вызовет конфликтов из-за попытки использования одних и тех же узлов или линий связи.

Полоса бисекции сети (b) — это наименьшая полоса пропускания по всем возможным бисекциям сети. Она характеризует пропускную способность тех линий связи, которые разрываются при бисекции сети, и позволяет оценить наихудшую пропускную способность сети при попытке одномоментной передачи нескольких сообщений, если эти сообщения должны проходить из одной половины сети в другую. Полоса бисекции b определяется выражением $b = \min_{bisection} B(N_1,\ N_2)$. Для сетей с одинаковой шириной полосы b_c во всех каналах справедливо: $b = b_c \times B$. Малое значение полосы бисекции свидетельствует о возможности конфликтов при одновременной пересылке нескольких сообщений.

Функции маршрутизации данных

Важнейшим вопросом при выборе топологии является способ маршрутизации данных, то есть правило выбора очередного узла, которому пересылается сообщение. Основой маршрутизации служат адреса узлов. Каждому узлу в сети присваивается уникальный адрес. Исходя из этих адресов, а точнее их двоичных представлений, производится соединение узлов в статических топологиях или их коммутация в топологиях динамических. В сущности, принятая система соответствия между двоичными кодами адресов смежных узлов — *функция маршрутизации данных* — и определяет топологию сети. Последнюю можно описать как набор функций маршрутизации, задающий порядок выбора промежуточных узлов на пути от узла-источника к узлу-получателю. В некоторых топологиях используется единая для всей сети функция маршрутизации, в других — многоступенчатых — при переходе от одной ступени к другой может применяться иная функция маршрутизации.

Функция маршрутизации данных определяет алгоритм манипуляции битами адреса узла-источника для определения адреса узла-получателя. Ниже приводится формальное описание наиболее распространенных функций маршрутизации данных. Для всех функций предполагается, что размер сети равен N, а разрядность

адреса — n, где $n = \log_2 N$. Биты адреса обозначены как x_i. В приводимых ниже примерах принято, что $N = 8$ ($n = 3$).

Кубическая перестановка

Функция *кубической перестановки* (cube permutation) отвечает следующему соотношению:

$$E_i(x_{n-1} \ldots x_{i+1}x_i x_{i-1} \ldots x_0) = x_{n-1} \ldots x_{i+1}\overline{x_i}\, x_{i-1} \ldots x_0, \quad 0 \le i \le n-1.$$

Двоичное представление адреса узла-получателя получается путем инвертирования i-го бита в адресе источника.

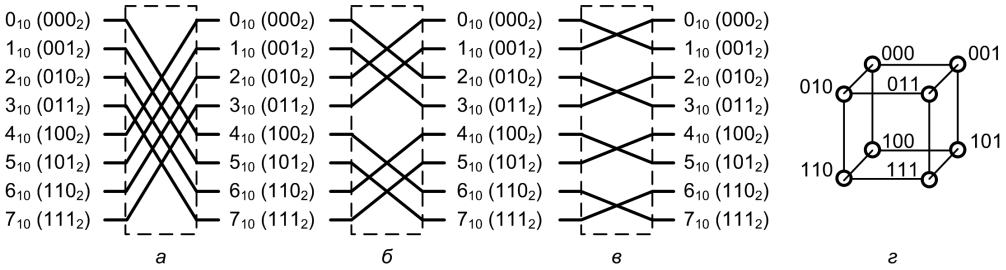


Рис. 12.3. Примеры топологий с кубической перестановкой: *а* — при $i = 2$; *б* — при $i = 1$; *в* — при $i = 0$; *г* — трехмерный гиперкуб

На рис. 12.3, *а*, *б*, *в* показана топология связей в сети, построенной в соответствии с функцией кубической перестановки, для трех значений i. Примером использования данной функции маршрутизации, где использованы все три возможных значения i, может служить топология трехмерного гиперкуба (рис. 12.3, *г*). Вариант функции для $i = 0$ известен также под названием *обменной перестановки* (exchange permutation).

Тасующая подстановка

Функция *тасующей подстановки* может быть реализована в одном из четырех вариантов, из которых наиболее распространены два: *совершенная тасующая подстановка* (perfect shuffle permutation) и *инверсная совершенная тасующая подстановка* (inverse perfect shuffle permutation). Ниже приведены формальные описания этих вариантов, а на рис. 12.4 — примеры соответствующих им топологий.

- *совершенная тасующая подстановка*:

$$S(x_{n-1}x_{n-2} \ldots x_1x_0) = x_{n-2} \ldots x_1x_0x_{n-1}.$$

  Из приведенной формулы видно, что адрес узла-получателя может быть получен из двоичного кода узла-источника циклическим сдвигом этого кода влево на одну позицию. Если использовать аналогию с картами, то тасующая подстановка эквивалентна разбиению колоды карт на две половины с последующим равномерным чередованием карт из каждой половины.

- *инверсная совершенная тасующая подстановка*:

$$U(x_{n-1}x_{n-2} \ldots x_1x_0) = x_0x_{n-1} \ldots x_2x_1.$$

Здесь также используется циклический сдвиг, но вправо.

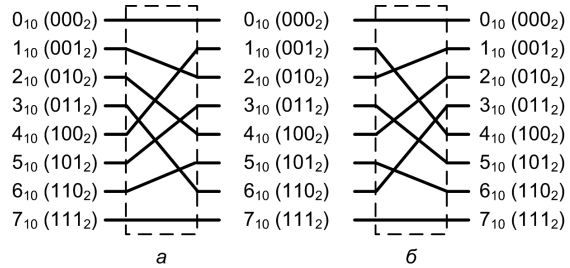


Рис. 12.4. Примеры топологий с тасующей подстановкой: *а* — совершенная; *б* — инверсная совершенная

Баттерфляй

Функция «*баттерфляй*» (butterfly) — «бабочка», была разработана в конце 60-х годов Рабинером и Гоулдом. Свое название она получила из-за того, что построенная в соответствии с ней сеть по конфигурации напоминает крылья бабочки (рис. 12.5).

Математически функция может быть записана в виде:

$$B_i(x_{n-1} \ldots x_{i+1}x_ix_{i-1} \ldots x_0) = x_{n-1} \ldots x_{i+1}x_0x_{i-1} \ldots x_1x_i, \quad 0 \le i \le n-1.$$

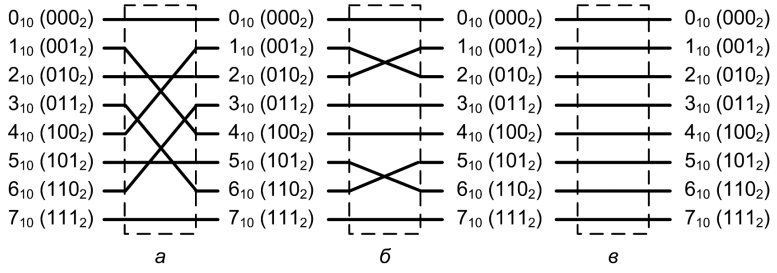


Рис. 12.5. Примеры топологии «баттерфляй»: *а* — при $i = 2$; *б* — при $i = 1$; *в* — при $i = 0$

Двоичное представление узла-получателя получается путем взаимной перестановки в адресе узла-источника битов с индексами i и 0. Хотя *баттерфляй*-функция используется в основном при объединении ступеней в сетях с динамической многоступенчатой топологией, известны также и «чистые» *баттерфляй*-сети.

Реверсирование битов

Как следует из названия, функция сводится к перестановке битов адреса в обратном порядке:

$$R(x_{n-1}x_{n-2} \ldots x_1x_0) = x_0x_1 \ldots x_{n-2}x_{n-1}.$$

Соответствующая топология для $n = 4$ показана на рис. 12.6. Хотя для значений $n \leqslant 3$ топология реверсирования битов совпадает с топологией «баттерфляй», при больших значениях n различия становятся очевидными.

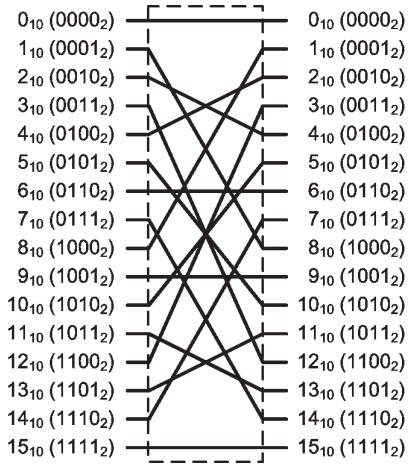


Рис. 12.6. Пример топологии на основе формулы реверсирования битов

Базисная линия

Функция маршрутизации типа *базисной линии* определяется соотношением:

$$L_i(x_{n-1} \ldots x_{i+1}x_ix_{i-1} \ldots x_1x_0) = x_{n-1} \ldots x_{i+1}x_0x_i \ldots x_1, \quad 0 \le i \le n-1$$

и сводится к циклическому сдвигу $i + 1$ младших цифр адреса узла-источника на одну позицию вправо. Соответствующие топологии для различных значений i показаны на рис. 12.7.

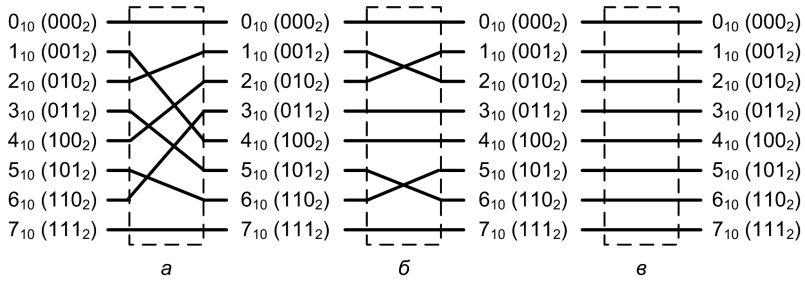


Рис. 12.7. Примеры топологии базисной линии: *а* — при $i = 2$; *б* — при $i = 1$; *в* — при $i = 0$

Статические топологии

К статическим топологиям относят такие, где между двумя узлами возможен только один прямой фиксированный путь, то есть статические топологии не

предполагают наличия в сети коммутирующих устройств. Если же такие устройства имеются, то используются они только перед выполнением конкретной задачи, а в процессе всего времени вычислений топология остается неизменной.

Из возможных показателей классификации статических сетей чаще всего выбирают их размерность. С этих позиций различают:

- одномерные топологии (линейный массив);
- двумерные топологии (кольцо, звезда, дерево, решетка, систолический массив);
- трехмерные топологии (полносвязная топология, хордальное кольцо);
- гиперкубическую топологию.

Ниже рассматриваются основные виды статических топологий.

Линейная топология

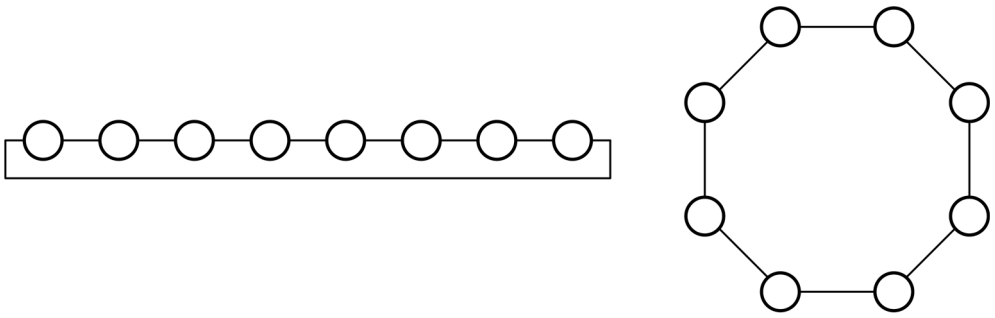
В *линейной топологии* узлы сети образуют одномерный массив и соединены в *цепочку* (рис. 12.8). Линейная топология характеризуется следующими параметрами: $D = N - 1$; $d = 1$ (для крайних узлов) и $d = 2$ (для всех остальных узлов); $I = N - 1$; $B = 1$.

Рис. 12.8. Линейная топология

Линейная топология не обладает свойством полной симметричности, поскольку узлы на концах цепочки имеют только одну коммуникационную линию, то есть их степень равна 1, в то время как степень остальных узлов равна 2. Время пересылки сообщения зависит от расстояния между узлами, а отказ одного из них способен привести к невозможности пересылки сообщения. По этой причине в линейных сетях используют отказоустойчивые узлы, которые при отказе изолируют себя от сети, позволяя сообщению миновать неисправный узел. Данный вид топологии наибольшее распространение нашел в системах класса SIMD.

Кольцевые топологии

Стандартная *кольцевая топология* представляет собой линейную цепочку, концы которой соединены между собой (рис. 12.9). В зависимости от числа каналов между соседними узлами (один или два) различают *однонаправленные* и *двунаправленные* кольца. Кольцевая топология характеризуется следующими параметрами: $D = \min\left[\frac{N}{2}\right]$ (для двунаправленного кольца) и $D = N - 1$ (для однонаправленного кольца); $d = 2$; $I = N$; $B = 2$. Введение одного дополнительного канала вдвое уменьшает диаметр и увеличивает ширину бисекции. Кольцевая топология, по сравнению с линейной, менее популярна, поскольку добавление или удаление узла требует демонтажа сети. Тем не менее она нашла применение в ряде ВС, например в вычислительных системах KSR-1 и SCI.



а *б*

Рис. 12.9. Кольцевая топология: *а* — стандартное представление; *б* — альтернативное представление

Один из способов уменьшения диаметра и увеличения ширины бисекции кольцевой сети — добавление линий связи в виде хорд, соединяющих определенные узлы кольца. Подобная топология носит название *хордальной*. Если хорды соединяют узлы с шагом 1 или $\frac{N}{2}-1$, диаметр сети уменьшается вдвое. На рис. 12.10 показана *хордальная кольцевая сеть* с шагом 3.

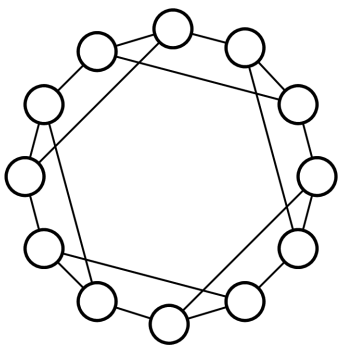


Рис. 12.10. Кольцевая хордальная топология

Введение хорд приводит к возрастанию степени узлов, а значит, и их сложности. В то же время появляется возможность отключения неисправного узла с сохранением работоспособности сети. Примером использования хордальной топологии может служить вычислительная система ASP.

Звездообразная топология

Звездообразная сеть объединяет множество узлов с $d = 1$ посредством специализированного центрального узла — концентратора (рис. 12.11). Топология характеризуется такими параметрами: $D = 2$; $d = 1$ (для краевых узлов) и $d = N - 1$ (для узла-концентратора); $I = N - 1$; $B = 1$.

Звездообразная организация узлов и соединений редко используется для объединения процессоров многопроцессорной ВС, но хорошо работает, когда поток информации идет от нескольких вторичных узлов, соединенных с одним первичным узлом, например, при подключении терминалов. Общая пропускная способность сети обычно ограничивается быстродействием концентратора, аналогично тому, как сдерживающим элементом в одношинной топологии выступает шина. По производительности эти топологии также идентичны. Основное преимущество звездообразной схемы в том, что конструктивное исполнение узлов на концах сети может быть очень простым.

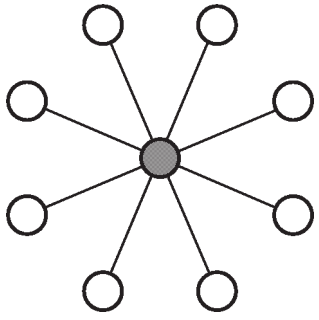


Рис. 12.11. Звездообразная топология

Древовидные топологии

Еще одним вариантом структуры сети является *древовидная топология* (рис. 12.12, *а*). Сеть строится по схеме так называемого строго двоичного дерева, где каждый узел более высокого уровня связан с двумя узлами следующего по порядку более низкого уровня. Узел, находящийся на более высоком уровне, принято называть *родительским*, а два подключенных к нему нижерасположенных узла — *дочерними*. В свою очередь, каждый дочерний узел выступает в качестве родительского для двух узлов следующего более низкого уровня. Отметим, что каждый узел связан только с двумя дочерними и одним родительским. Такую сеть можно охарактеризовать следующими параметрами: $D = 2\log_2\left(\frac{N+1}{2}\right)$; d = 1 (для краевых узлов), d = 2 (для корневого узла) и d = 3 (для остальных узлов); I = N – 1; B = 1. Диаметр для двоичного дерева возрастает пропорционально лишь логарифму числа узлов, в то время как степень узла остается постоянной. Сеть с такой топологией хорошо масштабируется. Основной недостаток топологии — малая ширина бисекции, что предполагает ограниченную полосу пропускания.

Топология двоичного дерева была использована в мультипроцессорной системе DADO из 1023 узлов, разработанной в Колумбийском университете.

При больших объемах пересылок между несмежными узлами древовидная топология оказывается недостаточно эффективной, поскольку сообщения должны проходить через один или несколько промежуточных звеньев. Очевидно, что на более

высоких уровнях сети вероятность затора из-за недостаточно высокой пропускной способности линий связи выше. Этот недостаток устраняют с помощью топологии, называемой «*толстым*» деревом (рис. 12.12, *б*).

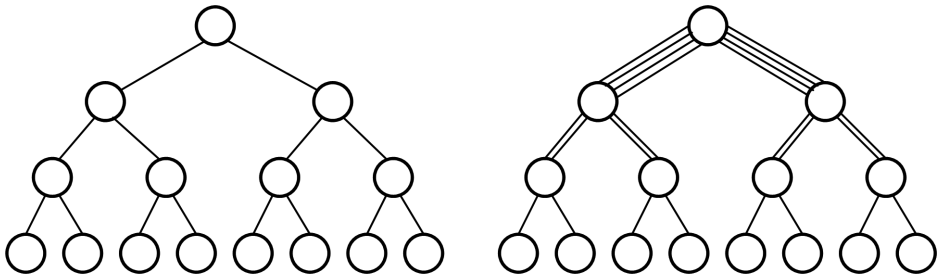


Рис. 12.12. Древовидная топология: *а* — стандартное дерево; *б* — «толстое» дерево

Идея «толстого» дерева состоит в увеличении пропускной способности коммуникационных линий на прикорневых уровнях сети. С этой целью на верхних уровнях сети родительские и дочерние узлы связывают не одним, а несколькими каналами, причем чем выше уровень, тем больше число каналов. На рисунке это отображено в виде множественных линий между узлами верхних уровней. Топология «толстого» дерева реализована в вычислительной системе СМ-5.

Решетчатые топологии

Поскольку значительная часть научно-технических задач связана с обработкой массивов, желательно отражать эту специфику в топологии соответствующих ВС. Такие топологии относят к *решетчатым* (mesh), а их конфигурация определяется видом и размерностью массива.

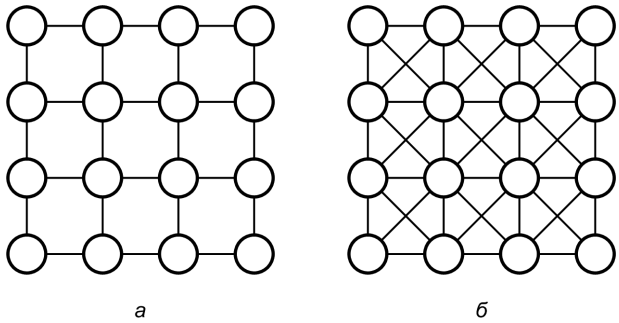


Рис. 12.13. Топологии плоской решетки: *а* — с 4 связями; *б* — с 8 связями

Простейшими примерами для одномерных массивов могут служить цепочка и кольцо. Для двухмерных массивов данных наиболее подходит топология плоской

прямоугольной матрицы узлов, каждый из которых соединен с ближайшим соседом (рис. 12.13, *а*). Такая сеть размерности $n \times n$ $(n = \sqrt{N})$ имеет следующие характеристики: $D = 2(n - 1)$; $d = 2$ (для угловых узлов), $d = 3$ (для краевых узлов) и $d = 4$ (для остальных узлов); $I = 2(N - n)$; $B = n$.

Возможны и иные формы плоской двухмерной решетчатой топологии (рис. 12.13, *б*). Этот вариант характеризуется следующими параметрами: $D = n$; $d = 3$ (для угловых узлов), $d = 5$ (для краевых узлов) и $d = 8$ (для остальных узлов); $I = 2(2N - 3n + 1)$; $B = 2n$. Введение диагональных связей позволяет сократить диаметр и увеличить ширину бисекции сети, но при этом возрастает степень узлов.

Двухмерные решетчатые топологии встречаются в вычислительных системах класса SIMD и транспьютерных ВС. Они были использованы в системах Intel Paragon и Intel Touchstone Delta.

Если провести операцию *свертывания* (wraparound) плоской матрицы, соединив информационными трактами одноименные узлы верхней и нижней строк плоской матрицы (рис. 12.14, *а*) или левого и правого столбцов (рис. 12.14, *б*), так чтобы столбцы (строки) образовывали кольцо, то из плоской конструкции получаем топологию типа цилиндра. В топологии цилиндра каждый ряд (или столбец) матрицы представляет собой кольцо. Если одновременно произвести свертывание плоской матрицы в обоих направлениях, получим тороидальную топологию сети (рис. 12.14, *в*). Двухмерный тор на базе решетки $n \times n$ обладает следующими параметрами: $D = 2\min\left[\frac{n}{2}\right]$; $d = 4$; $I = 2N$; $B = 2n$. Сеть с топологией тора используется в системах AP3000 компании Fujitsu.

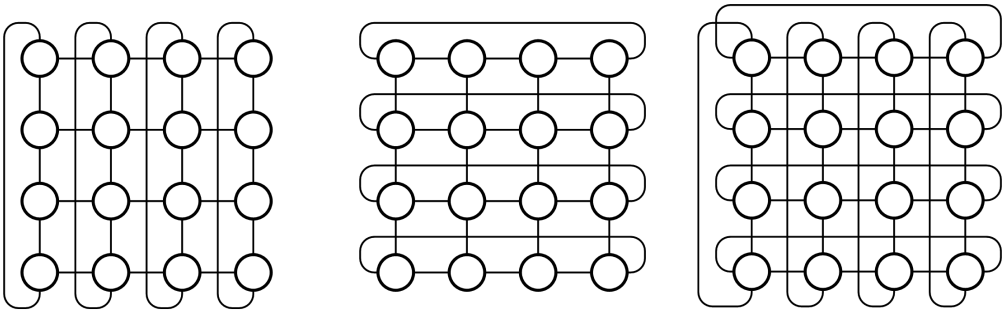


Рис. 12.14. Двухмерные решетчатые топологии с применением операции свертывания: *а* — по столбцам (цилиндр); *б* — по строкам (цилиндр); *в* — по столбцам и строкам (двухмерный тор)

Помимо свертывания к плоской решетке может быть применена операция *скручивания* (twisting). Суть этой операции состоит в том, что все узлы объединяются в разомкнутую или замкнутую спираль, то есть узлы, расположенные с противоположных краев плоской решетки, соединяются с некоторым сдвигом. Это приводит к топологиям *витого цилиндра* и *витого тора*. Если горизонтальные петли объеди-

нены в виде спирали, образуется так называемая сеть типа ILLIAC. На рис. 12.15, *а* показана подобная конфигурация сети, соответствующая хордальной сети четвертого порядка и характеризуемая следующими значениями: $D = n - 1$; $d = 4$; $I = 2N$; $B = 2n$. Если же столбцы узлов также объединены в виде спирали, получаем топологию, показанную на рис. 12.15, *б*. В этом случае имеем два больших кольца.

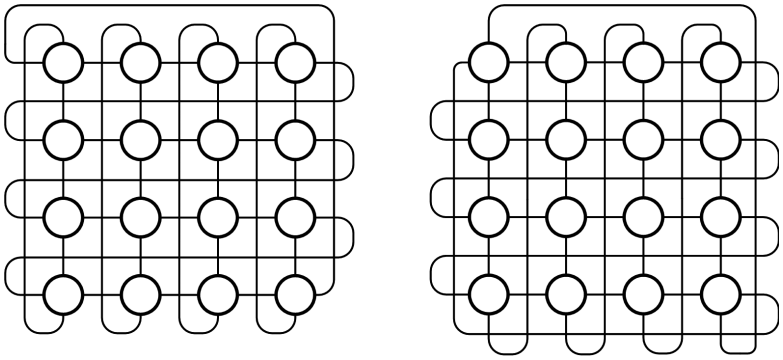


Рис. 12.15. Двухмерные решетчатые топологии с применением операции скручивания: *а* — ILLIAC; *б* — витой тор

Следует упомянуть и трехмерные сети. Один из вариантов, реализованный в архитектуре ВС Cray T3D, представляет собой трехмерный тор, образованный объединением процессоров в кольца по трем координатам: *x*, *y* и *z*.

Примерами ВС, где реализованы различные варианты решетчатых топологий, могут служить: ILLIAC IV, MPP, DAP, CM-2, Paragon и др.

Полносвязная топология

В *полносвязной топологии* (рис. 12.16), известной также под названием топологии «*максимальной группировки*» или «*топологии клика*» (clique — полный подграф), каждый узел напрямую соединен со всеми остальными узлами сети. Сеть, состоящая из N узлов, имеет следующие параметры: $D = 1$; $d = N - 1$; $I = \frac{N(N-1)}{2}$; $B = \frac{N^2}{4}$.

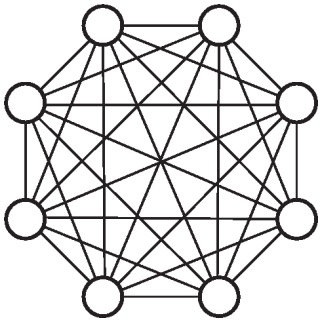


Рис. 12.16. Полносвязная топология

Если размер сети велик, топология становится дорогостоящей и трудно реализуемой. Более того, топология максимальной группировки не дает существенного улучшения производительности, поскольку каждая операция пересылки требует, чтобы узел проанализировал состояние всех своих $N - 1$ входов. Для ускорения этой операции необходимо, чтобы все входы анализировались параллельно, что, в свою очередь, усложняет конструкцию узлов.

Топология гиперкуба

При объединении параллельных процессоров весьма популярна топология гиперкуба, показанная на рис. 12.17. Линия, соединяющая два узла (рис. 12.17, *а*), определяет одномерный гиперкуб. Квадрат, образованный четырьмя узлами (рис. 12.17, *б*), — двумерный гиперкуб, а куб из 8 узлов (рис. 12.17, *в*) — трехмерный гиперкуб и т. д. Из этого ряда следует алгоритм получения n-мерного гиперкуба: начинаем с $(n - 1)$-мерного гиперкуба, делаем его идентичную копию, а затем добавляем связи между каждым узлом исходного гиперкуба и одноименным узлом копии. Гиперкуб размерности $n = \log_2 N$ имеет следующие характеристики: $D = n$; $d = n$; $I = \frac{nN}{2}$; $B = \frac{N}{2}$. Увеличение размерности гиперкуба на единицу ведет к удвоению числа его узлов, увеличению степени узлов и диаметра сети на единицу.

Обмен сообщениями в гиперкубе базируется на двоичном представлении номеров узлов. Нумерация узлов производится так, что для любой пары смежных узлов двоичное представление номеров этих узлов отличается только в одной позиции. В силу сказанного, узлы 0010 и 0110 — соседи, а узлы 0110 и 0101 таковыми не являются. Простейший способ нумерации узлов при создании n-мерного гиперкуба из двух $(n - 1)$-мерных показан на рис. 12.17, *г*. При копировании $(n - 1)$-мерного гиперкуба и соединении его с исходным $(n - 1)$-мерным гиперкубом необходимо, чтобы соединяемые узлы имели одинаковые номера. Далее к номерам узлов исходного гиперкуба слева добавляется бит, равный 0, а к номерам узлов копии — единичный бит.

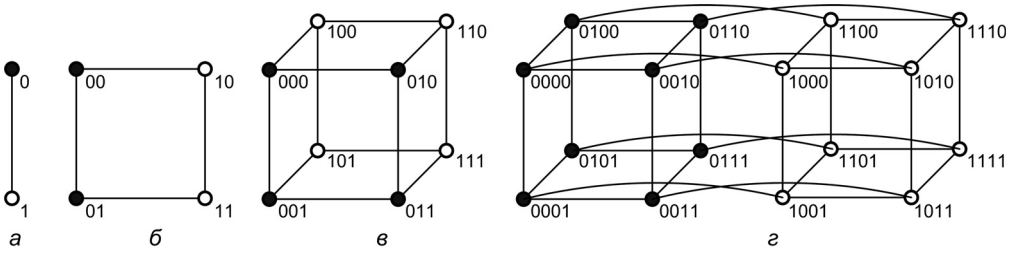


Рис. 12.17. Топология гиперкуба: *а* — одномерная; *б* — двумерная; *в* — трехмерная; *г* — четырехмерная

Номера узлов являются основой маршрутизации сообщений в гиперкубе. Такие номера в n-мерном гиперкубе состоят из n битов, а пересылка сообщения из узла A в узел B выполняется за n шагов. На каждом шаге узел может либо сохранить сообщение и не пересылать его дальше до следующего шага, либо отправить его дальше

по одной из линий. На шаге i узел, хранящий сообщение, сравнивает i-й бит своего собственного номера с i-м битом номера узла назначения. Если они совпадают, продвижение сообщения приостанавливается до следующего шага, в противном случае сообщение передается вдоль линии i-го измерения. Линией i-го измерения считается та, которая была добавлена на этапе построения i-мерного гиперкуба из двух $(i-1)$-мерных.

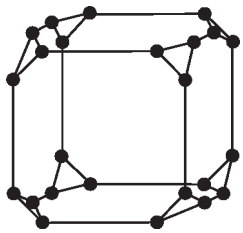


Рис. 12.18. Куб из циклически соединенных узлов третьего порядка

Создание гиперкуба при большом числе процессоров требует увеличения степени узлов, что сопряжено с большими техническими проблемами. Компромиссное решение, несколько увеличивающее диаметр сети при сохранении базовой структуры, представляет собой *куб из циклически соединенных узлов* (рис. 12.18). Здесь степень узла равна трем при любом размере сети.

Данные по рассмотренным статическим топологиям сведены в табл. 12.1.

Таблица 12.1. Характеристики сетей со статической топологией

Топология	Диаметр	Степень узла	Число связей	Ширина бисекции
Полносвязная	1	$N-1$	$\frac{N(N-1)}{2}$	$\frac{N^2}{4}$
Звезда	2	$N-1$	$N-1$	1
Двоичное дерево	$2\log_2\left(\frac{N+1}{2}\right)$	3	$N-1$	1
Линейный массив	$N-1$	2	$N-1$	1
Кольцо	$\left\lfloor\frac{N}{2}\right\rfloor$	2	N	2
Двухмерная решетка	$2(\sqrt{N}-1)$	4	$2(N-\sqrt{N})$	$\sqrt{N}$
Двухмерный тор	$2\left\lfloor\frac{\sqrt{N}}{2}\right\rfloor$	4	$2N$	$2\sqrt{N}$
Гиперкуб	$\log_2 N$	$\log_2 N$	$\frac{N\log_2 N}{2}$	$\frac{N}{2}$

Динамические топологии

В *динамической топологии сети* соединение узлов обеспечивается электронными ключами, варьируя установки которых можно менять топологию сети. В отличие от ранее рассмотренных топологий, где роль узлов играют сами объекты информационного обмена, в узлах динамических сетей располагаются коммутирующие элементы, а устройства, обменивающиеся сообщениями (терминалы), подключаются к входам и выходам этой сети. В роли терминалов могут выступать процессоры или процессоры и банки памяти.

Можно выделить два больших класса сетей с динамической топологией: *сети на основе шины* и *сети на основе коммутаторов*.

Одношинная топология

Сети на основе шины — наиболее простой и дешевый вид динамических сетей. Рисунок 12.19 иллюстрирует систему на базе *одношинной топологии*.

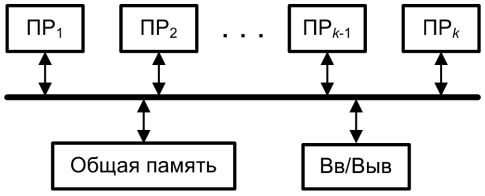


Рис. 12.19. Пример системы с одной шиной

В общем случае такая система состоит из k процессоров, каждый из которых подключен к общей совместно используемой шине. Все процессоры взаимодействуют через общую память. Узлы при одношинной топологии имеют степень 1 ($d = 1$). В каждый момент времени обмен сообщениями может вести только одна пара узлов, то есть на период передачи сообщения шину можно рассматривать как сеть, состоящую из двух узлов, в силу чего ее диаметр всегда равен единице ($D = 1$). Также единице равна и ширина бисекции (B), поскольку топология допускает одновременную передачу только одного сообщения. Одношинная конфигурация может быть полезной, когда число узлов невелико, то есть когда трафик шины мал по сравнению с ее пропускной способностью. Размер подобных систем изменяется в диапазоне от 2 до 50 процессоров. Одношинную архитектуру часто используют для объединения нескольких узлов в группу (кластер), после чего из таких кластеров образуют сеть на базе других видов топологии.

Многошинная топология

Естественным развитием идеи одношинной топологии является *многошинная топология*. В вычислительной системе с многошинной топологией для объединения множества процессоров и множества модулей памяти используются несколько параллельных шин. Процессоры ВС, как правило, подключаются к каждой из шин, а при подключении модулей памяти может использоваться одна из четырех схем:

- с подключением ко всем шинам;
- с подключением к одной из шин;
- с подключением к части шин;
- с подключением по классам.

Перечисленные схемы иллюстрирует рис. 12.20. В варианте (*а*) все банки памяти подключаются к каждой из шин. В схеме (*б*) каждый банк памяти подключен лишь к определенной шине. В случае (*в*) каждый банк памяти подключен к определенному подмножеству шин. Наконец, в варианте (*г*) банки памяти группируются в несколько классов, а каждый класс, в свою очередь, имеет соединение с определенным подмножеством шин.

Такая топология вполне пригодна для высокопроизводительных ВС. Диаметр сети по-прежнему равен 1, в то время как пропускная способность возрастает пропорционально числу шин. По сравнению с одношинной архитектурой управление сетью с несколькими шинами сложнее из-за необходимости предотвращения конфликтов, возникающих, когда в парах узлов, обменивающихся по разным шинам, присутствует общий узел. Кроме того, с увеличением степени узлов сложнее становится их техническая реализация.

Блокирующие, неблокирующие и реконфигурируемые топологии

В динамических сетях на основе коммутаторов тракты между терминальными узлами формируются с использованием коммутирующих устройств. Минимальным требованием к сети с коммутацией является поддержка соединения любого входа с любым выходом. Для этого в сети с n входами и n выходами система ключей обязана предоставить $n!$ вариантов коммутации входов и выходов. Проблема усложняется, когда сеть должна обеспечивать одновременную передачу данных между многими парами терминальных узлов, причем так, чтобы не возникали конфликты (блокировки) из-за передачи данных через одни и те же коммутирующие элементы в одно и то же время. Подобные топологии должны поддерживать n^n вариантов перестановок. С этих позиций все топологии сетей с коммутацией разделяются на три типа: блокирующие, неблокирующие и реконфигурируемые.

В *блокирующих сетях* установление соединения между свободным входом и свободным выходом возможно не всегда, поскольку это может вызвать конфликт с другим уже установленным соединением из-за наличия в этих соединениях общих коммутаторов. Обычно для минимизации общего числа коммутаторов в сети предполагают лишь единственный путь между каждой парой вход/выход. В то же время для уменьшения числа конфликтов и повышения отказоустойчивости в сети могут предусматриваться множественные пути. Такие блокирующие сети известны как *сети с обходными путями*.

В *неблокирующих сетях* любой входной порт может быть подключен к любому свободному выходу без влияния на уже существующие соединения. В рамках этой группы различают сети строго неблокирующие и неблокирующие в широком смысле.

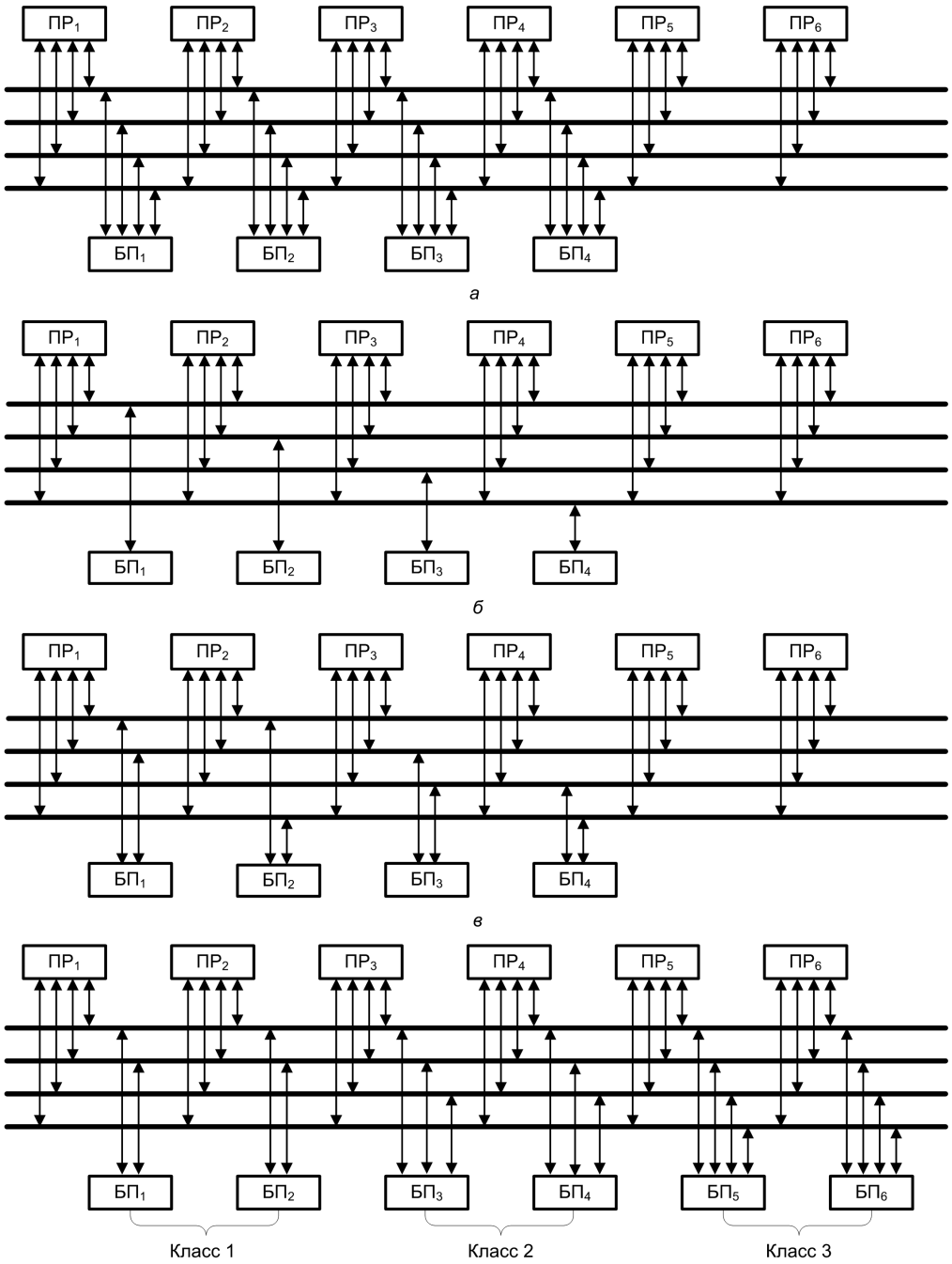


Рис. 12.20. Множественные шины с подключением банков памяти: *а* — ко всем шинам; *б* — к одной из шин; *в* — к части шин; *г* — по классам системы с одной шиной

В *строго неблокирующих сетях* возникновение блокировок принципиально невозможно в силу примененной топологии. *Неблокирующими в широком смысле* называют топологии, в которых конфликты при любых соединениях не возникают только при соблюдении определенного алгоритма маршрутизации. В любом случае такие сети, как правило, являются многоступенчатыми. В многоступенчатых динамических сетях коммутаторы группируются в так называемые *ступени коммутации*. Наличие более чем одной ступени коммутации позволяет обеспечить множественность путей между любыми парами входов и выходов.

В *реконфигурируемых сетях* возможна такая установка коммутаторов сети, при которой допустима одновременная передача сообщений между множеством пар входных и выходных узлов без взаимного блокирования сообщений. Здесь, однако, следует учитывать, что все тракты передачи сообщений должны формироваться одновременно. Если новый тракт формируется при уже функционирующих других трактах, неблокируемость без реконфигурации этих функционирующих трактов не гарантируется. В реконфигурируемых сетях обеспечивается множественность путей между каждой парой вход/выход, однако количество таких путей по сравнению с неблокирующими сетями меньше, но при этом меньше и стоимость сети. Сети данного типа были предложены для построения матричных ВС. В других типах многопроцессорных ВС реконфигурация оказывается более сложной, поскольку процессоры обращаются к сети асинхронно, и в этом случае рассматриваемые сети ведут себя как блокирующие.

Топология полносвязной коммутационной матрицы («кроссбар»)

Топология полносвязной коммутационной матрицы (кроссбар) представляет собой пример одноступенчатой динамической сети. Смысл, вкладываемый в понятие «полносвязная матрица», заключается в том, что любой входной порт может быть связан с любым выходным портом. Не совсем официальный термин «кроссбар», который будет применяться в дальнейшем для обозначения данной топологии, берет свое начало с самых первых телефонных коммутаторов, где с помощью штекеров входные линии замыкались на выходные, образуя соединение.

Коммутатор типа кроссбар представляет собой матрицу, строки которой образованы проводниками, связанными с входами, а столбцы — с выходами. На пересечении линий и столбцов матрицы находятся управляемые переключатели, которые могут либо замыкать соединение строки на столбец, либо, наоборот, размыкать его. Для управления переключателями, находящимися на пересечении строк и столбцов такой коммутационной матрицы, кроссбар должен иметь контроллер, который управляет состоянием переключателей на основе анализа информации об адресе назначения.

Кроссбар $n \times l$ способен соединить n входных и l выходных терминальных узлов (процессоров, банков памяти и т. п.), причем так, что обмен информацией одновременно могут вести $\min(n, l)$ пар терминальных узлов, и конфликты при этом не возникают. Новое соединение может быть установлено в любой момент при условии, что входной и выходной порты свободны.

Главное достоинство топологии состоит том, что сеть получается неблокирующей и обеспечивает меньшую задержку в передаче сообщений по сравнению с другими топологиями, поскольку любой путь содержит только один ключ. Тем не менее из-за значительного числа ключей в кроссбаре ($n \times l$) использование такой топологии в больших сетях становится непрактичным, хотя это достаточно хороший выбор для малых сетей.

Кроссбары могут использоваться в коммутаторах, ориентированных как на коммутацию пакетов, так и на коммутацию соединений. Они пригодны для применения в синхронных и асинхронных сетях.

Сети кроссбар традиционно используются в небольших ВС с разделяемой памятью, где все процессоры могут одновременно обращаться к банкам памяти при условии, что каждый процессор работает со своим банком памяти. В качестве примера рассмотрим сеть кроссбар, приведенную на рис. 12.21. Сеть содержит переключающий элемент в каждой из 64 точек пересечения горизонтальных и вертикальных линий. На рисунке показано состояние кроссбара при одновременном соединении каждого из 8 процессоров ($ПР_i$) со своим банком памяти ($БП_{8-i+1}$). Показаны также два возможных состояния переключающего элемента — это прямое соединение (см. 12.21, *а*) и диагональное соединение (см. 12.21, *б*).

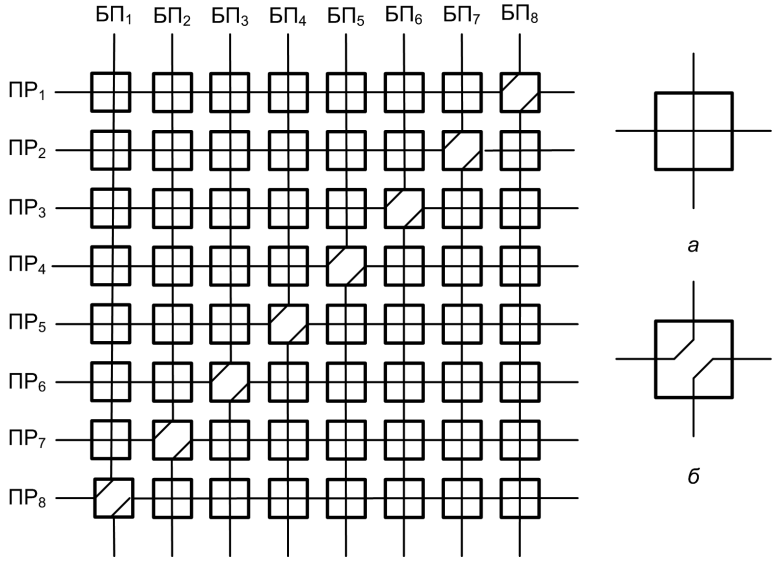


Рис. 12.21. Сеть кроссбар 8 × 8: *а* — прямое соединение в переключающем элементе; *б* — диагональное соединение в переключающем элементе

Когда два или более процессоров соперничают за один и тот же банк памяти, срабатывает схема арбитража, разрешающая доступ к банку лишь одному из процессоров, в то время как остальные процессоры вынуждены ожидать. В целом же схема арбитража в кроссбаре может быть менее сложной, чем в случае шины, поскольку конфликты в кроссбаре являются скорее исключением, чем правилом.

При $l = n$ кроссбар называют *полным*. Полный кроссбар на n входов и n выходов содержит n^2 ключей. Диаметр кроссбара равен 1, ширина бисекции — —. Большие кроссбары можно реализовать путем разбиения на меньшие. Так, полный кроссбар $n \times n$ можно построить из $\frac{n}{k} \times \frac{n}{k}$ кроссбаров размерности $k \times k$.

Топология используется для организации соединений в некоторых серийно выпускаемых вычислительных системах, например в вычислительной системе Earth Simulator фирмы NEC используется кроссбар 639×639.

Коммутирующие элементы сетей с динамической топологией

Поскольку последующие рассматриваемые топологии относятся к многоступенчатым, сначала необходимо определить типы коммутирующих элементов, применяемых в ступенях коммутации таких сетей. По этому признаку различают:

- сети на основе полносвязной коммутационной матрицы;
- сети на основе базового коммутирующего элемента.

В сетях, относящихся к первой группе, в качестве базового коммутирующего элемента используется кроссбар $n \times l$. Для второй категории роль коммутирующего элемента играет «полный кроссбар» 2×2. Потенциально такой коммутатор управляется четырехразрядным двоичным кодом и обеспечивает 16 вариантов коммутации, из которых полезными можно считать 12. На практике же обычно задействуют только четыре возможных состояния кроссбара 2×2, которые определяются двухразрядным управляющим кодом (рис. 12.22). Подобный кроссбар называют *базовым коммутирующим элементом* (БКЭ) или β-*элементом*. Первые два состояния БКЭ являются основными: в них входная информация может транслироваться на выходы прямо либо перекрестно. Два следующих состояния предназначены для широковещательного режима, когда сообщение от одного узла одновременно транслируется на все подключенные к нему прочие узлы. Широковещательный режим используется редко. Сигналы на переключение БКЭ в определенное состояние могут формироваться устройством управления сетью. В более сложном варианте β-элемента эти сигналы формируются внутри него, исходя из адресов узла-источника и узла-получателя, содержащихся в заголовке сообщения.

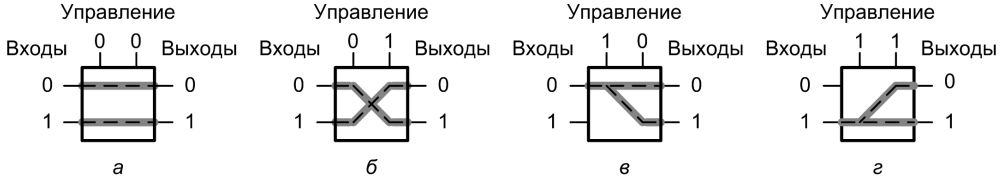


Рис. 12.22. Состояния β-элемента: *а* — прямое; *б* — перекрестное; *в* — широковещание с верхнего входа; *г* — широковещание с нижнего входа

Структура β-элемента показана на рис. 12.23.

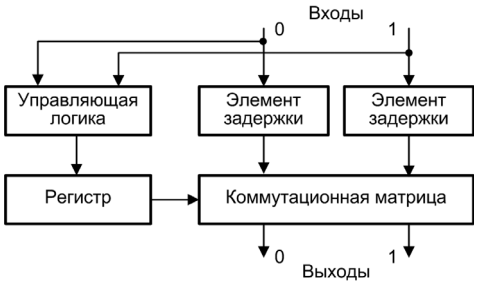


Рис. 12.23. Структура β-элемента

Выбор в пользу того или иного варианта коммутации входных сообщений (пакетов) осуществляется управляющей логикой β-элемента. Назначение этой схемы состоит в анализе заголовков пакетов и в принятии решения о требуемом состоянии коммутационной матрицы. Результат решения фиксируется в регистре. В зависимости от принятого решения коммутационная матрица может обеспечивать либо прямое соединение между входами и выходами, либо перекрестное (кроссоверное). Элементы задержки служат для синхронизации процессов принятия решения и пересылки пакетов с входов на выходы.

Сложность β-элемента находится в зависимости от логики принятия решения. В ряде архитектур БКЭ их состояние определяется только битом активности пакета. В иных архитектурах используются адреса источника и получателя данных, хранящиеся в заголовке пакета, что может потребовать поддержания в БКЭ специальных таблиц. Тем не менее во всех своих вариантах β-элементы достаточно просты, что позволяет реализовать их на базе интегральных микросхем.

Многоступенчатые динамические сети

Многоступенчатые динамические сети (МДС) разрабатывались как средство для устранения ограничений, свойственных одноступенчатым сетям, при сохранении стоимости сети в допустимых пределах. В частности, наличие дополнительных ступеней позволяет одновременно иметь несколько альтернативных трактов между любой парой узлов.

Многоступенчатые динамические сети соединяют входные устройства с выходными посредством нескольких ступеней коммутации, построенных из базовых коммутирующих элементов, либо кроссбаров большей размерности. Количество ступеней и способ их соединения определяют возможности маршрутизации в сети.

Существует много способов соединения смежных ступеней МДС. На рис. 12.24, *а* показана обобщенная многоступенчатая сеть с n входами и l выходами. МДС состоит из g ступеней $G_0 - G_{g-1}$. Как показано на рис. 12.24, *б*, каждая ступень, скажем G_i, содержит w_i коммутаторов размерности $a_{i,j} \times b_{i,j}$, где $1 \le j \le w_i$. Таким образом, ступень G_i имеет p_i входов и q_i выходов, где

$$p_i = \sum_{j=1}^{w_i} a_{i,j} \text{ и } \qquad q_i = \sum_{j=1}^{w_i} b_{i,j}.$$

Соединение между двумя смежными ступенями G_{i-1} и G_i, обозначенное C_i, определяется функцией маршрутизации для $p_i = q_{i-1}$ линий связи, где $p_0 = n$ и $q_{g-1} = l$. Таким образом, МДС может быть представлена в виде

$$C_0(N)G_0(w_0)C_1(p_1)G_1(w_1)...G_{g-1}(w_{g-1})C_g(M).$$

Функция маршрутизации $C_i(p_i)$ определяет, каким образом p_i линий должны быть использованы между $q_{i-1} = p_i$ выходами ступени G_{i-1} и p_i входами ступени G_i. Различные функции маршрутизации определяют различие в характеристиках и топологических свойствах МДС.

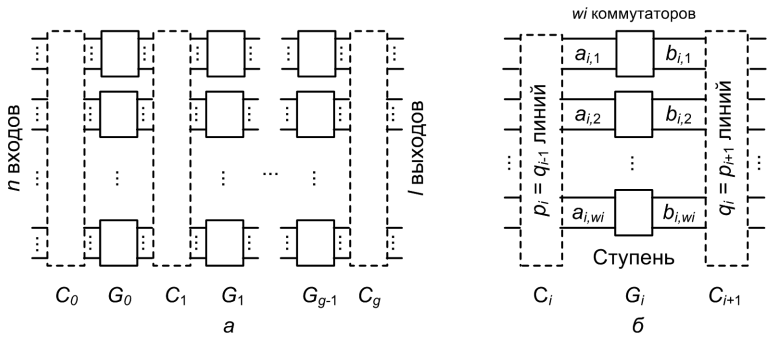


Рис. 12.24. Многоступенчатая сеть с *n* входами, *l* выходами и *g* ступенями: *а* — общая структура; *б* — детальное представление ступени G_i

На практике все коммутаторы должны быть идентичными, чтобы снизить стоимость разработки. Если входы и выходы сети коммутирующих элементов разделены, сеть называют *двухсторонней*. При совмещенных входах и выходах сеть является *односторонней*.

В зависимости от типа каналов связи и коммутаторов динамические многоступенчатые сети можно разделить на два класса: *однонаправленные* или *двунаправленные*. В однонаправленных сетях используются однонаправленные каналы и коммутаторы. В двунаправленных сетях каналы и коммутаторы — двунаправленные.

Блокирующие многоступенчатые сети

Среди блокирующих многоступенчатых сетей наибольшее распространение получили баньяноподобные сети [83]. *Баньян-сети* образуют обширное семейство МДС, в которых между любым входом и любым выходом возможен только единственный путь [112]. Название свое сети получили из-за того, что их конфигурация напоминает воздушные корни дерева баньян (индийской смоковницы).

Среди баньян-сетей наибольшее распространение получили так называемые сети «Дельта», предложенные Пателом в 1981 году [127]. Дельта-сеть на *n* входов и *n* выходов имеет $\log_2 n$ ступеней коммутации, каждая из которых состоит из $\frac{n}{2}$ базовых коммутирующих элементов. На рис. 12.25 показаны несколько вариантов дельта-сетей.

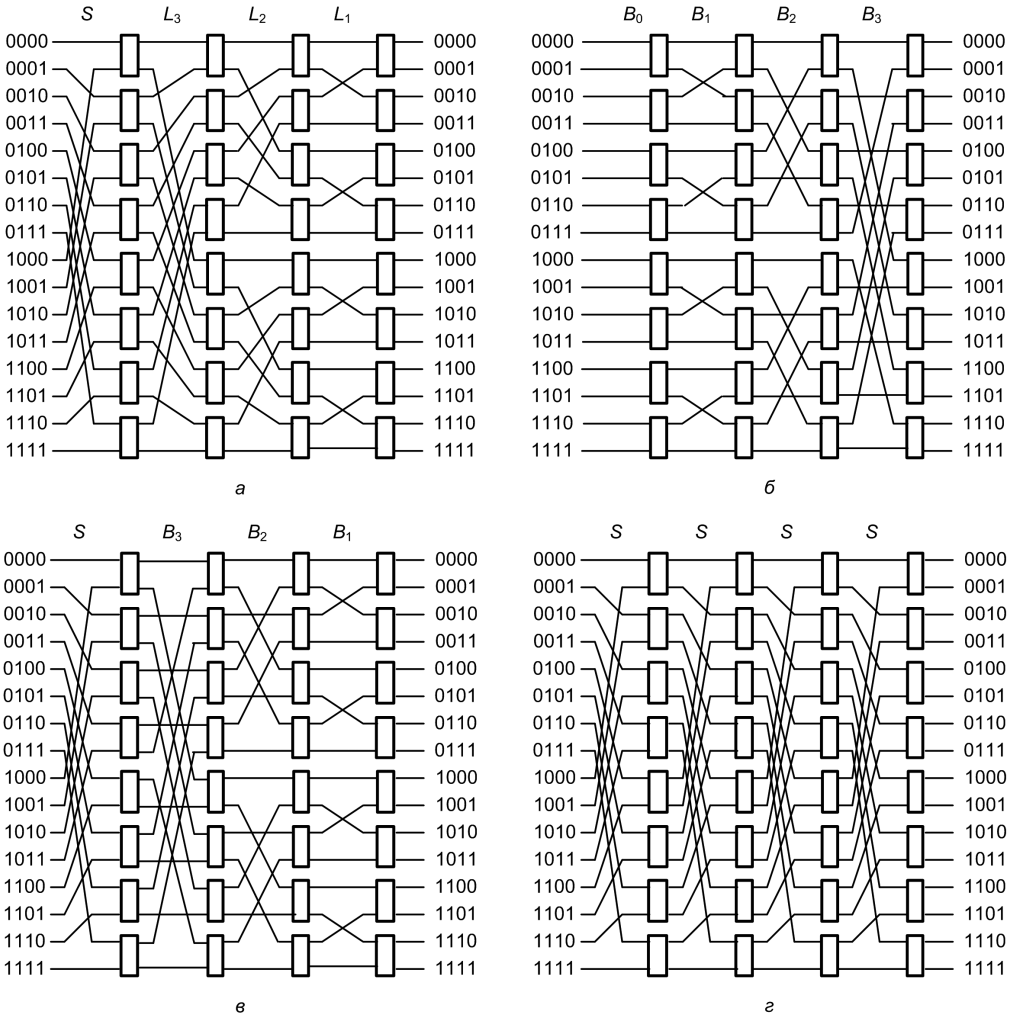


Рис. 12.25. Примеры сетей класса Дельта размерности 16 × 16 с топологией: *а* — базисной линии; *б* — баттерфляй; *в* — обобщенного куба; *г* — Омега

Представленные дельта-топологии [63, 106, 145] различаются тем, какие функции маршрутизации использованы между ступенями сети и на ее входе. Эти функции указаны в верхней части каждой схемы. Отметим, что по отношению к произвольному трафику все сети обеспечивают эквивалентную производительность.

Существенным достоинством рассматриваемых сетей, определившим их популярность, является свойство *самомаршрутизации*. Чтобы доставить сообщение к узлу-получателю, используется адрес этого узла, содержащийся в заголовке передаваемого пакета. Этот адрес не только определяет маршрут сообщения к нужному узлу, но и используется для управления прохождением сообщения по этому маршруту. Число битов в двоичном представлении адреса равно числу ступеней сети, причем

каждый бит соответствует определенной ступени: старший бит — нулевой (левой) ступени, младший бит — последней (правой) ступени.

Каждый БКЭ, куда попадает пакет, просматривает один бит адреса (соответствующий ступени сети, где этот БКЭ расположен), и в зависимости от его значения направляет сообщение на верхний или нижний выход. Если значение бита равно 0, то сообщение пропускается через верхний выход БКЭ, а при единичном значении — через нижний.

Проиллюстрируем этот алгоритм на примере сети «Омега» 8 × 8 (рис. 12.26, *а*). На рис. 12.26, *а* показан процесс маршрутизации сообщения с входного терминала 2_{10} (010_2) на выходной терминал с номером 6_{10} (110_2). Старший бит адреса назначения управляет коммутаторами нулевой ступени, средний бит — первой ступени, младший бит — второй ступени.

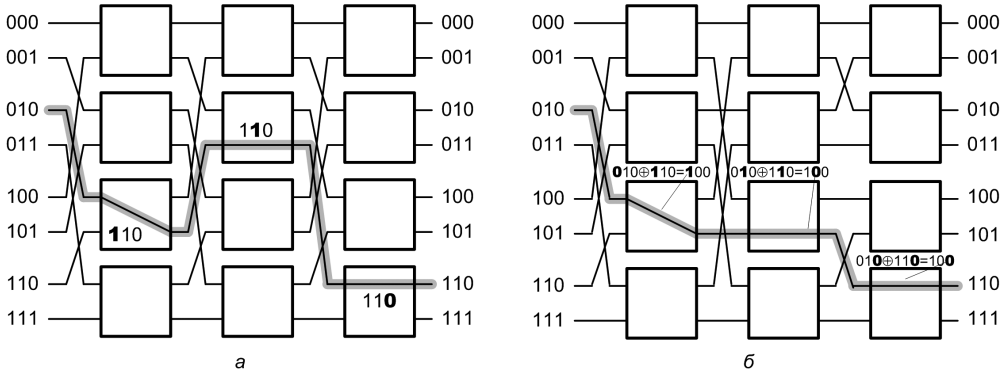


Рис. 12.26. Самомаршрутизация в дельта-сетях размерности 8 × 8: *а* — по адресу получателя; *б* — по адресам источника и получателя

Возможен и иной алгоритм самомаршрутизации, но в сообщении помимо адреса получателя необходимо передавать и адрес источника. Эти адреса в двоичном представлении имеют длину m битов (m — число ступеней). Самомаршрутизация реализуется за счет следующего алгоритма установки β-элементов сети. Состояние, в которое переключается БКЭ на i-й ступени, определяется с помощью операции сложения по модулю 2 значений i-го бита в адресах входного ($a_{m-1} \ldots a_0$) и выходного ($b_{m-1} \ldots b_0$) терминальных узлов. Если $a_i \oplus b_i = 0$, то БКЭ, расположенный на $(m - i - 1)$-й ступени сети, обеспечивает прямую связь входа с выходом, а при $a_i \oplus b_i = 1$ — перекрестное соединение. На рис. 12.26, *б* показан процесс прохождения сообщения по сети с топологией обобщенного куба размерности 8 × 8 от входного терминала 2_{10} (010_2) к выходному терминалу 6_{10} (110_2).

В то же время не следует забывать, что рассматриваемый класс топологий относится к блокирующим. Количество соединений, обеспечиваемых дельта-сетью, равно $n^{\frac{n}{2}}$, что гораздо меньше, чем $n!$, то есть топология также является блокирующей. Так, при $n = 8$ процент комбинаций, возможных в омега-сети, по отношению

к потенциально допустимому числу комбинаций составляет $\frac{8^4}{8!} = \frac{4096}{40320} = 0{,}1016$, или 10,16%. Блокирование в омега-сети иллюстрирует рис. 12.27, *а,* где показана ситуация одновременной передачи сообщений с входного терминала 0_{10} (000_2) на выходной терминал 1_{10} (001_2) и с входного терминала 4_{10} (100_2) на выходной терминал 2_{10} (010_2). Как видно, данная ситуация предполагает такую коммутацию в левом верхнем БКЭ, которая для базовых коммутирующих элементов невозможна. Аналогичная ситуация возникает в другом варианте дельта-сети (рис. 12.27, *б*) при попытке одновременной передачи пакетов с входного терминала 1_{10} (001_2) на выходной терминал 5_{10} (101_2) и с входного терминала 3_{10} (011_2) на выходной терминал 4_{10} (100_2).

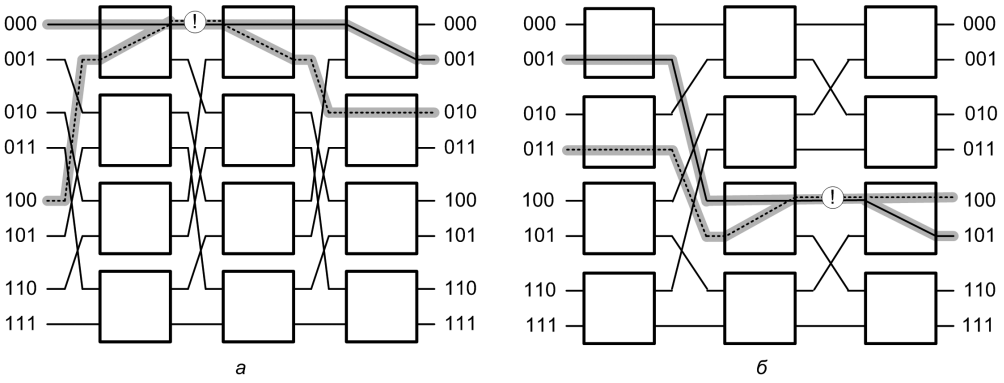


Рис. 12.27. Иллюстрация блокировки в дельта-сетях размерности 8 × 8:
а — «Омега»; *б* — базисной линии

Топология «Баньян» весьма популярна из-за того, что коммутация обеспечивается простыми БКЭ, работающими с одинаковой скоростью, сообщения передаются параллельно. Кроме того, большие сети могут быть построены из стандартных модулей меньшего размера.

Неблокирующие многоступенчатые сети

Топология Клоза

В 1953 году Клоз показал, что многоступенчатая сеть на основе элементов типа кроссбар, содержащая не менее трех ступеней, может обладать характеристиками неблокирующей сети [54, 67].

Сеть Клоза с тремя ступенями, показанная на рис. 12.28, содержит r_1 кроссбаров во входной ступени, m кроссбаров в промежуточной ступени и r_2 кроссбаров в выходной ступени. У каждого коммутатора входной ступени есть n_1 входов и m выходов — по одному выходу на каждый кроссбар промежуточной ступени. Коммутаторы промежуточной ступени имеют r_1 входов по числу кроссбаров входной ступени

и r_2 выходов, что соответствует количеству переключателей в выходной ступени сети. Выходная ступень сети строится из кроссбаров с m входами и n_2 выходами. Отсюда числа n_1, n_2, r_1, r_2 и m полностью определяют сеть. Число входов сети $N = r_1 n_1$, а выходов — $M = r_2 n_2$.

Связи внутри составного коммутатора организованы по следующим правилам:

- k-й выход i-го входного коммутатора соединен с i-м входом k-го промежуточного коммутатора;
- k-й вход j-го выходного коммутатора соединен с j-м выходом k-го промежуточного коммутатора.

Каждый модуль первой и третьей ступеней сети соединен с каждым модулем второй ее ступени.

Хотя в рассматриваемой топологии обеспечивается путь от любого входа к любому выходу, ответ на вопрос, будет ли сеть неблокирующей, зависит от числа промежуточных звеньев. Клоз доказал, что подобная сеть является неблокирующей, если количество кроссбаров в промежуточной ступени m удовлетворяет условию: $m \geqslant n_1 + n_2 - 1$. Если $n_1 = n_2 = n$, то матричные переключатели в промежуточной ступени представляют собой «полный кроссбар» и критерий неблокируемости приобретает вид: $m \geqslant 2n - 1$. При условии $m \geqslant n_2$ сеть Клоза можно отнести к неблокирующим сетям с реконфигурацией. Во всех остальных соотношениях между m, n_1 и n_2 данная топология становится блокирующей.

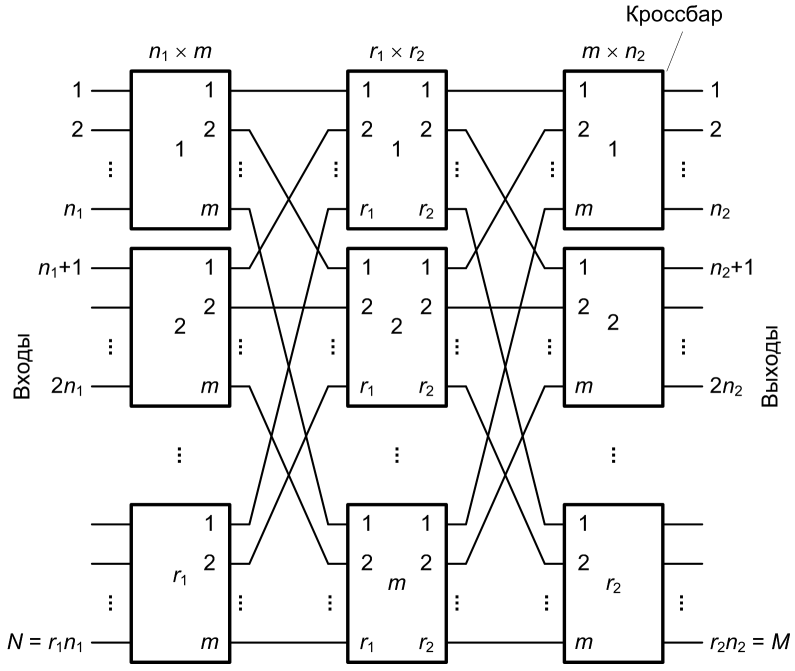


Рис. 12.28. Трехступенчатая сеть с топологией Клоза

Вычислительные системы, в которых соединения реализованы согласно топологии Клоза, выпускают многие фирмы, в частности Fujitsu (FETEX-150), Nippon Electric Company (ATOM), Hitachi. Частный случай сети Клоза при $n_1 = r_1 = r_2 = n_2$ называется сетью «Мемфис». Топология «Мемфис» нашла применение в вычислительной системе GF-11 фирмы IBM.

Сети Бэтчера-Баньяна

Как уже отмечалось, баньян-топология относится к блокирующим, то есть может приводить к конфликтам, если два сообщения пытаются выйти на один и тот же выход БКЭ. В случае коллизии одно сообщение передается, а другое — отбрасывается. Однако если сообщения, поступающие на входы такой сети, упорядочены определенным образом, то конфликтов не происходит. В частности, баньян-сети $n \times n$ в состоянии передавать одновременно вплоть до n пакетов (при условии, что они предназначены разным получателям), если адреса назначения в пакетах, поступающих на входы сети с возрастающими номерами, также упорядочены в порядке возрастания. Такое упорядочение можно реализовать, если перед баньян-сетью поместить так называемую сортирующую сеть, например сеть *Бэтчера* [52].

Сеть *Бэтчера* также состоит из стандартных БКЭ, но они работают несколько по-иному, чем аналогичные элементы баньян-сети. При получении двух пакетов коммутирующий элемент сравнивает хранящиеся в них адреса назначения и направляет пакет с большим адресом по стрелке, а пакет с меньшим адресом — в противоположном направлении. Если имеется только один пакет, то он посылается в направлении, противоположном указанию стрелки. Данный алгоритм иллюстрирует рис. 12.29, в левой части которого показан порядок сортировки пакетов, одновременно направляемых на 4 выходных терминала.

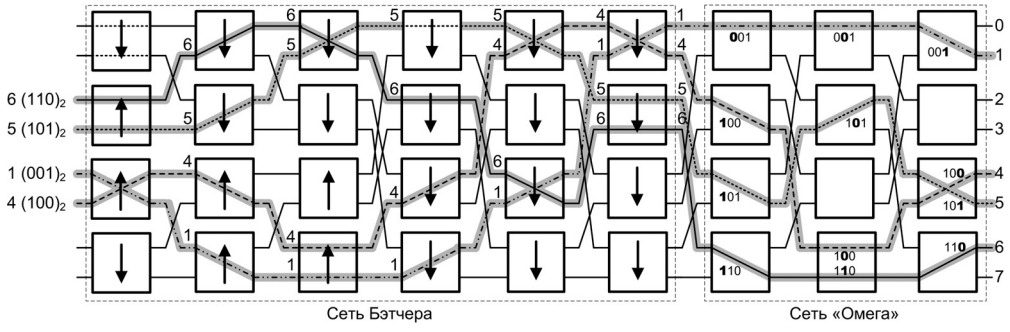


Рис. 12.29. Сеть Бэтчера–Баньяна

С ростом числа входов n количество БКЭ в сети растет в соответствии с зависимостью $n\log_2 n$. При получении k сообщений сеть Бэтчера упорядочивает их в порядке возрастания адресов назначения и выдает на свои первые k выходов. Если теперь выходы сети Бэтчера по схеме полного тасования соединить с входами баньян-сети типа, то образовавшаяся сеть становится неблокирующей. Это относится ко всем вариантам баньян-сетей.

Реконфигурируемые многоступенчатые сети

Топология Бенеша

Рассматриваемая топология [109, 123] относится к типу реконфигурируемых сетей. Это означает, что возможна такая установка коммутаторов сети, при которой одновременная передача сообщений между множеством пар входных и выходных узлов будет происходить без взаимного блокирования. Реконфигурирование предполагает разрыв и переустановку всех соединений [109, 123].

В качестве коммутаторов в *сетях Бенеша* используются β-элементы. В сети $n \times n$ общее количество БКЭ равно $n \log_2 n$. Ввиду того, что рассматриваемые сети не обладают свойством самомаршрутизации, то есть для коммутации маршрута недостаточно информации, содержащейся во входном пакете, для определения состояния всех β-элементов необходим специальный контроллер.

Сущность реконфигурирования и достигаемый при этом эффект иллюстрирует рис. 12.30. На рис. 12.30, *а* показана топология сети Бенеша, обеспечивающая одновременную пересылку трех сообщений: с входа 0_{10} (000_2) на выход 4_{10} (100_2), входа 1_{10} (001_2) на выход 0_{10} (000_2) и входа 2_{10} (010_2) на выход 2_{10} (010_2). Как видно, блокировки сообщений не происходит. При попытке одновременно с этим передать пакет с входа 3_{10} (011_2) на выход 1_{10} (001_2) передача блокируется. Однако блокировки можно избежать, если изменить топологию сети. Рисунок 12.30, *б* иллюстрирует состояние сети после такой реконфигурации. Благодаря изменению трактов прохождения всех четырех сообщений конфликт не возникает.

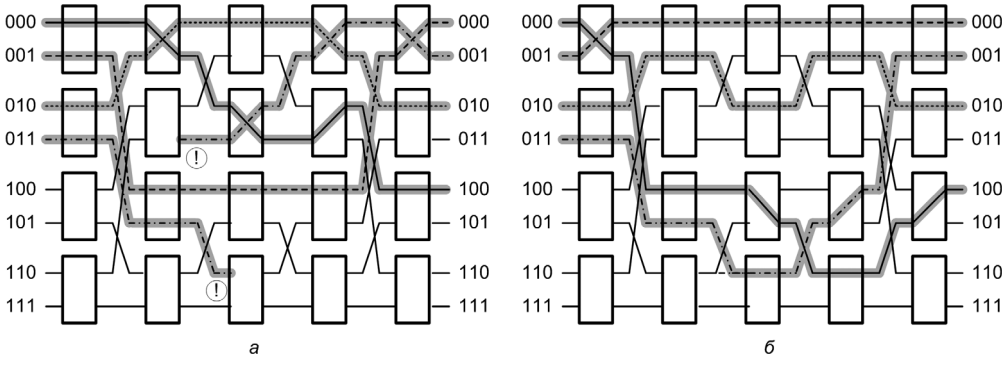


Рис. 12.30. Сеть Бенеша: *а* — до реконфигурации; *б* — после реконфигурации

Обратим внимание на то, что все тракты передачи сообщений должны формироваться одновременно. Если новый тракт формируется при уже функционирующих других трактах, неблокируемость без реконфигурирования «старых» трактов не гарантируется.

Сеть Бенеша может рассматриваться как частный случай сети Клоза, для которой выполняется условие $m \geqslant n$. В этом случае, как доказал Бенеш, сеть Клоза можно отнести к неблокирующим сетям с реконфигурацией.

С добавлением к такой сети дополнительной ступени БКЭ число возможных маршрутов удваивается. Дополнительные пути позволяют изменять трафик сообщения с целью устранения конфликтов. Сеть Бенеша с n входами и n выходами имеет симметричную структуру, в каждой половине которой (верхней и нижней) между входными и выходными БКЭ расположена такая же сеть Бенеша, но с $\frac{n}{2}$ входами и $\frac{n}{2}$ выходами.

Контрольные вопросы

1. Прокомментируйте классификацию сетей по топологии, а также стратегиям синхронизации, коммутации и управления.
2. Дайте развернутую характеристику метрик, описывающих соединения сети.
3. Сформулируйте достоинства и недостатки наиболее известных функций маршрутизации данных.
4. Обоснуйте достоинства и недостатки линейной топологии сети.
5. Дайте характеристику плюсов и минусов кольцевой топологии сети. Какие варианты этой топологии практически используются?
6. Проведите сравнительный анализ звездообразной и древовидной топологий сети.
7. Выполните сравнительный анализ известных вариантов решетчатой топологии сети.
8. Поясните идею динамической топологии. Охарактеризуйте оба класса сетей с динамической топологией.
9. В чем суть деления сетей на основе коммутаторов на блокирующие, неблокирующие и реконфигурируемые?
10. Проведите сравнительный анализ одношинной и многошинной топологий динамических сетей, акцентируя их сильные и слабые стороны.
11. Дайте развернутую характеристику коммутирующих элементов для сетей с динамической топологией.
12. Какая цель преследуется при использовании многоступенчатых динамических сетей? Как классифицируются эти сети?
13. Сравните популярные разновидности баньян-сетей: «Омега», «Дельта».
14. Дайте развернутое объяснение отличий топологии Клоза от баньян-сетей. Можно ли найти у них сходные черты, и если можно, то какие?
15. Какую проблему баньян-сетей позволяет решить сеть Бэтчера-Баньяна? Каким именно образом? Ответ обоснуйте, приведя конкретный пример.
16. Можно ли сказать, что сеть Бенеша занимает промежуточное положение между баньян-сетью и сетью Клоза? Ответ обоснуйте.

Глава 13

Вычислительные системы класса SIMD

Процессоры потоков данных

В предыдущем разделе упоминалась особенность мультимедийных приложений — работа с массивами данных, все элементы которых подвергаются однотипной обработке. Реализация подобных приложений предполагает чрезвычайно высокую скорость арифметических вычислений. Для этого процессор должен поддерживать одновременную работу от десятков до сотен АЛУ. Второй аспект проблемы — минимизация времени доступа к памяти при считывании и записи обрабатываемых массивов данных, а также промежуточных массивов, возникающих в ходе вычислений.

Решить эту задачу позволяет архитектура обработки потоков данных, зачастую не совсем корректно называемая потоковой архитектурой. Центральная ее идея состоит в представлении медийных вычислений в виде потоков данных (data streams) и вычислительных ядер (kernels), что позволяет явно отразить их характерные черты — локальность и параллелизм.

Потоком данных здесь называют последовательность (набор, список) однотипных элементов. Элементы потоков (записи) могут быть как простыми (одиночное число), так и сложными (например, координаты треугольника в трехмерном пространстве). Потоки не обязательно должны иметь одинаковую длину (содержать одинаковое количество записей).

Под вычислительным ядром понимается небольшая циклическая программа, которая применяется к каждому элементу обрабатываемого потока данных.

ВНИМАНИЕ

Используемый здесь термин «ядро» (kernel) не следует путать с аналогичным понятием, применяемым для многоядерных процессоров, которое в английском языке обозначается словом core и имеет иной смысл.

Программа обработки потоков данных представляется в виде взаимосвязанных вычислительных ядер, через которые последовательно проходят потоки данных. На рис. 13.37 показана структура программы, которая преобразует изображение, полученное с помощью цифровой фотокамеры, в формат MPEG2. Рисунок отображает вычисления как последовательность вычислительных ядер и проходящих через них потоков данных. Заметим, что ядра 2 и 3 применяются двукратно.

Такая трактовка медийных приложений позволяет эффективно использовать их потенциал локальности и параллелизма и поддерживается процессорами потоков данных (stream processors).

Для распараллеливания аппаратной обработки элементов потока необходимо иметь достаточное количество операционных устройств, которые объединяются в так называемые кластеры АЛУ. На рис. 13.38 показан пример кластера АЛУ Imagine, созданного в Массачусетском технологическом институте и Стэнфордском университете.

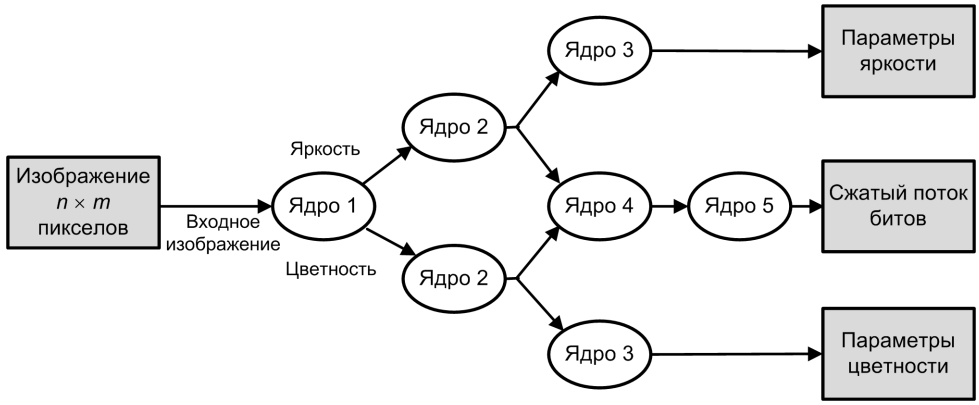


Рис. 13.37. Преобразование матрицы пикселов в формат MPEG2

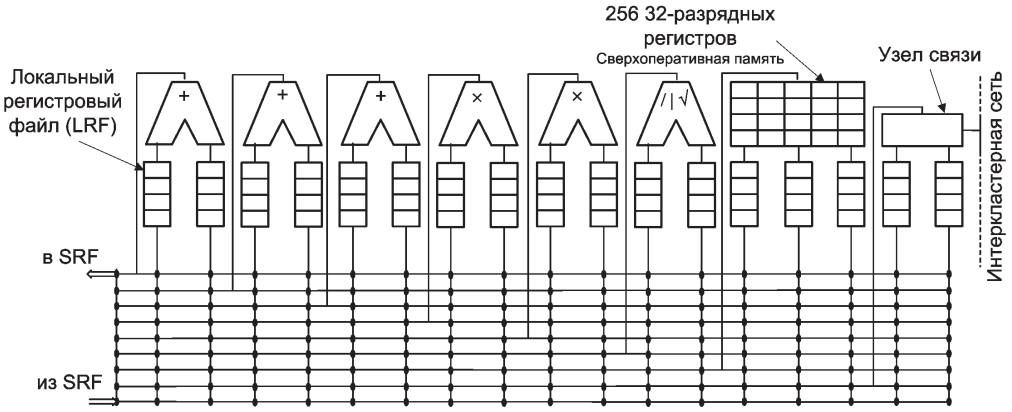


Рис. 13.38. Кластер АЛУ

Кластер АЛУ содержит шесть операционных устройств: три сумматора, два устройства умножения и одно устройство деления/извлечения квадратного корня. Каждому операционному устройству придан входной локальный регистровый файл (LRF — Local Register File). Кроме того, в кластер входят коммуникационный узел (узел связи) и сверхоперативная память, состоящая из 256 32-разрядных регистров. Они тоже используют свои LRF.

Команда, поступающая в кластер АЛУ, имеет структуру VLIW, то есть объединяет несколько простых команд, каждая из которых предназначается для одного из

операционных устройств кластера АЛУ. Идея проста: максимально загрузить работой имеющиеся ОПУ. Например, если элемент потока — это координаты точки в трехмерном пространстве, то кластер АЛУ способен одновременно задействовать все три сумматора. Соответствующая VLIW-команда будет содержать три простые команды. Таким образом, кластер АЛУ обеспечивает распараллеливание действий при обработке *одного* элемента.

В состав процессора потоков данных (ППД) входят несколько одинаковых кластеров АЛУ. Это позволяет распараллеливать действия при обработке *нескольких* элементов потока. Так, в ППД Imagine 8 кластеров обрабатывают параллельно 8 элементов одного или нескольких потоков данных.

Обобщенная структура процессора потоков данных показана на рис. 13.39.

Помимо кластеров АЛУ в составе ППД имеются контроллер потоков, микроконтроллер, контроллер внешней памяти (строго говоря, он не входит в состав ППД) и регистровый файл для работы с потоками (SRF — Stream Register File). Поясним назначение этих устройств.

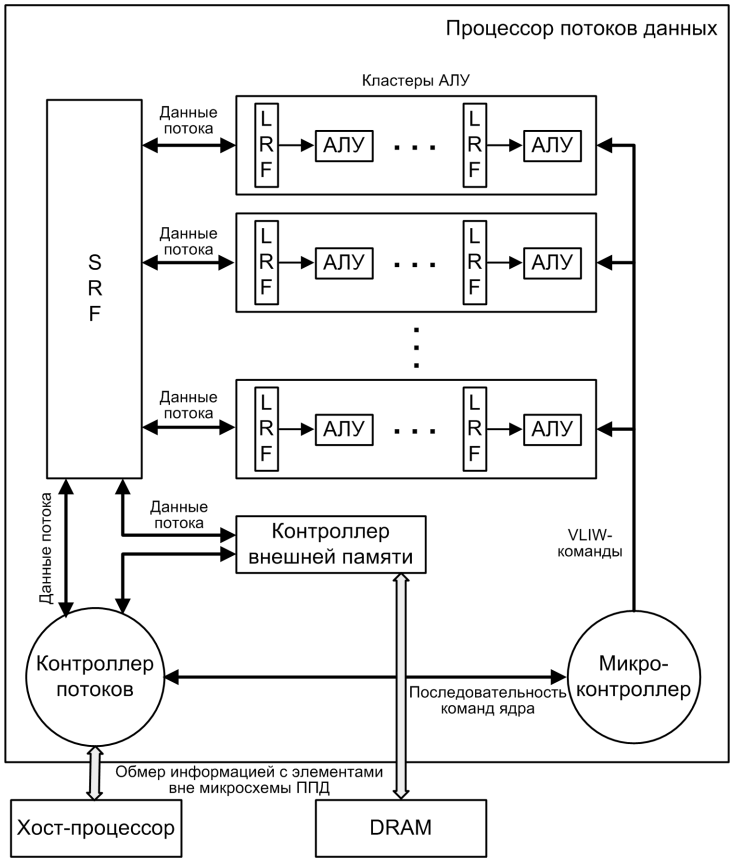


Рис. 13.39. Обобщенная схема процессора потоков данных

ППД обычно разрабатывается как сопроцессор для процессора общего назначения (хост-процессора). Хост-процессор хранит программу приложения и выполняет скалярные вычисления. Обработка потоков данных делегируется ППД. Для этого каждая команда вычислительного ядра, а также команды загрузки/сохранения потоков передаются из хост-процессора в контроллер потоков ППД.

Контроллер потоков управляет обменом информацией с хост-процессором. Этот контроллер принимает команды, предназначенные для ППД, и транслирует их во VLIW-команды, посылаемые в микроконтроллер.

Получив VLIW-команду, микроконтроллер направляет ее в кластеры АЛУ. Каждый кластер АЛУ оперирует с одним элементом потока, а все кластеры одновременно выполняют общую команду. Это существенно упрощает продвижение данных через процессор потоков данных, если учесть, что количество записей в разных потоках может меняться в широких пределах.

Естественно, что эффективность подобной системы должна обеспечиваться высоким темпом подачи информации в ОПУ и сохранения полученных результатов. В этих условиях было решено отказаться от кэш-памяти, а вместо нее реализовать трехуровневую иерархию запоминающих устройств. Эти три уровня образуют:

- локальные регистровые файлы (LRF), расположенные на входе каждого операционного устройства кластера АЛУ и служащие для хранения соответствующих операндов;
- регистровый файл для работы с потоками (SRF), хранящий обрабатываемые потоки данных и обменивающийся информацией с LRF;
- внешняя динамическая оперативная память (DRAM), не входящая в ППД и используемая для хранения глобальных данных.

Контроллер внешней памяти управляет обращениями к глобальным данным, хранящимся в DRAM.

Три уровня памяти формируют иерархию значений пропускной способности, где этот показатель у LRF на порядок больше, чем у SRF, а у SRF — на порядок больше, чем у DRAM. Такая иерархия отвечает природе медиаприложений и позволяет сотням АЛУ работать на скоростях, близких к пиковым.

Рассмотренный процессор потоков данных использует параллелизм на уровне данных в потоке путем параллельного выполнения команд вычислительного ядра над несколькими последовательными элементами потока (по одному в каждом кластере). В известном смысле ППД напоминает векторный процессор, если рассматривать поток как вектор, не учитывая возможную сложную структуру элементов этого вектора. Это обстоятельство дает основания причислить ППД к классу SIMD.

Глава 17

Бортовые вычислительные машины и комплексы летательных аппаратов

Эффективность БЦВМ

Большие потенциальные возможности цифровой вычислительной техники обусловили использование БЦВМ в качестве основного средства реализации алгоритмов управления ЛА. Число и сложность решаемых с помощью БЦВМ задач имеют постоянную тенденцию к росту. Вместе с тем, установка БЦВМ на борту ЛА накладывает жесткие ограничения на физические, вычислительные и эксплуатационные характеристики БЦВМ. Указанное противоречие делает особо острой проблемы всестороннего анализа параметров БЦВМ, оценки правильности и обоснованности принятых проектно-конструкторских решений.

Современные БЦВМ, встраиваемые в летательные аппараты и управляющие их работой, относятся к классу сложных систем. Они состоят из большого числа компонентов, размещенных в кристаллах СБИС, и имеют множество показателей, характеризующих многочисленные свойства машины. Рассмотрим методику оценки эффективности БЦВМ.

Основные показатели для оценки эффективности БЦВМ

Показатели, позволяющие оценить эффективность БЦВМ, зададим в виде двух векторов:

- вектора технических показателей *TP*;
- вектора алгоритмических показателей *AP*.

Вектор технических показателей имеет вид

$$TP = \{C, M, W, P, R_к, R_ч, E_{ОЗУ}, E_{ПЗУ}\},$$

где $TP_1 = C$ — сложность, условных элементов; $TP_2 = M$ — масса, кг; $TP_3 = W$ — объем, л; $TP_4 = P$ — потребляемая мощность, Вт; $TP_5 = R_к$ — разрядность команды, битов; $TP_6 = R_ч$ — разрядность числа, битов; $TP_7 = E_{ОЗУ}$ — емкость ОЗУ, К слов, К байт; $TP_8 = E_{ПЗУ}$ — емкость ПЗУ, К слов, К байт.

Числовые значения этих показателей для первых отечественных БЦВМ на интегральных схемах с малой (ИС), средней (СИС) и большой (БИС) степенью интеграции приведены в табл. 16.5[1].

[1] Конечно, в настоящее время эти данные имеют лишь историческое значение и иллюстрируют разработки, выполненные в XX веке, но они реально отображают этапы «большого пути»:).

Таблица 16.5. Технические показатели первых БЦВМ на ИС, СИС, БИС

Элементная база	**ИС**	**СИС**	**БИС**
Сложность, условных элементов	8150–8500	3600–3750	3500–3900
Масса, кг	25–29	13–14	10–12
Объем, л	35–40	18–20	12–15
Потребляемая мощность, Вт	118–170	80–84	50–70
Разрядность команды, битов	16–26	16–24	32
Разрядность числа, битов	12–16	16–24	32
Емкость ОЗУ	0,5 К–1 К слов	2 К–8 К слов	4 К–16 К байтов
Емкость ПЗУ	14 К–16 К слов	32 К–64 К слов	64 К–128 К байтов

Дадим некоторые пояснения по поводу расчета сложности. Сложность БЦВМ должна оцениваться аппаратными затратами на ее реализацию. Обычно показатель сложности зависит от количества и сложности элементов, составляющих машину:

$$C = \sum_{i=1}^{n} c_i h_i,$$

где n — количество типов элементов; c_i — сложность элемента i-го типа; h_i — количество элементов i-го типа.

Практика разработки БЦВМ породила много вариантов оценки сложности по этой формуле, отличающихся друг от друга:

- выбором объекта в роли элемента;
- методикой оценки сложности элемента.

В инженерной практике в роли элементов для оценки сложности используются:

- электрорадиоэлементы (ЭРЭ) — резисторы, конденсаторы, усилители и т. д.;
- логические элементы (электрические схемы для реализации простейших логических функций);
- интегральные схемы.

В качестве меры сложности элементов применяются следующие оценки:

- суммарное число входов логического элемента (оценка по Квайну);
- оценка, пропорциональная площади, занимаемой элементом на кристалле полупроводника;
- число функций, реализуемых элементом.

Довольно часто при использовании однотипной элементной базы сложность элементов вообще игнорируется, то есть расчет ведется по количеству корпусов интегральных схем и электрорадиоэлементов. Именно этот простейший способ расчета мы и использовали в табл. 16.5, называя условными элементами различные разновидности как цифровых, так и аналоговых элементов, применяемых для аппаратной реализации БЦВМ, и игнорируя их «внутреннюю» сложность.

Заметим, что переход от ИС к СИС привел к сокращению числа комплектующих элементов более чем в два раза, в то время как переход от СИС к БИС такого уменьшения не дал. Причина проста: в первых БЦВМ на БИС по-прежнему пришлось применять большое число интегральных схем с малой и средней степенью интеграции. Их доля составляла 93–94 % от общего числа интегральных схем.

Вектор алгоритмических показателей представим следующим образом:

$$AP = \{V_{\text{ном}},\ V_{\text{ср}}^{\text{полета}},\ \mu,\ \Pi_{\text{полета}},\ \delta_R,\ \delta_E\},$$

где $V_{\text{ном}}$ — номинальное быстродействие БЦВМ; $V_{\text{ср}}^{\text{полета}}$ — среднее быстродействие БЦВМ при решении задач управления полетом летательного аппарата; μ — коэффициент эффективности системы операций; $\Pi_{\text{полета}}$ — производительность БЦВМ при решении задач управления полетом ЛА; δ_R — показатель расширения разрядности чисел БЦВМ ($\delta_R = 1$, если расширение разрядности возможно, $\delta_R = 0$ в противном случае); δ_E — показатель расширения емкости памяти БЦВМ ($\delta_E = 1$, если увеличение емкости памяти возможно).

Номинальное быстродействие зависит только от аппаратных возможностей БЦВМ при выполнении стандартной операции. В качестве стандартной обычно выбирают короткую операцию сложения в формате RX. Если обозначить через $\tau_{\text{сл-RX}}$ время этого сложения, то номинальное быстродействие определится из выражения

$$V_{\text{ном}} = \frac{1}{\tau_{\text{сл-RX}}}\ \ [\text{оп/с}].$$

Среднее быстродействие в задачах управления полетом является обобщенным показателем и может использоваться в интегральном критерии эффективности, поскольку зависит как от большого числа параметров БЦВМ, так и от алгоритмов решения задач управления ЛА.

Формула для расчета $V_{\text{ср}}^{\text{полета}}$ была, фактически, представлена в главе 1 учебника как формула (1.5) среднего быстродействия при решении «полной» задачи. С учетом настройки на управление полетом она имеет следующий вид:

$$V_{\text{ср}}^{\text{полета}} = \frac{1}{\sum_{j=1}^{m}\sum_{i=1}^{l}\beta_j q_{ji}\tau_i}\ \ [\text{оп/с}], \qquad (16.15)$$

где m — количество частных задач в полной задаче управления полетом; l — количество типов операций БЦВМ на j-й частной задаче; β_j — частота появления операций j-й частной задачи в полной задаче управления полетом ЛА; q_{ji} — частота операций i-го типа в j-й частной задаче; τ_i — время выполнения операции i-го типа в j-й частной задаче управления.

Для расчета по формуле (16.15) необходимо знать параметры БЦВМ, представленные вектором $\{\tau_1, \tau_2, ..., \tau_l\}$, параметры каждой j-й частной задачи управления — вектор $\{q_{j1}, q_{j2}, ..., q_{jl}\}$ и параметры полной задачи управления полетом — вектор $\{\beta_1, \beta_2, ..., \beta_m\}$. Наиболее трудоемко определение множества векторов $\{q_{ji}, | j = 1, ..., m, i = 1, ..., l\}$, так как при этом требуется статистическая оценка программ реализации всех частных задач управления.

Для универсальных ВМ статистическая оценка реальных программ заменяется использованием стандартного частотного вектора, определенного на смеси научно-технических задач. Однако параметры типовой смеси научно-технических задач сильно отличаются от параметров задач управления полетом ЛА, поэтому стандартный частотный вектор не может быть использован при расчете $V_{\text{ср}}^{\text{полета}}$. Отсюда вытекает необходимость определения множества эталонных частотных векторов для задач управления полетом.

Зная $V_{\text{ном}}$ и $V_{\text{ср}}^{\text{полета}}$, легко вычислить эффективность системы операций БЦВМ:

$$\mu = V_{\text{ном}} / V_{\text{ср}}^{\text{полета}}.$$

Значение коэффициента μ характеризует степень соответствия системы операций БЦВМ алгоритмам задач управления и показывает, во сколько раз снижается быстродействие БЦВМ при работе по реальным программам полета по сравнению с $V_{\text{ном}}$.

Чтобы иметь возможность анализировать эффективность реализации отдельных операций БЦВМ, введем критерий для оценки влияния отдельных операций на среднюю длительность операций частных задач управления полетом. В качестве такого критерия удобно использовать коэффициент относительных приращений к средней длительности операции $\tau_{\text{ср}j}$ за счет операций i-го типа

$$\eta_{ji} = q_{ji}\left(\frac{\tau_i}{\tau_{\text{сл-RX}}} - 1\right) \to \min. \qquad (16.16)$$

С использованием (16.16) выражение для средней длительности операций j-й частной задачи управления примет вид

$$\tau_{\text{ср}j} = \left(1 + \sum_{j=1}^{l} \eta_{ji}\right)\tau_{\text{сл-RX}}. \qquad (16.17)$$

Из (16.17) следует, что модуль коэффициента η_{ji} характеризует величину изменения средней длительности относительно длительности базовой операции из-за появления в программе i-й операции с частотой q_{ji}. Знак коэффициента η_{ji} свидетельствует о направлении этого изменения — в сторону увеличения или уменьшения $\tau_{\text{ср}j}$ относительно $\tau_{\text{сл-RX}}$.

Производительность БЦВМ при решении задач управления полетом определяется из отношения

$$\Pi_{\text{полета}} = \frac{V_{\text{ср}}^{\text{полета}}}{\sum_{j=1}^{m} \alpha_j N_j}, \qquad (16.18)$$

где α_j — цикличность включения j-й частной задачи в полную задачу управления полетом; N_j — количество операций в j-й частной задаче. Понятно, что произведение $\alpha_j N_j$ — это общее количество операций j-й частной задачи в составе полной задачи управления полетом.

Если учесть, что

$$\beta_j = \frac{\alpha_j N_j}{\sum_{j=1}^{m} \alpha_j N_j}, \qquad (16.19)$$

то, подставляя в (16.18) выражение (16.15) и переходя к размерности «задача/с», получим окончательную формулу расчета производительности БЦВМ:

$$\Pi_{\text{полета}} = \frac{1}{\sum_{j=1}^{m} \sum_{i=1}^{l} \alpha_j N_j q_{ji} \tau_i} \ [\text{задач/с}]. \qquad (16.20)$$

Обобщенные критерии эффективности БЦВМ

Комплексный выбор наиболее эффективной БЦВМ является многокритериальной задачей, качество решения x которой требует введения множества частных критериев:

$$K(x) = \{k_1(x), k_2(x), ..., k_n(x)\}. \qquad (16.21)$$

Частные критерии $k_s(x)$ строятся на основе векторов *TP* и *AP*.

При многокритериальном подходе недостаточно задать множество частных критериев и их связей с параметрами БЦВМ — необходимо также выбрать схему компромиссов, на основе которой формулируются правила поиска решения. Выделяют две группы схем компромиссов и, соответственно, два подхода к решению задачи:

- свертывание векторного критерия в скалярный;
- лексикографическая оптимизация.

В методах свертывания векторная задача сводится к скалярной задаче

$$K(x) \to \underset{x \in X}{\text{extr}},$$

где X — множество допустимых решений; $K(x)$ — скалярный критерий, являющийся некоторой функцией от значений компонентов векторного критерия (16.21):

$$K(x) = f\left(k_1(x), k_2(x), ..., k_n(x)\right).$$

Главная особенность этого подхода состоит в необходимости построения функции свертки f. Основные трудности при этом возникают в процессе учета приоритетов частных критериев, который выполняется посредством задания вектора коэффициентов важности критериев

$$\Lambda = \{\lambda_1, \lambda_2, ..., \lambda_n\},$$

где λ_i — коэффициент важности частного критерия $k_i(x)$.

В основе лексикографической оптимизации лежит идея абсолютного упорядочения частных критериев по взаимной важности и последовательного их применения для получения оптимального решения. Принципиально важным отличием лексикографической оптимизации от методов свертывания является достаточность

качественного, а не количественного упорядочения. Возможность качественного упорядочения частных критериев в значительной степени упрощает решение задачи.

Для оценки эффективности БЦВМ особый интерес представляет частный случай лексикографической оптимизации — оптимизация по скалярному критерию при наличии ограничений:

$$k_1(x) \to \underset{x \in X}{\text{extr}};$$
$$k_i(x) \geq k_i^{\text{треб}},\ i = 2, ..., n;$$
$$k_j(x) \leq k_j^{\text{треб}},\ j = n+1, ..., m,$$

где $k_1(x)$ — наиболее важный частный критерий; $k_i^{\text{треб}}$ — требуемое значение по i-му критерию.

Основное преимущество данного подхода состоит в том, что задача может решаться хорошо проработанными методами математического программирования. Кроме того, при такой постановке на области значений $\underset{x \in X}{k_i(x)}$ не накладываются требования непрерывности и выпуклости.

Обобщенный критерий эффективности БЦВМ должен учитывать как технические, так и алгоритмические характеристики БЦВМ. Поэтому при синтезе первого частного критерия удобно взять за основу критерий цены эффективного быстродействия, предложенный С. А. Майоровым и Г. И. Новиковым и основанный на идеях В. М. Глушкова:

$$K = \frac{S}{V_{\text{ср}}} \to \min, \qquad (16.22)$$

где S — стоимость; $V_{\text{ср}}$ — среднее быстродействие ВМ.

Очевидно, что для «настройки» на БЦВМ показатель $V_{\text{ср}}$ в (16.22) надо заменить на $V_{\text{ср}}^{\text{полета}}$:

$$K = \frac{S}{V_{\text{ср}}^{\text{полета}}} \to \min. \qquad (16.23)$$

Формула критерия (16.23) характеризует денежные затраты, приходящиеся на единицу быстродействия в процессе управления полетом ЛА.

Применительно к целям оценки БЦВМ усовершенствуем критерий (16.23) следующим образом. Во-первых, вместо денежных будем оценивать аппаратные затраты БЦВМ, используя показатель сложности C [усл. элем.]. Во-вторых, вместо среднего быстродействия при управлении полетом введем показатель информационного быстродействия при управлении полетом

$$V_{\text{инф}}^{\text{полета}} = V_{\text{ср}}^{\text{полета}} \sum_{j=1}^{m} \sum_{i=1}^{l} \beta_j q_{ji} R_i \ \ [\text{тыс. бит/с}],$$

где R_i — разрядность операнда при выполнении i-й операции; показатель $V_{\text{ср}}^{\text{полета}}$ и параметры β_j, q_{ij} рассчитываются по формулам (16.15), (16.19). Использование

показателя $V_{\text{инф}}^{\text{полета}}$ позволяет сравнивать БЦВМ с различной разрядностью (в том числе с изменяющейся разрядностью).

С учетом сказанного получим критерий цены информационного быстродействия при управлении полетом:

$$K_{\text{циб}} = \frac{C}{V_{\text{ср}}^{\text{полета}} \sum_{j=1}^{m} \sum_{i=1}^{l} \beta_j q_{ji} R_i} \left[\text{усл. элем.} \cdot \text{с/тыс. бит}\right] \to \min;$$
$$TP_k \le TP_k^{\text{треб}},\ k = 1,\ 2,\ 3,\ 4; \tag{16.24}$$
$$TP_k \ge TP_k^{\text{треб}},\ k = 5,\ 6,\ 7,\ 8,$$

где TP_k — k-й технический показатель БЦВМ (компонент вектора TP); $TP_k^{\text{треб}}$ — требуемое значение k-го технического показателя БЦВМ.

Формула критерия (16.24) учитывает основные особенности вычислительного процесса в БЦВМ:

- работу в реальном масштабе времени;
- разделение машинного времени между частными алгоритмами управления;
- модульность частных алгоритмов;
- цикличность выполнения алгоритмов управления;
- динамически изменяемую разрядность вычислений;
- наличие жестких ресурсных ограничений.

Однако при этом игнорируется степень совмещения в работе центрального процессора и устройств ввода-вывода, а также время, затрачиваемое на пересылку данных из внешней памяти в основную. Отмеченные факторы можно учесть в критерии цены производительности при управлении полетом:

$$K_{\text{ЦПр}} = \frac{C}{\Pi_{\text{полета}}} \left[\text{усл. элем.} \cdot \text{с/задача}\right] \to \min;$$
$$TP_k \le TP_k^{\text{треб}},\ k = 1,\ 2,\ 3,\ 4;$$
$$TP_k \ge TP_k^{\text{треб}},\ k = 5,\ 6,\ 7,\ 8,$$

где производительность БЦВМ при выполнении полного алгоритма управления полетом оценивается по формуле (16.20).

Эталонные частотные векторы для задач управления полетом

Для создания эталонных частотных векторов управления полетом была принята следующая *гипотеза*: система команд эталонной БЦВМ является подмножеством команд с фиксированной запятой IBM 370/390, базовая разрядность чисел — 16 битов (обозначим R), удвоенная разрядность — 32 бита (обозначим $2R$).

Программировались типовые алгоритмы управления полетом: навигационной задачи (нз), наведения (нав), стабилизации центра масс (стаб). Транслятор генерировал ассемблерный код выбранного подмножества команд [185]. Результаты статистической обработки программ управления полетом имеют следующий вид.

Вектор навигации

Определяется по формуле

$$\tau_{ср}^{нз} = 0{,}54\tau_{RX} + 0{,}36\tau_{RI} + 0{,}04\tau_{2R} + 0{,}04\tau_{Mul-2R} + 0{,}02\tau_{Sft11-2R},$$

где $\tau_{ср}^{нз}$ — средняя длительность операции навигации; τ_{RX} — длительность операции типа сложения в формате RX; τ_{RI} — длительность операции типа сложения в формате RI; τ_{2R} — длительность операции типа сложения удвоенной разрядности в формате RX; τ_{Mul-2R} — длительность операции умножения удвоенной разрядности в формате RX; $\tau_{Sft11-2R}$ — длительность операции сдвига на 11 разрядов операнда удвоенной разрядности в формате RX. Длина программы навигации $N_{нз} = 1137$, частота появления ее операций в цикле полного управления $\beta_{нз} = 0{,}144$, цикличность (количество включений) программы навигации в цикле полного управления $\alpha_{нз} = 1$.

Вектор наведения

Определяется по формуле

$$\tau_{ср}^{нав} = 0{,}56\tau_{RX} + 0{,}35\tau_{RI} + 0{,}01\tau_{Mul} + 0{,}03\tau_{2R} + 0{,}03\tau_{Mul-2R} + 0{,}02\tau_{Sft4-2R},$$

где $\tau_{ср}^{нав}$ — средняя длительность операции наведения; τ_{Mul} — длительность операции умножения в формате RX; $\tau_{Sft4-2R}$ — длительность операции сдвига на 4 разряда операнда удвоенной разрядности в формате RX. Длина программы наведения $N_{нав} = 3330$, частота появления ее операций в цикле полного управления $\beta_{нав} = 0{,}421$, цикличность (количество включений) программы наведения в цикле полного управления $\alpha_{нав} = 1$.

Вектор стабилизации

Определяется по формуле

$$\tau_{ср}^{стаб} = 0{,}59\tau_{RX} + 0{,}28\tau_{RI} + 0{,}09\tau_{Sft6} + 0{,}04\tau_{Mul},$$

где $\tau_{ср}^{стаб}$ — средняя длительность операции стабилизации; τ_{Sft6} — длительность операции сдвига на 6 разрядов операнда одинарной разрядности в формате RX. Длина программы стабилизации $N_{стаб} = 1721$, частота появления ее операций в цикле полного управления $\beta_{стаб} = 0{,}435$, цикличность (количество включений) программы стабилизации в цикле полного управления $\alpha_{стаб} = 2$.

Информационное быстродействие при управлении полетом

Поскольку в эталонной БЦВМ обрабатываются операнды только с одинарной и двойной разрядностью, вычисление информационного быстродействия при управлении полетом можно проводить по упрощенной формуле

$$V_{\text{инф}}^{\text{полета}} = V_{\text{ср}}^{\text{полета}} \sum_{j=1}^{3} \beta_j \left(q_{jR} R + q_{j2R} 2R \right) \left[\text{тыс. бит/с}\right],$$

где q_{jR} — частота операций обработки чисел разрядности R; q_{j2R} — частота операций обработки чисел разрядности $2R$ в j-й частной задаче управления.

Для значений эталонных частотных векторов формула принимает вид

$$V_{\text{инф}}^{\text{полета}} = V_{\text{ср}}^{\text{полета}} \left(0,95R + 0,05 \cdot 2R\right) \left[\text{тыс. бит/с}\right].$$

Результаты расчета показателей эффективности для БЦВМ, созданной на основе БИС серии К583, приведены в табл. 16.6.

Таблица 16.6. Показатели эффективности для БЦВМ на основе БИС серии К583

Показатель	**Значение**	**Размерность, пояснение**
Сложность	3900	Условных элементов
Средняя длительность операции навигации	4,06	мкс
Средняя длительность операции наведения	3,93	мкс
Средняя длительность операции стабилизации	3,83	мкс
Среднее быстродействие на задачах управления полетом	255 754	Операций в секунду
Эффективность системы операций	1,3	Показывает, во сколько раз снижается быстродействие при работе по реальным программам полета
Производительность при решении задач управления полетом	32,38	Задач управления полетом в секунду
Информационное быстродействие при управлении полетом	4297	Тысяч битов, обрабатываемых в секунду
Цена информационного быстродействия при управлении полетом	0,91	Условных элементов на тысячу битов, обрабатываемых в секунду
Цена производительности при управлении полетом	120,44	Условных элементов на задачу управления полетом в секунду

Приведенные показатели позволяют оценить интегральную эффективность БЦВМ и учитывают основные особенности организации вычислительного процесса и структуры БЦВМ.

Приложение А

Арифметические основы вычислительных машин

В общем перечне проблем, связанных с разработкой и функционированием ВМ, важное место занимают способы записи и кодирования чисел. Обсуждение этих вопросов составляет содержание данного приложения.

Системы счисления

Система счисления — это совокупность символов и правил для обозначения чисел. Известные системы счисления подразделяются на непозиционные и позиционные.

В *непозиционных системах счисления* вес цифры, то есть вклад, который она вносит в значение числа, не зависит от ее позиции в записи числа.

Наиболее известной из подобных систем является римская система счисления. В ней для записи числа используются цифры, обозначаемые буквами латинского алфавита: I = 1; V = 5; X = 10; L = 50; C = 100; D = 500; M = 1000. Вес каждой цифры в любой позиции записи числа остается неизменным. Так, в числе XXXII (тридцать два) вес цифры X в любой позиции равен просто десяти. Числа формируются в соответствии со следующими правилами.

1. Запись из стоящих подряд одинаковых цифр изображает сумму чисел, обозначаемых этими цифрами. Например, число XXX (тридцать) образуется как X+X+X (10 + 10 + 10).
2. Запись, в которой меньшая по весу цифра стоит слева от цифры с большим весом, изображает разность соответствующих чисел. Так, число IV (четыре) формируется как V–I (5 – 1).
3. Запись, в которой цифра с меньшим весом стоит справа от цифры с большим весом, изображает сумму соответствующих чисел. Например, число VI (шесть) образуется как V+I (5 + 1).

В римской системе отсутствует символ нуля, а также обычные и десятичные дроби. Эти особенности плюс неудобство правил образования чисел обусловили то, что римская система, как, впрочем, и иные непозиционные системы счисления, в вычислительной технике применения не нашли.

Для вычислительной техники характерно использование *позиционных систем счисления*, где числа представляются в виде последовательности цифр, в которой значение каждой цифры зависит от ее позиции в этой последовательности. Количество *s* различных цифр, употребляемых в позиционной системе, называется основанием, или базой системы счисления. В общем случае положительное число *X* в позиционной системе с основанием *s* может быть представлено в виде полинома:

$$X = x_{r-1}...x_1x_0,x_{-1}...x_{-p} = x_{r-1}s^{r-1} + ... + x_0s^0 + x_{-1}s^{-1} + ... + x_{-p}s^{-p},$$

где s — база системы счисления, x_i — цифры, допустимые в данной системе счисления ($0 \leqslant x_i \leqslant s-1$). Последовательность $x_{r-1}x_{r-2}...x_0$ образует *целую часть X*, а последовательность $x_{-1}x_{-2}...x_{-p}$ — *дробную часть X*. Таким образом, r и p обозначают количество цифр в целой и дробной частях соответственно. Целая и дробная части разделяются запятой[1]. Цифры x_{r-1} и x_{-p} называются соответственно *старшим* и *младшим* разрядами числа.

Чтобы представить в ВМ числа, заданные в позиционной системе с основанием *s*, необходимо использовать физические элементы, имеющие *s* устойчивых состояний. Природа электронных устройств накладывает ограничения на применение в машинных вычислениях десятичной системы счисления (DEC — decimal), хотя эта система и более привычна. Электронные элементы с более чем двумя устойчивыми состояниями имеют низкие показатели по надежности, быстродействию, габаритам и стоимости, поэтому в ВМ наибольшее применение нашли двоичная (BIN — binary) и двоично-кодированные системы счисления: восьмеричная (OCT — octal), шестнадцатеричная (HEX — hexadecimal), двоично-кодированная десятичная (BCD — binary coded decimal).

В дальнейшем для обозначения используемой системы счисления число будем заключать в скобки, а в индексе указывать основание системы. Так, число *X* по основанию *s* будем обозначать $(X)_s$.

Двоичная система счисления

Основанием системы счисления служит число 2 ($s = 2$), и для записи чисел используются только две цифры: 0 и 1. Чтобы представить любой разряд двоичного числа, достаточно иметь физический элемент с двумя четко различимыми устойчивыми состояниями, одно из которых изображает 1, а другое — 0.

Пример записи числа в двоичной системе:

$$(173{,}625)_{10} = 1\times2^7 + 0\times2^6 + 1\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + \\ + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} = (10101101{,}101)_2.$$

[1] В некоторых странах — точкой.

Недостатком двоичной системы можно считать ее относительную громоздкость, так как для изображения двоичного числа требуется примерно в 3,3 раза больше разрядов, чем при десятичном представлении.

Восьмеричная и шестнадцатеричная системы счисления

Эти системы счисления относятся к двоично-кодированным, в которых основание системы счисления представляет собой целую степень двойки: 2^3 — для восьмеричной и 2^4 — для шестнадцатеричной.

В восьмеричной системе ($s = 8$) используются 8 цифр — 0, 1, 2, 3, 4, 5, 6, 7.

Пример записи в 8-ричной системе:

$$(173{,}625)_{10} = 2 \times 8^2 + 5 \times 8^1 + 5 \times 8^0 + 5 \times 8^{-1} = (255{,}5)_8.$$

В шестнадцатеричной системе ($s = 16$) используется 16 цифр — 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Пример записи в 16-ричной системе:

$$(173{,}625)_{10} = A \times 16^1 + D \times 16^0 + A \times 16^{-1} = (AD{,}A)_{16}.$$

Широкое применение 8-ричной и 16-ричной систем счисления обусловлено двумя факторами.

Таблица А.1. Соответствие между цифрами в различных системах счисления

DEC	BIN	OCT	HEX	BCD
0	0000	0	0	0000
1	0001	1	1	0001
2	0010	2	2	0010
3	0011	3	3	0011
4	0100	4	4	0100
5	0101	5	5	0101
6	0110	6	6	0110
7	0111	7	7	0111
8	1000	10	8	1000
9	1001	11	9	1001
10	1010	12	A	0001 0000
11	1011	13	B	0001 0001
12	1100	14	C	0001 0010
13	1101	15	D	0001 0011
14	1110	16	E	0001 0100
15	1111	17	F	0001 0101

Во-первых, эти системы позволяют заменить запись двоичного числа более компактным представлением (запись числа в 8-ричной и 16-ричной системах будет соответственно в 3 и 4 раза короче двоичной записи этого числа). Во-вторых, взаимное преобразование чисел между 2-ичной системой с одной стороны и 8-ричной

и 16-ричной — с другой осуществляется сравнительно просто. Действительно, поскольку для 8-ричного числа каждый разряд представляется группой из трех двоичных разрядов (триад), а для 16-ричного — группой из четырех двоичных разрядов (тетрад), то для преобразования двоичного числа достаточно объединить его цифры в группы по 3 или 4 разряда соответственно, продвигаясь от разделительной запятой вправо и влево. При этом, в случае необходимости, добавляют нули слева от целой части и/или справа от дробной части, и каждую такую группу — триаду или тетраду — заменяют эквивалентной 8-ричной или 16-ричной цифрой (см. табл. А.1)[1]. Для обратного перевода каждая ОСТ или HEX цифра заменяется соответственно триадой или тетрадой двоичных цифр, причем незначащие нули слева и справа отбрасываются.

Для рассмотренных ранее примеров это выглядит следующим образом:

$$\begin{matrix}010 & 101 & 101 & 101\\ (2 & 5 & 5, & 5)_8\end{matrix} = (10101101{,}101)_2;$$

$$\begin{matrix}1010 & 1101 & 1010\\ (A & D, & A)_{16}\end{matrix} = (10101101{,}1010)_2.$$

Двоично-десятичная система счисления

В двоично-десятичной системе вес каждого разряда равен степени 10, как в десятичной системе, а каждая десятичная цифра кодируется четырьмя двоичными цифрами. Для записи десятичного числа в BCD-системе достаточно заменить каждую десятичную цифру эквивалентной четырехразрядной двоичной комбинацией (см. табл. А.1):

$$\begin{matrix}0001 & 0111 & 0011 & 0110 & 0010 & 0101\\ (1 & 7 & 3, & 6 & 2 & 5)_{10}\end{matrix} = (0001\ 0111\ 0011, 0110\ 0010\ 0101)_{2\text{–}10}.$$

Любое десятичное число можно представить в двоично-десятичной записи, но следует помнить, что это не двоичный эквивалент числа. Это видно из следующего примера:

$$(173, 625)_{10} = (10101101{,}101)_2 = (0001\ 0111\ 0011, 0110\ 0010\ 0101)_{2\text{–}10}.$$

Преобразование позиционных систем счисления

Пусть X — число в системе счисления с основанием s, которое требуется представить в системе с основанием h. Удобно различать два случая.

В первом случае $s < h$ и, следовательно, при переходе к основанию h можно использовать арифметику этой системы. Метод преобразования состоит в представлении числа $(X)_s$ в виде многочлена по степеням s, а также в вычислении этого многочлена по правилам арифметики системы счисления с основанием h. Так, например,

[1] В табл. А.1 ячейки с затемненным фоном содержат не цифры, а числа в соответствующих системах счисления.

удобно переходить от двоичной или восьмеричной системы к десятичной. Описанный прием иллюстрируют следующие примеры:

$$(1101{,}01)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = (13{,}25)_{10},$$
$$(432{,}2)_8 = 4 \times 8^2 + 3 \times 8^1 + 2 \times 8^0 + 2 \times 8^{-1} = (282{,}25)_{10}.$$

В обоих случаях арифметические действия выполняются по правилам системы с основанием 10.

Во втором случае ($s > h$) удобнее пользоваться арифметикой по основанию s. Здесь следует учитывать, что перевод целых чисел и правильных дробей производится по различным правилам. При переводе смешанных дробей целая и дробная части переводятся каждая по своим правилам, после чего полученные числа записываются через запятую.

Перевод целых чисел

Правило перевода целых чисел становится ясным из общей формулы записи числа в произвольной позиционной системе. Пусть число $(X)_s$ в исходной системе счисления s имеет вид $(x_{r-1} \ldots x_1 x_0)_s$. Требуется получить запись числа в системе счисления с основанием h:

$$(X)_h = (y_{w-1} \ldots y_1 y_0)_h = y_{w-1}h^{w-1} + y_{w-2}h^{w-2} + \ldots + y_1 h^1 + y_0 h^0.$$

Для нахождения значений y_i разделим этот многочлен на h:

$$\frac{X}{h} = y_{w-1}h^{w-2} + y_{w-2}h^{w-3} + \ldots + y_1 + \frac{y_0}{h}.$$

Как видно, младший разряд $(X)_h$, то есть y_0, равен первому остатку. Следующий значащий разряд y_1 определяется делением частного $Q_0 = y_{w-1}h^{w-2} + y_{w-2}h^{w-3} + \ldots + y_1$ на h:

$$\frac{Q_0}{h} = y_{w-1}h^{w-3} + y_{w-2}h^{w-4} + \ldots + y_2 + \frac{y_1}{h}.$$

Остальные y_i также вычисляются путем деления полученных частных до тех пор, пока Q_{w-1} не станет равным нулю.

Для перевода целого числа из s-ричной системы в h-ричную необходимо последовательно делить это число и получаемые частные на h (по правилам системы с основанием s) до тех пор, пока частное не станет равным нулю. Старшей цифрой в записи числа в системе с основанием h служит последний остаток, а следующие за ней цифры образуют остатки от предшествующих делений, выписываемые в последовательности, обратной их получению.

В качестве примера рассмотрим последовательность перевода числа 75 из десятичной системы в 2-ичную, 8-ричную и 16-ричную системы счисления (рис. А.1).

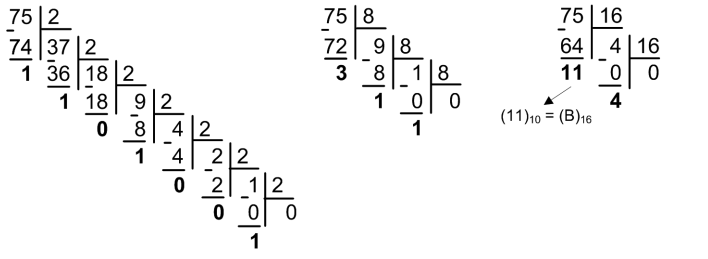


Рис. А.1. Пример перевода целого десятичного числа 75 в 2-ичную, 8-ричную и 16-ричную системы счисления

Перевод правильных дробей

Правильную дробь $(X)_s$, имеющую в системе с основанием s вид $(x_{-1}x_{-2}...x_{-p})_s$, можно выразить в системе с основанием h как многочлен вида:

$$(X)_h = (y_{-1}y_{-2}...y_{-p})_h = y_{-1}h^{-1} + y_{-2}h^{-2} + ... + y_{-p}h^{-p}.$$

Старшая цифра y_{-1} может быть найдена умножением этого многочлена на h, то есть

$$X \times h = y_{-1} + y_{-2}h^{-1} + ... + y_{-p}h^{-p+1}.$$

Если это произведение меньше 1, то цифра y_{-1} равна 0, если же оно больше или равно 1, то цифра y_{-1} равна целой части произведения. Следующая цифра справа y_{-2} определяется путем умножения дробной части указанного выше произведения на h и выделения его целой части и т. д. Процесс может оказаться бесконечным, так как не всегда можно представить дробь по основанию h конечным набором цифр.

Для перевода правильной дроби из системы счисления с основанием s в систему счисления с основанием h нужно умножать исходную дробь и дробные части получающихся произведений на основание h (по правилам «старой» s-системы). Целые части полученных произведений дают последовательность цифр дроби в h-системе.

Описанная процедура продолжается до тех пор, пока дробная часть очередного произведения не станет равной нулю либо не будет достигнута требуемая точность изображения числа X в h-ричной системе. Представлением дробной части числа X в новой системе счисления будет последовательность целых частей полученных произведений, записанных в порядке их получения и изображенных h-ричной цифрой. Абсолютная погрешность перевода числа X при p знаков после запятой равняется $\frac{h^{-(p+1)}}{2}$.

На рис. А.2 приведены примеры перевода правильной дроби 0,453 из десятичной системы в 2-ичную (рис. А.2, *а*), 8-ричную (рис. А.2, *б*) и 16-ричную (рис. А.2, *в*) системы счисления.

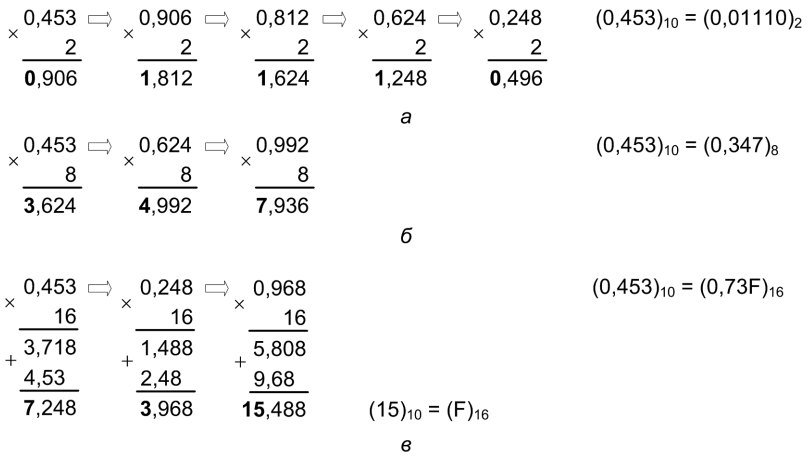


Рис. А.2. Пример перевода десятичной дроби 0,453 в 2-ичную, 8-ричную и 16-ричную системы счисления

Перевод правильных дробей, у которых знаменатель кратен степени нового основания системы счисления

Если преобразуемая дробь имеет вид $(X)_s = \frac{z}{h^m}$, *где m — целое число, то числитель дроби z переводится как целое число, после чего запятая в записи числа перемещается влево на m разрядов.*

Пример перевода из десятичной системы в 2-ичную:

$$X = \frac{12}{256} = \frac{12}{2^8}; (12)_{10} = (1100)_2; X = (0{,}00001100)_2.$$

В отличие от ранее рассмотренного варианта подобный перевод всегда является конечным и точным.

Кодирование отрицательных чисел в ВМ

До сих пор мы рассматривали только положительные числа. При записи двоичных чисел со знаком в их формате необходимо предусмотреть два поля: поле, определяющее знак числа, и поле, характеризующее модуль числа. Под знак числа отводится специальный знаковый бит (двоичный разряд). Остальные разряды определяют модуль числа. Знаковый бит приписывается слева от модуля числа, причем знаку «+» соответствует нулевое значение знакового бита, а «–» – единичное.

В истории развития архитектуры ВМ использовались три основных варианта представления знаковых чисел:

- прямой код или знак и величина;
- обратный код или код с дополнением до единицы;
- дополнительный код или код с дополнением до двух.

Во всех трех кодах (прямом, обратном и дополнительном) положительные числа выглядят одинаково. Для отрицательных чисел различия в форме записи числа касаются только способа представления модуля числа, в то время как способ кодирования и место расположения знакового бита остаются неизменными.

Прямой код двоичного числа

В системе представления в прямом коде число состоит из кода знака и модуля числа, причем обе эти части обрабатываются по отдельности. Формула для образования прямого кода правильной дроби X имеет вид[1]:

$$[X]_{\text{п}} = X, \quad \text{если } X \geqslant 0,$$

$$[X]_{\text{п}} = 1 - X, \quad \text{если } X < 0.$$

Примеры прямого кода для правильных дробей:

$$x_1 = +0{,}0101\ [x_1]_{\text{п}} = 0{,}0101;\ x_2 = -0{,}1011\ [x_2]_{\text{п}} = 1{,}1011.$$

Прямой код целого числа получается по формуле:

$$[X]_{\text{п}} = X, \text{ если } X \geqslant 0,$$

$$[X]_{\text{п}} = 1 \times 2^{n-1} - X, \text{ если } X < 0.$$

Примеры прямого кода для целых чисел:

$$x_1 = +1101\ [x_1]_{\text{п}} = 0{,}1101;\ x_2 = -1011\ [x_2]_{\text{п}} = 1{,}1011.$$

При всей своей наглядности представление чисел в прямом коде имеет существенный недостаток — формальное суммирование чисел с различающимися знаками дает неверный результат. Проиллюстрируем это на примере сложения двух чисел: $x_1 = +5_{10}\ (+101)_2$ и $x_2 = -7\ (-111)_2$. В прямом коде эти числа имеют вид: $[x_1]_{\text{п}} = 0{,}101$ и $[x_2]_{\text{п}} = 1{,}111$. Очевидно, что результат должен быть равен -2, что в прямом коде может быть записано как 1,010. В то же время при непосредственном сложении получаем

$$0{,}101 + 1{,}111 = 10{,}100,$$

то есть значение, существенно отличающееся от ожидаемого.

Для корректного выполнения процедуры суммирования чисел с разными знаками, представленными в прямом коде, необходимо определить больший по модулю операнд, вычесть из него абсолютное значение меньшего операнда и присвоить результату знак большего по модулю операнда.

Отметим также второй недостаток прямого кода — нуль имеет два различных представления, а именно $[+0]_{\text{п}} = 0{,}00..0$ и $[-0]_{\text{п}} = 1{,}00..0$, что математически не имеет смысла.

В силу отмеченных недостатков прямого кода обрабатываемая информация чаще представляется в форме дополнения.

---

[1] В дальнейшем при рассмотрении кодов условимся отделять знаковый разряд от числа точкой (в целых числах) и запятой (в правильных дробях). Кроме того, говоря о n-разрядных числах, будем помнить, что n — количество разрядов, учитывающее также разряд знака.

Представление чисел в форме дополнения

Негативный итог в случае прямого кода является следствием неудачного выбора способа кодирования. Его нужно выбирать исходя из логики использования кода, а не из «напрашивающегося» или «очевидного» на первый взгляд подхода. Основное требование при выборе системы кодирования чисел состоит в том, что полученный код должен удовлетворять правилам выполнения сложения и вычитания в двоичной системе счисления. В связи с этим, код каждого следующего положительного числа должен получаться прибавлением единицы к коду текущего числа, а код каждого следующего отрицательного числа должен получаться вычитанием единицы из кода текущего числа.

При представлении чисел в прямом коде для изменения знака достаточно поменять только значение знакового разряда. В системе *представления чисел в форме дополнения* местоположение знакового разряда и способ кодирования знака остаются теми же, что и при прямом кодировании. В то же время знаковый разряд уже не рассматривается как обособленный, а считается неотъемлемой частью числа и обрабатывается аналогично значащим разрядам и совместно с ними.

Введем понятия поразрядного и точного дополнения цифры. *Поразрядное дополнение цифры x* по основанию *s* определяется выражением

$$x' = (s-1) - x\ .$$

Поразрядное дополнение x' равно разности между наибольшей цифрой в системе счисления *s* и цифрой *x*. В двоичной системе счисления 0' = 1 и 1' = 0, так как *s* = 2, а наибольшая цифра — 1. В десятичной системе счисления наибольшей является цифра 9. Поэтому неполным дополнением цифры 3 будет 6 (9–3).

Точное дополнение цифры x по основанию *s* на единицу больше поразрядного и может быть описано как

$$x'' = s - x\ .$$

Точное дополнение цифры x'' равно разности между числом, равным основанию системы счисления *s*, и цифрой *x*. Так, в десятичной системе счисления точным дополнением цифры 3 будет цифра 7 (10–3).

Взятие дополнения — более сложная процедура, нежели изменение знака, однако два числа, представленные в форме дополнения, можно суммировать и вычитать непосредственно, не проверяя их знаки и величины, как это требуется в прямом коде. В вычислительной технике наибольшее распространение получили две системы представления чисел в форме дополнения, в основе которых лежат соответственно поразрядное дополнение и точное дополнение. Первая из этих систем более известна как обратный код, или дополнение до 1 (1's complement), а вторая — как дополнительный код, или дополнение до 2 (2's complement).

Обратный код двоичного числа

Формула образования обратного кода двоичной дроби *X* имеет вид:

$$[X]_o = 2 - 2^{-(n-1)} + X.$$

Примеры обратного кода правильных дробей:

$$x_1 = +0{,}0101\ [x_1]_о = 0{,}0101;\ x_2 = -0{,}1011\ [x_2]_о = 1{,}0100.$$

Обратный код[1] целого числа формируется в соответствии с выражением:

$$[X]_о = 2^{n-1} + X.$$

Примеры обратного кода целых чисел:

$$x_1 = +1101\ [x_1]_о = 0{,}1101;\ x_2 = -1011\ [x_2]_о = 1{,}0100.$$

Для отрицательных двоичных чисел процедуру образования обратного кода можно свести к следующему формальному правилу: в знаковый разряд записывается единица, а в цифровых разрядах прямого кода единицы заменяются нулями, а нули — единицами.

Напомним, что кодирование распространяется только на отрицательные числа, положительные числа в прямом и обратном коде выглядят идентично[2]:

Диапазон целых чисел, которые могут быть отражены обратным кодом — от $-(2^{n-1} - 1)$ до $+(2^{n-1} - 1)$.

Хотя обратный код и позволяет решить проблему сложения и вычитания чисел с различными знаками, он имеет и определенные недостатки. Так, процесс суммирования чисел является двухэтапным, что увеличивает время выполнения данной операции. Кроме того, как и в прямом коде, имеют место два представления нуля: $[+0]_о = 0{,}00..0$ и $[-0]_о = 1{,}11..1$, поэтому схема обнаружения нуля в вычислительном устройстве должна либо проверять обе возможности, либо преобразовывать 1,11..1 в 0,00..0.

Дополнительный код двоичного числа

Дополнительный код двоичной дроби X образуется по формуле:

$$[X]_д = 2 + X.$$

Примеры дополнительного кода для правильных дробей:

$$x_1 = 0{,}0101\ [x_1]_д = 0{,}0101;\ x_2 = -0{,}1011\ [x_2]_д = 1{,}0101.$$

Формула для образования дополнительного кода целого числа X:

$$[X]_д = 2^n + X.$$

Примеры дополнительного кода для целых чисел:

$$x_1 = 1101\ [x_1]_д = 0{,}1101;\ x_2 = -1011\ [x_2]_д = 1{,}0101.$$

Дополнительный код отрицательного двоичного числа формируется по следующему правилу: в знаковый разряд записать единицу, в цифровых разрядах прямого кода

---

[1] При рассмотрении формул для обратного и дополнительного кодов следует помнить, что речь идет об отрицательных двоичных числах, то есть X в формулах — это отрицательное двоичное число.

[2] В дальнейшем при рассмотрении кодов условимся отделять знаковый разряд от числа точкой (в целых числах) и запятой (в правильных дробях). Кроме того, говоря о n-разрядных числах, будем помнить, что n — количество разрядов, учитывающее также разряд знака.

единицы заменить нулями, а нули — единицами, после чего к младшему разряду прибавить единицу.

Для примера рассмотрим число X, которое в прямом коде имеет вид: $[X]_{п} = 1{,}01101010$. Тогда обратный код можно записать как $[X]_{о} = 1{,}10010101$. Для получения дополнительного кода прибавим 1 к младшему разряду обратного кода: $[X]_{д} = 1{,}10010101 + 0{,}00000001 = 1{,}10010110$.

Дополнительный код отрицательного числа легко получить непосредственно из прямого кода, воспользовавшись следующим формальным приемом.

Найти в прямом коде числа самую правую единицу (знаковый разряд при этом не учитывается). Эту единицу, а также цифры, расположенные справа от нее и знак числа, оставить неизменными. Во всех остальных позициях поменять единицы на нули, а нули на единицы.

Проиллюстрируем описанный прием примером того же числа X: $[X]_{п} = 1{,}01101010$. Отметим вертикальными линиями область между знаком числа, и самой правой единицей: 1,|011010|10. Внутри этой области поменяем единицы на нули, и наоборот, а вне области все оставим без изменений. Таким образом, получаем дополнительный код числа: 1,|100101|10. Убрав наши условные вертикальные линии, имеем $[X]_{д} = 1{,}10010110$ так же, как и при стандартном способе перевода.

При представлении двоичных чисел дополнительным кодом имеется только одна форма записи нуля 0,0...00, причем ноль считается положительным числом, так как его знаковый бит равен 0. Поскольку возможно только одно двоичное представление нуля, в нижней части диапазона представляемых чисел имеется еще одно отрицательное число -2^{n-1}, у которого нет положительного эквивалента. Таким образом, диапазон целых чисел, которые могут быть представлены дополнительным кодом, составляет от $-(2^{n-1})$ до $+(2^{n-1} - 1)$. Положительные числа в дополнительном коде записываются так же, как и в прямом.

Представленное в дополнительном коде n-разрядное двоичное число X можно преобразовать в m-разрядное, учитывая при этом следующие особенности.

Если $m > n$, то необходимо добавить к числу X слева $m - n$ битов, являющихся копиями знакового бита числа X. Иными словами, положительное число дополняется нулями, а отрицательное — единицами. Такое действие называется знаковым расширением. Ниже приводятся примеры знакового расширения при $n = 5$ и $m = 8$.

$$1{,}0101 \Rightarrow 1{,}1110101;\ 0{,}0101 \Rightarrow 0{,}0000101.$$

Если $m < n$, то нужно отбросить $m - n$ битов слева, однако результат будет правильным только в том случае, когда все отбрасываемые биты имеют то же значение, что и знаковый бит остающегося числа.

В большинстве ВМ используется представление чисел в дополнительном коде.

Сложение и вычитание чисел в обратном и дополнительном кодах

Преимущество дополнительного и обратного кодов состоит в том, что алгебраическое сложение этих кодов сводится к арифметическому. Это, в частности, позволяет

заменить операцию вычитания двоичных чисел $x_1 - x_2$ сложением с дополнениями $[x_1]_д + [-x_2]_д$ или $[x_1]_о + [-x_2]_о$.

При выполнении алгебраического сложения знаковый разряд и цифры модуля рассматриваются как единое целое и обрабатываются совместно. Особенность состоит в том, что перенос из старшего (знакового) разряда в обратном и дополнительном кодах учитывается по-разному. В случае обратного кода единица переноса из знакового разряда прибавляется к младшему разряду суммы (осуществляется так называемый циклический перенос). При использовании дополнительного кода единица переноса из знакового разряда отбрасывается.

Пример сложения чисел $x_1 = 0{,}10010001$ и $x_2 = -0{,}01100110$ в обратном коде приведен на рис. А.3, *а*, а в дополнительном коде — на рис А.3, *б*.

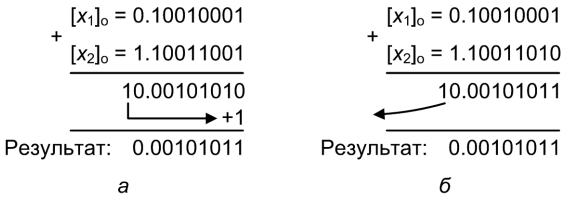


Рис. А.3. Пример сложения чисел: *а* — в обратном коде; *б* — в дополнительном коде

Если знаковый разряд результата равен нулю, это означает, что получено положительное число, которое выглядит так же, как в прямом коде. Единица в знаковом разряде означает, что результат отрицательный и его запись соответствует представлению в том коде, в котором производилась операция.

Чтобы избежать ошибок при выполнении сложения (вычитания), перед переводом чисел в обратные и дополнительные коды необходимо выровнять количество разрядов прямого кода операндов.

При сложении чисел, имеющих одинаковые знаки, могут быть получены числа, представление которых требует еще одного дополнительного разряда. Эта ситуация называется переполнением разрядной сетки. Для обнаружения переполнения в ВМ часто применяются *модифицированные прямой, обратный и дополнительный коды*. В этих кодах знак дублируется, причем знаку «плюс» соответствует комбинация 00, а знаку «минус» — комбинация 11.

Правила сложения для модифицированных кодов те же, что и для обычных. Единица переноса из старшего знакового разряда в модифицированном дополнительном коде отбрасывается, а в модифицированном обратном коде добавляется к младшему цифровому разряду.

Признаком переполнения служит появление в знаковых разрядах результата комбинации 01 при сложении положительных чисел (положительное переполнение) или 10 при сложении отрицательных чисел (отрицательное переполнение). Старший знаковый разряд в этих случаях содержит истинное значение знака суммы, а младший является старшей значащей цифрой числа.

Приложение Б

Логические основы вычислительных машин

Теоретическим фундаментом цифровой техники является *алгебра логики*, или *булева алгебра*, называемая так по имени ее основоположника Джорджа Буля, английского математика и логика середины XIX века.

В алгебре логики переменная может принимать одно из двух значений: *True* (истинно) и *False* (ложно). Эти значения в цифровой технике принято рассматривать как логическую «1» (True) и логический «0» (False), или как двоичные числа 1 и 0, и физически представлять присутствием или отсутствием некоторого сигнала.

Логические функции

Функция $f(x_1, x_2, ..., x_n)$, зависящая от n переменных, называется *функцией алгебры логики* (ФАЛ), если сама функция и любой из ее аргументов могут принимать значения только из множества {0, 1}. Другие названия ФАЛ: *логическая*, *булева* или *переключательная* функция. Совокупность значений аргументов функции алгебры логики называется *набором*, или точкой, и обозначается $< x_1, x_2, ..., x_n >$. Число возможных наборов из n аргументов равно 2^n. Аргументы булевой функции иногда называют булевыми.

Для описания логических функций используют один из нижеперечисленных способов.

Словесный способ. Здесь все случаи, при которых функция принимает значения 0 или 1, описываются словесно. Так, функцию «ИЛИ» со многими аргументами можно описать следующим образом: *функция принимает значение* 1, *если хотя бы один из аргументов принимает значение* 1, *иначе значение функции равно* 0.

Табличный способ. Булева функция $f(x_1,...,x_n)$ задается в виде *таблицы истинности* (соответствия). В левой части таблицы истинности записываются все возможные n-разрядные двоичные комбинации аргументов (табл. Б.1, *а*), а в

правой — значения функции на этих наборах. Таблица истинности содержит 2^n строк (по числу наборов аргументов), n столбцов по числу аргументов и один столбец значений функции.

Иногда вместо двоичных наборов аргументов в таблице истинности указывают их десятичные эквиваленты (табл. Б.1, *б*).

Таблица Б.1. Варианты представления таблицы истинности

x_1	x_2	x_3	f		Номер набора	f
0	0	0	0		0	0
0	0	1	1		1	1
0	1	0	0		2	0
0	1	1	0		3	0
1	0	0	1		4	1
1	0	1	1		5	1
1	1	0	0		6	0
1	1	1	1		7	1

а *б*

Числовой способ. Функция задается в виде последовательности десятичных эквивалентов тех наборов аргументов, на которых функция принимает значение 1. Например, двоичные наборы 010 и 101 имеют десятичные номера 2 и 5 соответственно. При таком подходе логическая функция трех аргументов, представленная в табл. Б.1, может быть записана в виде $f(1,4,5,7) = 1$. Та же булева функция может быть задана и по нулевым значениям: $f(0,2,3,6) = 0$.

Аналитический способ. Предполагает описание ФАЛ в виде алгебраического выражения, получаемого путем применения к аргументам операций булевой алгебры. Способ получения такого описания булевой функции будет рассмотрен в последующих разделах.

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

x_3 \ x_1x_2	00	01	11	10
0	0	0	0	1
1	1	0	1	1

Рис. Б.1. Таблица истинности и эквивалентная ей карта Карно

Координатный способ. При этом способе задания таблица истинности заменяется координатной картой состояний, известной под названием карты Карно. Такая карта содержит 2^n клеток по числу возможных наборов из n переменных.

Аргументы функции разбиваются на две группы так, что одна группа определяет координаты столбца, а другая — координаты строки, то есть каждой строке таблицы истинности (каждому набору аргументов) соответствует уникальная клетка карты. Внутри клетки записывается значение функции на данном наборе (рис. Б.1).

Графический, или **геометрический способ.** Булева функция $f(x_1,...,x_n)$ задается с помощью n-мерного куба, поскольку в геометрическом смысле каждый двоичный набор x_1, x_2,..., x_n есть n-мерный вектор, определяющий точку n-мерного пространства. По этой причине любой набор переменных в графическом представлении булевых функций принято называть вектором. Переменные куба называют координатами. Количество переменных в кубе определяет его мерность (3-мерный, ..., n-мерный).

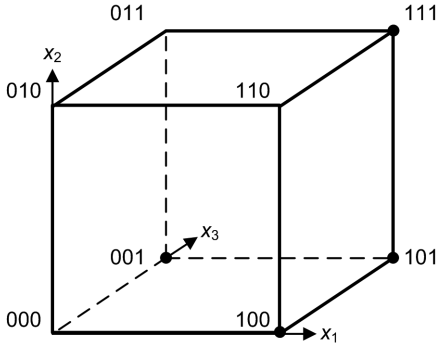


Рис. Б.2. Геометрическое представление булевой функции

Множество наборов, на которых определена функция n переменных, представляется вершинами n-мерного куба. Отметив точками те вершины куба, в которых функция принимает единичное (либо нулевое) значение, получаем геометрическое представление функции. Так, булева функция, приведенная в табл. Б.1, геометрически может быть представлена 3-мерным кубом (рис. Б.2).

Помимо перечисленных форм, иногда используются менее распространенные способы описания ФАЛ: комбинационной схемой, составленной из логических элементов; переключательной схемой; диаграммой Венна; диаграммой двоичного решения и т. д.

Булеву функцию, определенную на всех возможных наборах переменных, называют *полностью определенной*. Пример такой функции был показан в табл. Б.1. Булеву функцию n переменных называют *частично определенной*, или просто *частичной*, если она определена не на всех двоичных наборах длины n.

Наборы, на которых значение ФАЛ не определено, в таблице истинности обычно помечают каким-либо символом, например звездочкой (табл. Б.2).

Таблица Б.2. Таблица истинности для частично определенной булевой функции

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	*
0	1	1	0
1	0	0	1
1	0	1	*
1	1	0	0
1	1	1	*

Элементарные функции алгебры логики

В случае n переменных возможно не более 2^{2^n} различных булевых функций.

Элементарные функции одной переменной

В случае единственной переменной возможны 4 функции f_{10i} (x), показанные в табл. Б.3 и поясненные в табл. Б.4. При аналитическом описании ФАЛ наибольший интерес представляет функция логического отрицания f_{102}, согласно которой результат равен *True*, если значение аргумента было *False*, и наоборот.

Таблица Б.3. Логические функции одной переменной

Переменная	Логические функции			
x	f_{100}	f_{101}	f_{102}	f_{103}
0	0	0	1	1
1	0	1	0	1

Таблица Б.4. Описание логических функций одной переменной

Функция	Название	Обозначение
f_{100}	Константа нуля	$f_{100}(x) = 0$
f_{101}	Повторение x	$f_{101}(x) = x$
f_{102}	Логическое отрицание, инверсия (от лат. inversio – переворачиваю), «НЕ»	$f_{102}(x) = \overline{x};$ $f_{102}(x) = \neg x$
f_{103}	Константа единицы	$f_{103}(x) = 1$

Элементарные функции двух переменных

Для двух переменных можно сформировать 16 (и только 16) логических функций (табл. Б.5, Б.6).

Таблица Б.5. Логические функции двух переменных

Аргументы		Логические функции															
x_1	x_2	f_{200}	f_{201}	f_{202}	f_{203}	f_{204}	f_{205}	f_{206}	f_{207}	f_{208}	f_{209}	f_{210}	f_{211}	f_{212}	f_{213}	f_{214}	f_{215}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Функции двух переменных, рассмотренные в табл. Б.6, играют важную роль в алгебре логики и могут быть названы элементарными.

Таблица Б.6. Описание логических функций двух переменных

Функция	Название	Обозначение
f_{200}	Константа 0	$f_{200}(x_1, x_2) = 0 \;8;8\; False$
f_{201}	Логическое умножение, конъюнкция, «И»	$f_{201}(x_1, x_2) = x_1 \wedge x_2 \;8;8\; x_1 x_2$ $8;8\; x_1 \,\&\, x_2 \;8;8\; x_1 \; and \; x_2$
f_{202}	Запрет по x_2	$f_{202}(x_1, x_2) = x_1 \Delta x_2 \;8;8\; x_1 \wedge \overline{x_2}$
f_{203}	Переменная x_1[1]	$f_{203}(x_1, x_2) = x_1$
f_{204}	Запрет по x_1	$f_{204}(x_1, x_2) = x_2 \Delta x_1 \;8;8\; \overline{x_1} \wedge x_2$
f_{205}	Переменная x_2	$f_{205}(x_1, x_2) = x_2$
f_{206}	Сложение по модулю 2, отрицание эквивалентности, «Исключающее ИЛИ»	$f_{206}(x_1, x_2) = x_1 \oplus x_2 \;8;8\; x_1 \; xor \; x_2$ $8;8\; (\overline{x_1} \wedge x_2) \vee (x_1 \wedge \overline{x_2})$
f_{207}	Логическое сложение, дизъюнкция, «ИЛИ»	$f_{207}(x_1, x_2) = x_1 \vee x_2 \;8;8\; x_1 \quad x_2$ $8;8\; x_1 + x_2 \;8;8\; x_1 \; or \; x_2$
f_{208}	Функция Пирса (Вебба), «ИЛИ-НЕ»	$f_{208}(x_1, x_2) = x_1 \downarrow x_2 \;8;8\; \overline{x_1 \vee x_2}$ $8;8\; \overline{x_1} \wedge \overline{x_2}$
f_{209}	Логическая равнозначность, эквиваленция	$f_{209}(x_1, x_2) = x_1 \sim x_2 \;8;8\; x_1 \leftrightarrow x_2$ $8;8\; (\overline{x_1} \wedge \overline{x_2}) \vee (x_1 \wedge x_2)$ $8;8\; \overline{x_1 \oplus x_2}$
f_{210}	Отрицание x_1	$f_{210}(x_1, x_2) = \overline{x_2}$

[1] Если значение переменной в функции не влияет на значение функции, как в случае переменной x_2 в функции f_{203}, то такая переменная называется фиктивной (несущественной).

Таблица Б.6 (*продолжение*)

Функция	Название	Обозначение
f_{211}	Правая импликация	$f_{211}(x_1, x_2) = x_1 \vee \overline{x_2}$ 8; 8 $x_2 \rightarrow x_1$
f_{212}	Отрицание x_2	$f_{212}(x_1, x_2) = \overline{x_1}$
f_{213}	Левая импликация	$f_{213}(x_1, x_2) = \overline{x_1} \vee x_2$ 8; 8 $x_1 \rightarrow x_2$
f_{214}	Функция Шеффера, «И-НЕ»	$f_{214}(x_1, x_2) = x_1 \mid x_2$ 8; 8 $\overline{x_1 \wedge x_2}$ 8; 8 $\overline{x_1} \vee \overline{x_2}$
f_{215}	Константа 1	$f_{215}(x_1, x_2) = 1$ 8; 8 *True*

Рассмотренные элементарные функции позволяют строить более сложные булевы функции с помощью операции суперпозиции, заключающейся в подстановке в функцию новых функций вместо аргументов. Так, путем замены в функции $f_1(x_1, x_2)$ аргументов x_1 и x_2 на функции $f_2 = x_6$ и $f_3(x_5, x_7)$ получаем функцию $f_1(x_6, f_3(x_5, x_7))$. Суперпозиция функций двух аргументов дает возможность строить функции любого числа аргументов.

Суперпозиция булевых функций представляется в виде логических формул, причем следует учитывать, что одна и та же функция может быть представлена разными формулами, среди которых имеется и наиболее простая. Нахождение такой формулы составляет предмет минимизации ФАЛ, рассматриваемой ниже.

Правила алгебры логики

Как и любая другая математическая дисциплина, булева алгебра базируется на определенном своде правил, представленных в виде аксиом, теорем и законов (тождеств). Правила эти определяются для двух возможных логических значений «1» (True) и «0» (False), а также трех базовых логических операций: «НЕ», «И» и «ИЛИ», в силу чего основные правила алгебры логики формулируются, главным образом, для этих операций и их сочетаний. Базовые логические операции перечислены в порядке понижения их приоритетности. Учет приоритетности операций позволяет сократить число скобок при записи логических выражений.

Аксиомы алгебры логики

Как известно, аксиомами в математике принято называть предположения, не требующие доказательств.

4. **Аксиомы конъюнкции**

$$0 \wedge 0 = 0;\ 0 \wedge 1 = 0;\ 1 \wedge 0 = 0;\ 1 \wedge 1 = 1.$$

5. **Аксиомы дизъюнкции**

$$0 \vee 0 = 0;\ 0 \vee 1 = 1;\ 1 \vee 0 = 1;\ 1 \vee 1 = 1.$$

6. **Аксиомы отрицания**

$$\text{если } x = 0, \text{ то } \bar{x} = 1; \text{ если } x = 1, \text{ то } \bar{x} = 0.$$

Теоремы алгебры логики

Для доказательства теорем булевой алгебры используется простая подстановка значений булевых переменных. Это обусловлено тем, что переменные могут принимать только два значения — «0» и «1».

7. **Теоремы исключения констант**

$$x \vee 1 = 1; x \vee 0 = x; x \wedge 1 = x; x \wedge 0 = 0.$$

8. **Теоремы идемпотентности (тавтологии, повторения)**

$$x \vee x = x; x \wedge x = x.$$

Для n переменных:

$$x \vee x \vee ... \vee x = x; x \wedge x \wedge ... \wedge x = x.$$

Повторное применение не дает ничего нового.

9. **Теорема противоречия**

$$x \wedge \bar{x} = 0.$$

Невозможно, чтобы противоречащие высказывания были одновременно истинными.

10. **Теорема «исключенного третьего»**

$$x \vee \bar{x} = 1.$$

Из двух противоречивых высказываний об одном и том же предмете одно всегда истинно, а второе — ложно, третьего не дано.

11. **Теорема двойного отрицания (инволюции)**

$$\bar{\bar{x}} = x.$$

Двойное отрицание отменяет инверсию.

Законы алгебры логики

Излагаемые ниже правила обычно называют законами, или тождествами булевой алгебры.

12. **Ассоциативный (сочетательный) закон**

$$x_1 \vee (x_2 \vee x_3) = (x_1 \vee x_2) \vee x_3; x_1 \wedge (x_2 \wedge x_3) = (x_1 \wedge x_2) \wedge x_3.$$

При одинаковых знаках скобки можно ставить произвольно или вообще опускать.

10. **Коммутативный (переместительный) закон**

$$x_1 \vee x_2 = x_2 \vee x_1; x_1 \wedge x_2 = x_2 \wedge x_1.$$

Результат операции над высказываниями не зависит от того, в каком порядке берутся эти высказывания.

11. **Дистрибутивный (распределительный) закон**

$$(x_1 \vee x_2) \wedge x_3 = (x_1 \wedge x_3) \vee (x_2 \wedge x_3);$$
$$(x_1 \wedge x_2) \vee x_3 = (x_1 \vee x_3) \wedge (x_2 \vee x_3).$$

Закон определяет правило выноса общего высказывания за скобку.

12. **Законы де Моргана (законы общей инверсии или дуальности)**

$$\overline{x_1} \vee \overline{x_2} = \overline{x_1 \wedge x_2}\ ;\ x_1 \vee x_2 = \overline{\overline{x_1} \wedge \overline{x_2}}\ ;$$
$$\overline{x_1} \wedge \overline{x_2} = \overline{x_1 \vee x_2}\ ;\ x_1 \wedge x_2 = \overline{\overline{x_1} \vee \overline{x_2}}\ .$$

Расширенный закон де Моргана:

$$\overline{x_1} \vee \overline{x_2} \vee \ldots \vee \overline{x_n} = \overline{x_1 \wedge x_2 \wedge \ldots \wedge x_n}\ ;$$
$$\overline{x_1} \wedge \overline{x_2} \wedge \ldots \wedge \overline{x_2} = \overline{x_1 \vee x_2 \vee \ldots \vee x_n}\ .$$

Законы де Моргана можно рассматривать как частный случай теоремы Шеннона, которая утверждает, что инверсия любой функции в алгебре логики получается путем замены каждой переменной ее инверсией и одновременно взаимной заменой символов конъюнкции и дизъюнкции.

13. **Закон поглощения (элиминации)**

$$x_1 \vee (x_1 \wedge x_2) = x_1\ ;\ x_1 \wedge (x_1 \vee x_2) = x_1\ .$$

14. **Закон склеивания (исключения)**

$$(x_1 \wedge x_2) \vee (x_1 \wedge \overline{x_2}) = x_1\ ;\ (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) = x_1\ .$$

Справедливость приведенных законов можно доказать табличным способом: выписать все наборы значений x_1 и x_2, вычислить на них значения левой и правой частей доказываемого выражения и убедиться, что результирующие столбцы совпадут.

В дальнейшем для облегчения восприятия приводимых логических выражений знак конъюнкции будем опускать и записывать логическое произведение как обычное, то есть выражение типа $x_1 \wedge x_2 \wedge x_3$ будем писать в виде $x_1\, x_2\, x_3$. В ряде случаев, чтобы избежать слияния знаков отрицания, между переменными может вставляться точка, например $x_1\overline{x_2} \cdot \overline{x_3}$.

Дополнительные тождества алгебры логики

В данном разделе без доказательства приводятся некоторые дополнительные тождества алгебры логики, которые могут оказаться полезными при минимизации ФАЛ. Некоторые из них представляют собой лишь иную форму записи ранее рассмотренных тождеств.

$$x_1 \vee (\overline{x_1} \wedge x_2) = x_1 \vee x_2\ ;$$

$$x_1 \wedge (\overline{x_1} \vee x_2) = x_1 \wedge x_2\ ;$$

$$x_1x_2 \vee \overline{x_1}x_3 \vee x_2x_3 = x_1x_2 \vee \overline{x_1}x_3;$$

$$x_1(\overline{x_1} \vee x_2) = x_1x_2;$$

$$(x_1 \vee x_2)(\overline{x_1} \vee x_3) = x_1x_3 \vee \overline{x_1}x_2;$$

$$(x_1 \vee x_2)(\overline{x_1} \vee x_3)(x_2 \vee x_3) = (x_1 \vee x_2)(\overline{x_1} \vee x_3).$$

Логический базис

Любую булеву функцию с произвольным количеством аргументов можно построить через суперпозицию элементарных логических функций одной и двух переменных. Набор простейших функций, с помощью которого можно выразить любые другие, сколь угодно сложные логические функции, называется *функционально полным набором*, или *логическим базисом*. На практике в качестве базисов обычно используются следующие системы логических функций:

- «НЕ», «И», «ИЛИ» — булев базис;
- «И-НЕ» — универсальный базис Шеффера;
- «ИЛИ-НЕ» — универсальный базис Пирса;
- «Исключающее ИЛИ», «И», «Константа 1» — базис Жегалкина;
- «Запрет по X_2», «1».

Логический базис называется *минимальным*, если удаление хотя бы одной из входящих в него функций превращает этот набор в функционально неполный. Логический базис «И», «ИЛИ», «НЕ» не является минимальным, так как с помощью формул де Моргана можно исключить из логических выражений либо функцию «И», либо функцию «ИЛИ». В результате получаются минимальные базисы: «И», «НЕ» и «ИЛИ», «НЕ». Некоторые функции сами по себе представляют собой минимальный логический базис. К таким, например, относятся функции «И-НЕ» и «ИЛИ-НЕ». Чаще всего для записи логических выражений и их последующего преобразования используется базис «И», «ИЛИ», «НЕ».

Аналитическое представление булевых функций

В качестве исходного описания сложных логических функций обычно используется таблица истинности, однако упрощение функций выгоднее производить в аналитической форме. При аналитической записи ФАЛ представляется либо в виде логической суммы элементарных логических произведений (дизъюнкции элементарных конъюнкций), либо в виде логического произведения элементарных логических сумм (конъюнкции элементарных дизъюнкций). Первая форма записи носит название дизъюнктивной нормальной формы (ДНФ), вторая — конъюнктивной нормальной формы (КНФ).

Сначала определим основные понятия и терминологию, используемые при аналитическом представлении ФАЛ.

Элементарной конъюнкцией (элементарным логическим произведением) называют логическое произведение любого количества переменных (аргументов), взятых с отрицанием или без, но каждый из аргументов встречается в этом произведении лишь однократно. Элементарная конъюнкция, включающая все без исключения аргументы ФАЛ, носит название *полной элементарной конъюнкции*.

Дизъюнктивная нормальная форма (ДНФ) — это логическая сумма элементарных логических произведений (дизъюнкция элементарных конъюнкций). Термин «нормальная» означает, что в выражении отсутствуют групповые инверсии, то есть общий знак отрицания над несколькими переменными сразу, например, $\overline{x_1 x_2}$. Примером ДНФ может служить выражение $x_1 x_2 \vee \overline{x_1} x_2 x_3 \vee x_1 \overline{x_2} \cdot \overline{x_3}$.

Минтерм (единичный набор функции) — это логическое произведение всех переменных, взятых с отрицанием или без (полная элементарная конъюнкция), соответствующее набору аргументов, на которых функция принимает значение «1». Каждый минтерм соответствует одной строке таблице истинности, причем только такой, где значение функции равно 1 (True). Множество всех минтермов образует *единичное множество функции*.

Совершенной дизъюнктивной нормальной формой (СДНФ) называется ДНФ, состоящая из всех минтермов булевой функции. Для получения СДНФ на основе таблицы истинности необходимо:

- каждый из входных наборов, на которых булева функция принимает значение «1», представить в виде элементарного произведения (конъюнкции), причем если переменная равна 0, то она входит в конъюнкцию с инверсией, а если 1 — то без инверсии;
- полученные элементарные логические произведения объединить знаками логического сложения (дизъюнкции).

Число элементарных произведений (минтермов) в СДНФ равно числу строк таблицы истинности, на которых функция имеет значение «1». Для получения минтерма в него нужно включить все аргументы, причем те из них, которые имеют в данном наборе нулевое значение, входят в минтерм со знаком отрицания. Так, таблице истинности

x_1	x_2	f
0	0	1
0	1	1
1	0	1
1	1	0

отвечает совершенная дизъюнктивная нормальная форма $f = \overline{x_1} \cdot \overline{x_2} \vee \overline{x_1} x_2 \vee x_1 \overline{x_2}$.

Элементарная дизъюнкция (элементарная логическая сумма) — это логическая сумма (дизъюнкция) любого числа переменных, взятых с отрицанием или без, причем каждый отдельный аргумент встречается лишь однократно. Элементарная дизъюнкция, включающая все без исключения аргументы ФАЛ, называется *полной элементарной дизъюнкцией*.

Конъюнктивная нормальная форма (КНФ) — это логическое произведение элементарных логических сумм (конъюнкция элементарных дизъюнкций). Термин «нормальная» означает то же, что и в случае ДНФ, то есть отсутствие общей инверсии над несколькими переменными сразу, например $\overline{x_1 \vee x_2 \vee x_3}$. В качестве примера КНФ можно привести выражение $(x_1 \vee x_2)(\overline{x_1} \vee x_2 \vee x_3)(x_1 \vee \overline{x_2} \vee \overline{x_3})$.

Макстерм (нулевой набор функции) — это логическая сумма (дизъюнкция) всех аргументов (полная элементарная дизъюнкция), соответствующая набору аргументов, на которых функция принимает значение «0». Множество нулевых наборов называют *нулевым множеством функции*.

Совершенной конъюнктивной нормальной формой (СКНФ) называется КНФ, состоящая из всех макстермов булевой функции. Для получения СКНФ на основе таблицы истинности необходимо:

- каждый из входных наборов, на которых булева функция принимает значения «0», представить в виде элементарной логической суммы (дизъюнкции), причем если переменная равна 1, то она входит в эту сумму с инверсией, а если 0 — то без инверсии;
- полученные элементарные логические суммы объединить знаками логического умножения.

Число элементарных дизъюнкций в СКНФ равно числу строк таблицы истинности, на которых функция имеет значение 0. Для получения макстерма в него нужно включить все аргументы, причем те из них, которые имеют в данном наборе единичное значение, входят в элементарную дизъюнкцию со знаком отрицания. Так, таблице истинности

x_1	x_2	f
0	0	0
0	1	0
1	0	1
1	1	0

отвечает СКНФ $f = (x_1 \vee x_2)(x_1 \vee \overline{x_2})(\overline{x_1} \vee \overline{x_2})$.

Как минтермы, так и макстермы часто определяют общим термином «*терм*». Из вышесказанного следует общее правило установки знака отрицания над переменной терма (как минтерма, так и макстерма):

если значение переменной в таблице истинности отличается от значения логической функции, над такой переменной в терме ставится знак отрицания, в противном же случае знак отрицания не ставится.

Для перехода от таблицы истинности к эквивалентному аналитическому описанию используются совершенные ДНФ и КНФ.

Иногда удобнее пользоваться не самой логической функцией, а ее инверсией. В этом случае для записи СДНФ надо использовать нулевые, а для записи СКНФ — единичные значения функции.

По целому ряду причин более распространенной формой представления ФАЛ является СДНФ, поэтому в дальнейшем основное внимание будет уделено именно этой форме аналитической записи булевых функций.

Минимизация логических функций

Проблема минимизации логических выражений проистекает из практических задач создания логических схем. В качестве исходной аналитической формы обычно рассматривают совершенные ДНФ и КНФ, которые во многих случаях оказываются излишне сложными, из-за чего их техническая либо программная реализация получается избыточной. Для упрощения СДНФ и СКНФ используются различные *методы минимизации* — преобразования логической функции с целью упрощения ее аналитической записи.

К настоящему времени применяются следующие методы минимизации ФАЛ:

1) расчетный метод (метод непосредственных преобразований);
2) расчетно-табличный метод (метод Квайна);
3) метод Квайна–Мак-Класки (развитие метода Квайна);
4) метод Петрика (развитие метода Квайна);
5) табличный метод (метод карт Карно);
6) метод Блейка–Порецкого;
7) метод неопределенных коэффициентов;
8) метод гиперкубов;
9) метод факторизации;
10) метод функциональной декомпозиции и др.

Ниже будут рассмотрены первые пять методов, получившие наиболее широкое распространение.

Минимальный вариант булевой функции обычно ищут применительно к какому-либо логическому базису. Наилучшие результаты получаются с функциями «НЕ», «И», «ИЛИ», в силу чего именно эти функции в дальнейшем будем использовать в качестве основного логического базиса.

Сложность реализации логического выражения обычно характеризуют с помощью *коэффициента сложности* K_c. Для вычисления этого коэффициента нужно сложить количество термов, образующих выражение (слагаемых в ДНФ или сомножителей в КНФ), и сумму рангов всех этих термов. *Ранг терма* равен количеству переменных в терме, например, ранг терма $x_1 x_2 x_3$ равен трем. Следовательно, для функции $f = x_4 x_1 \vee x_3 x_2 \vee \overline{x_3} x_1 \vee \overline{x_4} x_3 \overline{x_2}$, содержащей 4 терма (три терма с рангом 2 и один терм с рангом 3), коэффициент сложности равен $K_c = 4 + (2 + 2 + 2 + 3) = 13$.

Целью минимизации является получение такой аналитической записи ФАЛ, которая имеет наименьший коэффициент сложности среди всех других эквивалентных вариантов записи данной функции. Подобную аналитическую запись логической функции называют *минимальной*, то есть можно сказать, что целью минимизации является получение *минимальной дизъюнктивной нормальной формы* (МДНФ) или *минимальной конъюнктивной нормальной формы* (МКНФ).

В основе практически всех методов минимизации лежит операция склеивания. Два элементарных произведения одинакового ранга (для ДНФ) или две элементарные суммы одинакового ранга (для КНФ) склеиваются, если они различаются только по одной переменной, а это различие состоит в присутствии знака инверсии над этой переменной в одном из произведений (сумм) и в его отсутствии в другом.

Прежде чем перейти к рассмотрению методов минимизации ФАЛ, приведем используемые в этом случае положения и определения.

Элементарные логические произведения, образовавшиеся в результате склеивания, называют *импликантами*. Склеивания начинаются с минтермов и продолжаются с импликантами, полученными в предшествующих операциях склеивания. Так, если СДНФ описывается выражением $f = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} \vee \overline{x_1} \cdot \overline{x_2} x_3 \vee \overline{x_1} x_2 \overline{x_3} \vee \overline{x_1} x_2 x_3$, то после склеивания минтермов $\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3}$ и $\overline{x_1} \cdot \overline{x_2} x_3$ получим импликанту $\overline{x_1} \cdot \overline{x_2}$, которая поглощает участвовавшие в склеивании минтермы. Склеивание другой пары минтермов — $\overline{x_1} x_2 \overline{x_3}$ и $\overline{x_1} x_2 x_3$ дает импликанту $\overline{x_1} x_2$, также поглощающую минтермы, из которых она была получена. В результате исходная ДНФ приобретает следующий вид: $f = \overline{x_1} \cdot \overline{x_2} \vee \overline{x_1} x_2$. Конъюнкции в правой части выражения (обратим внимание, что это уже не минтермы, а импликанты) также можно склеить, получив при этом импликанту $\overline{x_1}$. Импликанты, дальнейшее склеивание которых невозможно, называют *простыми*, или *первичными импликантами*. В нашем примере $\overline{x_1}$ является простой импликантой.

Выражение, полученное из совершенной ДНФ и состоящее только из простых импликант, носит название *сокращенной дизъюнктивной нормальной формы*.

Следующим этапом минимизации является нахождение тупиковых ДНФ. *Тупиковой дизъюнктивной нормальной формой* называется ДНФ, полученная из сокращенной ДНФ, в результате исключения из последней всех лишних простых импликант. В зависимости от того, какие из простых импликант были признаны лишними, может получиться несколько вариантов тупиковых ДНФ.

Тупиковая ДНФ, имеющая наименьший коэффициент сложности по сравнению с другими тупиковыми формами данной функции, носит название *минимальной дизъюнктивной нормальной формы* (МДНФ). Если среди возможных тупиковых форм имеется несколько с одинаковым коэффициентом сложности, это означает, что существует и несколько МДНФ.

Для приведенных выше понятий имеются аналоги, относящиеся к конъюнктивным нормальным формам. Приведем их.

Имплицентой называется элементарная логическая сумма, получаемая при склеивании пары макстермов или имплицент, образовавшихся в результате предшествующих склеиваний.

Простая имплицента — это имплицента, которую уже нельзя склеить ни с одной другой.

Сокращенная конъюнктивная нормальная форма — это конъюнкция всех простых имплицент.

Тупиковая конъюнктивная нормальная форма — это логическое произведение простых имплицент, из которых ни одна не является лишней.

Минимальная конъюнктивная нормальная форма (МКНФ) — тупиковая КНФ, имеющая наименьший коэффициент сложности среди других тупиковых форм данной ФАЛ.

Аналитическая минимизация производится в такой последовательности (описывается только применительно к ДНФ, поскольку для КНФ процедура аналогична):

- находят сокращенную дизъюнктивную нормальную форму (любая функция имеет только одну такую форму);
- находят все тупиковые нормальные формы;
- из полученных тупиковых форм выбирают минимальные формы.

Минимизация методом непосредственных преобразований

Исходной формой для этого метода минимизации служит СДНФ или СКНФ. Непосредственные преобразования логических формул служат для упрощения формул или приведения их к определенному виду путем использования основных правил алгебры логики. Некоторые преобразования логических формул похожи на преобразования формул в обычной алгебре (вынесение общего множителя за скобки, использование переместительного и сочетательного законов и т. п.), тогда как другие преобразования основаны на свойствах, которыми не обладают операции обычной алгебры (использование распределительного закона для конъюнкции, законов поглощения, склеивания, де Моргана и др.).

Минимизацию обычно проводят в такой последовательности (описывается только процесс минимизации СДНФ, поскольку минимизация СКНФ производится по аналогичной схеме). Сначала ищутся пары минтермов, отличающихся друг от друга только знаком инверсии и лишь в одном из аргументов. Такие минтермы склеиваются. В результате склеивания из двух минтермов образуется импликанта, ранг которой на единицу меньше, чем у склеиваемых минтермов:

$$x_1\overline{x_2}x_3 \vee \overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3} \vee x_1x_2x_3 = x_1x_3(\overline{x_2} \vee x_2) \vee \overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3} = x_1x_3 \vee \overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}.$$

С импликантами, образовавшимися в результате первого этапа склеивания, по возможности проводится очередной этап склеивания. Процесс продолжают до тех пор, пока дальнейшее склеивание импликант становится невозможным, то есть до получения простых импликант. Выражение, образованное только из простых импликант, как уже отмечалось, носит название сокращенной дизъюнктивной нормальной формы.

Далее предпринимается попытка исключить из сокращенной ДНФ избыточные простые импликанты, используя для этого прочие правила булевой алгебры, например теоремы противоречия, «исключенного третьего», закон поглощения и т. д. В зависимости от применяемых правил и порядка их использования может получиться несколько вариантов нормальных форм, в которых ни одну из входящих в них простых импликант уже нельзя исключить (тупиковых ДНФ).

Наконец, из тупиковых форм выбирается та, которая имеет наименьший коэффициент сложности (содержит минимальное суммарное количество букв и термов). Это и будет минимальная дизъюнктивная нормальная форма. Если минимальный коэффициент сложности одновременно имеет несколько тупиковых форм, то имеют место и несколько МДНФ.

Пример 1. Функция трех аргументов f задана таблицей истинности:

x_1	x_2	x_3	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$$f = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} \vee \overline{x_1} \cdot \overline{x_2} x_3 \vee \overline{x_1} x_2 \overline{x_3} \vee \overline{x_1} x_2 x_3 = \overline{x_1} \cdot \overline{x_2} (\overline{x_3} \vee x_3) \vee \overline{x_1} x_2 (\overline{x_3} \vee x_3) =$$
$$= \overline{x_1} \cdot \overline{x_2}(1) \vee \overline{x_1} x_2 (1) = \overline{x_1} \cdot \overline{x_2} \vee \overline{x_1} x_2 = \overline{x_1} (\overline{x_2} \vee x_2) = \overline{x_1}(1) = \overline{x_1}.$$

Очевидно, что полученное выражение является минимальной дизъюнктивной нормальной формой.

Пример 2. Функция четырех аргументов представлена в табл. Б.7.

Таблица Б.7. Таблица истинности к примеру

Номер набора	x_1	x_2	x_3	x_4	f
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

Минимизация функции может быть осуществлена следующим образом:

$$f = \overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}\cdot\overline{x_4} \vee \overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}x_4 \vee \overline{x_1}\cdot\overline{x_2}x_3\overline{x_4} \vee \overline{x_1}x_2\overline{x_3}x_4 \vee \overline{x_1}x_2x_3\overline{x_4} \vee \overline{x_1}x_2x_3x_4 \vee$$
$$\vee x_1\overline{x_2}\cdot\overline{x_3}\cdot\overline{x_4} \vee x_1\overline{x_2}\cdot\overline{x_3}x_4 \vee x_1\overline{x_2}x_3\overline{x_4} \vee x_1x_2x_3\overline{x_4} =$$
$$= (\overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}\cdot\overline{x_4} \vee \overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}x_4) \vee (\overline{x_1}\cdot\overline{x_2}x_3\overline{x_4} \vee \overline{x_1}x_2x_3x_4) \vee$$
$$\vee (\overline{x_1}x_2\overline{x_3}x_4 \vee \overline{x_1}x_2x_3x_4) \vee (x_1\overline{x_2}\cdot\overline{x_3}\cdot\overline{x_4} \vee x_1\overline{x_2}\cdot\overline{x_3}x_4) \vee (x_1\overline{x_2}x_3\overline{x_4} \vee x_1x_2x_3\overline{x_4}) =$$
$$= \overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3}(\overline{x_4} \vee x_4) \vee \overline{x_1}x_3\overline{x_4}(\overline{x_2} \vee x_2) \vee \overline{x_1}x_2x_4(\overline{x_3} \vee x_3) \vee$$
$$\vee x_1\overline{x_2}\cdot\overline{x_3}(\overline{x_4} \vee x_4) \vee x_1x_3\overline{x_4}(\overline{x_2} \vee x_2) =$$
$$= \overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3} \vee \overline{x_1}x_3\overline{x_4} \vee \overline{x_1}x_2x_4 \vee x_1\overline{x_2}\cdot\overline{x_3} \vee x_1x_3\overline{x_4} =$$
$$= \overline{x_2}\cdot\overline{x_3}(\overline{x_1} \vee x_1) \vee \overline{x_1}x_2x_4 \vee x_3\overline{x_4}(\overline{x_1} \vee x_1) =$$
$$= \overline{x_2}\cdot\overline{x_3} \vee \overline{x_1}x_2x_4 \vee x_3\overline{x_4} = \overline{x_2}\cdot\overline{x_3} \vee x_3\overline{x_4} \vee \overline{x_1}x_2x_4.$$

Получена тупиковая ДНФ, которая в данном случае совпадает с минимальной.

Ниже приводятся некоторые приемы и способы, которые могут быть полезными при упрощении логических выражений:

$$\overline{x_0 \vee x_1}(x_0\overline{x_1}) = 0;$$

$$\overline{x_0}x_1 \vee \overline{x_0 \vee x_1} \vee x_0 = 1;$$

$$(x_0 \vee x_1)(\overline{x_0} \vee x_1)(\overline{x_0} \vee \overline{x_1}) = \overline{x_0}x_1;$$

$$x_0\overline{x_1} \vee \overline{x_0}x_1x_2 \vee x_0x_2 = x_0\overline{x_1} \vee x_1x_2;$$

$$\overline{x_0x_1 \vee \overline{x_2}} = (\overline{x_0} \vee \overline{x_1})x_2$$

$$x_0x_1 \vee x_0x_1x_2 \vee x_0x_2x_3 = x_0(x_1 \vee x_2x_3);$$

$$x_0 \vee \overline{x_1x_2} \vee \overline{\overline{x_0} \vee x_1 \vee x_2} = x_0 \vee x_2 \vee \overline{x_1};$$

$$x_0x_1 \vee x_0x_1x_2 \vee x_0x_1x_2 \vee x_0x_1x_2 = x_0;$$

$$(x_0x_1 \vee x_2)(x_0 \vee x_1) \vee x_2 = x_0 \vee x_2 \vee x_1$$

$$x_0x_1(x_0x_2 \vee x_0x_1x_2 \vee x_2x_3) = x_0x_1.$$

При упрощении ФАЛ не всегда ясно, какое из правил алгебры логики следует применить на том или ином шаге. Навыки приходят с опытом.

Минимизация по методу Квайна

Метод Квайна (Willard van Orman Quine 1908–2000) по своей сути идентичен как по отношению к ДНФ, так и к КНФ, поэтому рассмотрим его на примере дизъюнктивной формы. В основе метода лежит использование двух основных законов алгебры логики — закона склеивания и закона поглощения. Процедура минимизации

проводится в несколько этапов (рассмотрим их на примере функции, приведенной в табл. Б.7):

1. **Нахождение простых импликант**

Перебирают все пары минтермов исходной СДНФ, склеивая те из них, для которых эта операция возможна. При составлении пар любой из минтермов может быть использован многократно. Процедура повторяется и над полученными импликантами — опять проводятся операции склеивания. Процесс повторяется до тех пор, пока не остается ни одной импликанты, допускающей склеивания с другими. Напомним, что такие импликанты называются простыми. Участвовавшие в склеивании элементарные конъюнкции помечаются каким-либо символом, например «*». Наличие такого символа означает, что данная элементарная конъюнкция уже учтена в какой-то из импликант, полученных в результате склеивания, и поглощается последней. Если какой-либо из минтермов изначально не удалось склеить ни с одним другим, то он уже сам по себе является простой импликантой. Дизъюнктивная нормальная форма, составленная из всех простых импликант, полученных в результате описанной процедуры, представляет собой сокращенную ДНФ, эквивалентную исходной СДНФ. Данный этап иллюстрирует табл. Б.8 и Б.9.

В табл. Б.8 показаны минтермы исходной совершенной ДНФ. Анализируются все возможные пары минтермов и, если это возможно, производится их склеивание. Минтермы, участвовавшие в операции склеивания, помечаются символом «*». Результаты склеивания с указанием номеров «склеенных» минтермов показаны в табл. Б.9.

Таблица Б.8. Минтермы совершенной ДНФ

Номер набора	Минтермы	Номер набора	Минтермы
0	$\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} \cdot \overline{x_4}$ *	7	$\overline{x_1} x_2 x_3 x_4$ *
1	$\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} x_4$ *	8	$x_1 \overline{x_2} \cdot \overline{x_3} \cdot \overline{x_4}$ *
2	$\overline{x_1} \cdot \overline{x_2} x_3 \overline{x_4}$ *	9	$x_1 \overline{x_2} \cdot \overline{x_3} x_4$ *
5	$\overline{x_1} x_2 \overline{x_3} x_4$ *	10	$x_1 \overline{x_2} x_3 \overline{x_4}$ *
6	$\overline{x_1} x_2 x_3 \overline{x_4}$ *	14	$x_1 x_2 x_3 \overline{x_4}$ *

Таблица Б.9. Склеивание минтермов совершенной ДНФ

Склеиваемые наборы	Импликанты	Склеиваемые наборы	Импликанты
0,1	$\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3}$ *	5,7	$\overline{x_1} x_2 x_4$
0,2	$\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_4}$ *	6,7	$\overline{x_1} x_2 x_3$
0,8	$\overline{x_2} \cdot \overline{x_3} \cdot \overline{x_4}$ *	6,14	$x_2 x_3 \overline{x_4}$ *

Таблица Б.9 (*продолжение*)

Склеиваемые наборы	Импликанты	Склеиваемые наборы	Импликанты
1,5	$\overline{x_1} \cdot \overline{x_3} x_4$	8,9	$x_1 \overline{x_2} \cdot \overline{x_3}$ *
1,9	$\overline{x_2} \cdot \overline{x_3} x_4$ *	8,10	$x_1 \overline{x_2} \cdot \overline{x_4}$ *
2,6	$\overline{x_1} x_3 \overline{x_4}$ *	10,14	$x_1 x_3 \overline{x_4}$ *
2,10	$\overline{x_2} x_3 \overline{x_4}$ *		

В результате первого этапа склеивания получены 13 импликант ранга 3. Поскольку помеченными оказались все минтермы, среди них простых импликант нет, тем самым в 13 импликантах содержится вся исходная информация, и ДНФ, составленная из этих импликант, полностью эквивалентна исходной совершенной ДНФ. Далее повторим процедуру попарного склеивания применительно к полученным 13 импликантам (табл. Б.10).

Таблица Б.10. Склеивание импликант ранга 3

Склеиваемые наборы	Импликанты
0,1,8,9	$\overline{x_2} \cdot \overline{x_3}$
0,2,8,10	$\overline{x_2} \cdot \overline{x_4}$
0,8,1,9	$\overline{x_2} \cdot \overline{x_3}$
0,8,2,10	$\overline{x_2} \cdot \overline{x_4}$
2,6,10,14	$x_3 \overline{x_4}$
2,10,6,14	$x_3 \overline{x_4}$

Из таблицы видно, что импликанты, обозначенные как 1,5; 5,7 и 6,7, остались непомеченными. Это означает, что они не были склеены ни с одной другой импликантой, поэтому являются простыми и должны учитываться на последующих этапах минимизации. В ходе склеивания образовались три пары импликант ранга 2. В соответствии с теоремой идемпотентности из нескольких одинаковых импликант можно оставить только одну. Нетрудно заметить, что дальнейшее склеивание трех оставшихся импликант ранга 2 невозможно, то есть эти импликанты — простые. Таким образом, в дополнение к трем простым импликантам ранга 3: $\overline{x_1} \cdot \overline{x_3} x_4$, $\overline{x_1} x_2 x_4$ и $\overline{x_1} x_2 x_3$ получены еще три простые импликанты ранга 2: $\overline{x_2} \cdot \overline{x_3}$, $\overline{x_2} \cdot \overline{x_4}$ и $x_3 \overline{x_4}$. Логическая сумма перечисленных простых импликант представляет собой сокращенную ДНФ: $f = \overline{x_1} \cdot \overline{x_3} x_4 \vee \overline{x_1} x_2 x_4 \vee \overline{x_1} x_2 x_3 \vee \overline{x_2} \cdot \overline{x_3} \vee \overline{x_2} \cdot \overline{x_4} \vee x_3 \overline{x_4}$.

2. **Составление импликантной матрицы и расстановка меток избыточности**

Этот и последующие шаги имеют целью убрать из сокращенной ДНФ все лишние простые импликанты, то есть перейти от сокращенной ДНФ к тупиковым формам, а затем и к минимальным. Задача решается с помощью специальной *импликантной матрицы*. Каждая строка такой матрицы соответствует одной из простых импликант, входящих в сокращенную ДНФ, иными словами, количество строк в матрице равно числу простых импликант в сокращенной ДНФ. Столбцы матрицы представляют минтермы исходной СДНФ, при этом каждому минтерму соответствует свой столбец.

Если минтерм в столбце импликантной матрицы содержит в себе простую импликанту из какой-либо строки матрицы, то на пересечении данного столбца и данной строки ставится метка избыточности. Это означает, что данная простая импликанта поглощает соответствующий минтерм и способна заменить его в окончательном логическом выражении.

Таблица Б.11. Импликантная матрица Квайна

		0	1	2	5	6	7	8	9	10	14
0,1,8,9	$\overline{x_2} \cdot \overline{x_3}$	√	√					√	√		
0,2,8,10	$\overline{x_2} \cdot \overline{x_4}$	√		√				√		√	
2,6,10,14	$x_3 \overline{x_4}$			√		√				√	√
1,5	$\overline{x_1} \cdot \overline{x_3} x_4$		√		√						
5,7	$\overline{x_1} x_2 x_4$				√		√				
6,7	$\overline{x_1} x_2 x_3$					√	√				

Импликантная матрица для рассматриваемого примера (табл. Б.11) содержит 6 строк (по числу простых импликант) и 10 столбцов (по числу минтермов исходной СДНФ). В матрице минтермы обозначены своими номерами, а слева от простых импликант перечислены номера минтермов, из которых эти импликанты были получены. Расставим в ней метки в тех позициях, где простая импликанта, указанная в левом столбце, покрывает минтерм, записанный в верхней строке.

3. **Нахождение существенных импликант и исключение связанных с ними строк и столбцов**

Присутствие в столбце только одной метки означает, что простая импликанта строки, где стоит эта метка, является *существенной*, или *базисной импликантой*, то есть обязательно войдет в минимальную ДНФ. Строка, содержащая существенную импликанту, а также столбцы, на пересечении с которыми в этой строке стоит метка

избыточности, вычеркиваются. Это позволяет упростить последующие шаги минимизации. Если после упомянутого вычеркивания в оставшейся части таблицы появятся строки, не содержащие меток или содержащие идентично расположенные метки, то такие строки также вычеркиваются. В последнем случае оставляют одну — ту, в которой простая импликанта имеет наименьший ранг среди остальных вычеркиваемых импликант.

Таблица Б.12. Удаление из импликантной матрицы существенных импликант и покрываемых ими минтермов

		0	1	2	5	6	7	8	9	10	14
0,1,8,9	$\overline{x_2} \cdot \overline{x_3}$	√	√					√	√		
0,2,8,10	$\overline{x_2} \cdot \overline{x_4}$	√		√				√		√	
2,6,10,14	$x_3 \overline{x_4}$			√		√				√	√
1,5	$\overline{x_1} \cdot \overline{x_3} x_4$		√		√						
5,7	$\overline{x_1} x_2 x_4$				√		√				
6,7	$\overline{x_1} x_2 x_3$					√	√				

В нашей функции по одной метке имеют столбцы 9 и 14, следовательно, имеют место две существенные импликанты: $\overline{x_2} \cdot \overline{x_3}$ и $x_3 \overline{x_4}$. С учетом этого поиск остальных импликант минимальной ДНФ можно упростить, исключив строки с существенными импликантами, а также перекрываемые ими столбцы. Это показано в табл. Б.12 (удаляемые строки и столбцы закрашены).

После вычеркивания существенных импликант $\overline{x_2} \cdot \overline{x_3}$ и $x_3 \overline{x_4}$, а также столбцов с минтермами, которые поглощаются этими импликантами, получим сокращенную матрицу (табл. Б.13).

Таблица Б.13. Сокращенная импликантная матрица

		5	7
0,2,8,10	$\overline{x_2} \cdot \overline{x_4}$		
1,5	$\overline{x_1} \cdot \overline{x_3} x_4$	√	
5,7	$\overline{x_1} x_2 x_4$	√	√
6,7	$\overline{x_1} x_2 x_3$		√

Первая строка не содержит меток избыточности, поэтому ее можно удалить (табл. Б.14).

Таблица Б.14. Сокращенная импликантная матрица после исключения пустых строк

		5	7
1,5	$\overline{x_1} \cdot \overline{x_3} x_4$	√	
5,7	$\overline{x_1} x_2 x_4$	√	√
6,7	$\overline{x_1} x_2 x_3$		√

4. **Выбор минимального покрытия**

В сокращенной импликантной матрице (табл. Б.14) нужно выбрать такую минимально возможную совокупность строк, которая включает метки во всех столбцах («покрывает» все оставшиеся в таблице минтермы). Дизъюнкция простых импликант, соответствующих строкам этой совокупности, а также ранее вычеркнутых существенных импликант, образует тупиковую ДНФ. В общем случае полных покрытий с одинаковым числом строк, а значит, и тупиковых ДНФ может быть несколько.

Из матрицы (табл. Б.14) видно, что минимальное покрытие не исключенных ранее минтермов обеспечивает простая импликанта $\overline{x_1} x_2 x_4$ либо пара импликант $\overline{x_1} \cdot \overline{x_3} x_4$ и $\overline{x_1} x_2 x_3$. С учетом ранее выявленных существенных импликант получаем две тупиковые ДНФ:

$$f = \overline{x_1} x_2 x_4 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3 \overline{x_4};$$

$$f = \overline{x_1} \cdot \overline{x_3} x_4 \vee \overline{x_1} x_2 x_3 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3 \overline{x_4}.$$

5. **Определение и запись минимальной нормальной формы**

В случае нескольких тупиковых форм предпочтение отдается той из них, которая имеет наименьший коэффициент сложности. Если получилась лишь одна тупиковая ДНФ, то она одновременно является и минимальной.

Коэффициент сложности первой из двух получившихся тупиковых форм равен 10, а второй — 14. По этой причине минимальной ДНФ следует признать первое выражение:

$$f = \overline{x_1} x_2 x_4 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3 \overline{x_4}.$$

Минимизация по методу Квайна–Мак-Класски

Метод Квайна–Мак-Класски отличается от метода Квайна только в той части, которая связана со способом нахождения простых импликант. Взамен громоздкой

процедуры склеивания всех возможных пар импликант Мак-Класски (Edward J. McCluskey) предложил следующую процедуру.

1. Все минтермы представляются соответствующими двоичными комбинациями — наборами.
2. Наборы разбиваются на группы. В i-ю группу объединяются наборы, содержащие i единиц.
3. Производятся все возможные операции склеивания, но только между наборами, входящими в соседние группы. В получаемых импликантах переменная, по которой происходило склеивание, заменяется каким-либо символом, например «–». Наборы, участвовавшие в склеивании, помечаются.
4. Процедуры, описанные в пунктах 2 и 3, повторяются применительно к импликантам, полученным на предыдущем этапе склеивания, до тех пор пока дальнейшее склеивание становится невозможным. Неотмеченные после склеивания наборы являются простыми импликантами, образующими сокращенную ДНФ.

Описанная модификация заменяет лишь первый шаг метода Квайна, при этом все последующие шаги производятся аналогично методу Квайна.

Проиллюстрируем описанную процедуру на примере ранее рассмотренной ФАЛ (см. табл. Б.7). Все наборы, на которых функция равна единице, распределим по группам (табл. Б.15). В i-ю группу включаются те наборы, которые содержат i единиц.

Таблица Б.15. Распределение минтермов по группам

Номер группы	Номер набора	Двоичный набор
0	0*	0000
1	1*	0001
	2*	0010
	8*	1000
2	5*	0101
	6*	0110
	9*	1001
	10*	1010
3	7*	0111
	14*	1110

Таблица Б.16. Первый этап склеивания членов групп

Группы	Склеиваемый наборы	Результат
0–1	0,1*	000–
	0,2*	00–0
	0,8*	–000
1–2	1,5	0–01
	1,9*	–001
	2,6*	0–10
	2,10*	–010
	8,9*	100–
	8,10*	10–0

Группы	Склеиваемый наборы	Результат
2–3	5,7 6,7 6,14* 10,14*	01–1 011– –110 1–10

Теперь проведем все возможные операции склеивания между парами наборов, входящими в соседние группы, при этом переменную, по которой производилось склеивание, заменим символом «–» (табл. Б.16).

Теперь аналогично произведем операцию над элементами новых соседних групп (табл. Б.17). В колонке «Группы» указаны номера новых групп, образовавшихся после первого этапа склеивания.

Таблица Б.17. Второй этап склеивания

Группы	Склеиваемые наборы	Результат
0–1	0,1,8,9 0,2,8,10 0,8,1,9 0,8,2,10	–00– –0–0 –00– –0–0
1–2	2,6,10,14 2,10,6,14	––10 ––10

Из таблицы видно, дальнейшее склеивание невозможно, вследствие чего первый этап минимизации завершается.

Исключим повторяющиеся наборы (это мы можем сделать на основании теоремы идемпотентности). Оставшиеся, а также непомеченные наборы соответствуют простым импликантам, которые можно представить в привычной записи, заменив единицы и нули обозначениями переменных соответственно без знака инверсии и со знаком инверсии и исключив переменные, обозначенные знаком «–» (табл. Б.18).

Таблица Б.18. Соответствие двоичных наборов простым импликантам

Двоичный набор	Простая импликанта
0–01	$\overline{x_1} \cdot \overline{x_3} x_4$
01–1	$\overline{x_1} x_2 x_4$
011–	$\overline{x_1} x_2 x_3$
–00–	$\overline{x_2} \cdot \overline{x_3}$
–0–0	$\overline{x_2} \cdot \overline{x_4}$
–10	$x_3 \overline{x_4}$

Полученные простые импликанты образуют сокращенную ДНФ:

$$f = \overline{x_1} \cdot \overline{x_3} x_4 \vee \overline{x_1} x_2 x_4 \vee \overline{x_1} x_2 x_3 \vee \overline{x_2} \cdot \overline{x_3} \vee \overline{x_2} \cdot \overline{x_4} \vee x_3 \overline{x_4}\,.$$

Нетрудно заметить, что получены те же простые импликанты, что и в методе Квайна. Дальнейшие действия совпадают с соответствующими этапами процедуры Квайна.

Минимизация по методу Петрика

Метод Петрика (Petrick) также имеет целью упрощение метода Квайна, но в части нахождения всех тупиковых форм по импликантной матрице.

Сначала стандартный вид импликантной матрицы заменяется конъюнктивным ее представлением, для чего все простые импликанты обозначаются прописными латинскими буквами, то есть теперь каждой строке импликантной матрицы соответствует какая-то буква. Для каждого i-го столбца импликантной матрицы записывается дизъюнкция, элементами которой служат буквы, обозначающие строки, где на пересечении с i-м столбцом стоит метка избыточности. Конъюнктивное представление импликантной матрицы — это конъюнкция вышеупомянутых дизъюнкций, составленных для всех столбцов матрицы. Далее выполняется упрощение конъюнктивного представления импликантной матрицы. Для этого могут применяться все известные правила булевой алгебры. После упрощения (раскрытия скобок, поглощений и т. п.) получается дизъюнкция конъюнкций, соответствующая тупиковой ДНФ.

В качестве иллюстрации метода рассмотрим импликантную матрицу, представленную в табл. Б.11.

Требуется найти все тупиковые ДНФ.

Сначала обозначим все имеющиеся простые импликанты прописными латинскими буквами:

$$\overline{x_2} \cdot \overline{x_3} = A; \quad \overline{x_2} \cdot \overline{x_4} = B; \quad x_3 \overline{x_4} = C;$$
$$\overline{x_1} \cdot \overline{x_3} x_4 = D; \; \overline{x_1} x_2 x_4 = E; \; \overline{x_1} x_2 x_3 = F.$$

		Простые импликанты	Минтермы									
			0	1	2	5	6	7	8	9	10	14
A	**0,1,8,9**	$\overline{x_2} \cdot \overline{x_3}$	√	√					√	√		
B	**0,2,8,10**	$\overline{x_2} \cdot \overline{x_4}$	√		√				√		√	
C	**2,6,10,14**	$x_3 \overline{x_4}$			√		√				√	√
D	**1,5**	$\overline{x_1} \cdot \overline{x_3} x_4$		√		√						
E	**5,7**	$\overline{x_1} x_2 x_4$				√		√				
F	**6,7**	$\overline{x_1} x_2 x_3$					√	√				

Теперь запишем конъюнктивное представление импликантной матрицы w:

$$w = (A \vee B)(A \vee D)(B \vee C)(D \vee E)(C \vee F)(E \vee F)(A \vee B)A(B \vee C)C.$$

После удаления одинаковых членов получаем:

$$w = (A \vee B)(A \vee D)(B \vee C)(D \vee E)(C \vee F)(E \vee F)AC.$$

Далее произведем возможные поглощения, в результате чего выражение принимает вид:

$$w = (D \vee E)(E \vee F)AC = (DE \vee EE \vee DF \vee EF)AC = (DE \vee E \vee DF \vee EF)AC =$$
$$= (E \vee DF)AC = ACE \vee ACDF.$$

Поскольку окончательное выражение содержит два слагаемых, имеют место две тупиковые ДНФ, представленные как *ACE* и *ACDF*. Первая содержит три простых импликанты $A = \overline{x_2} \cdot \overline{x_3}$, $C = x_3\overline{x_4}$ и $E = \overline{x_1}x_2x_4$. Таким образом, первая тупиковая ДНФ имеет вид $f = \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4} \vee \overline{x_1}x_2x_4$. Вторая тупиковая ДНФ состоит из 4 простых импликант — $A = \overline{x_2} \cdot \overline{x_3}$, $C = x_3\overline{x_4}$, $D = \overline{x_1} \cdot \overline{x_3}x_4$ и $F = \overline{x_1}x_2x_3$ — и может быть записана как $f = \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4} \vee \overline{x_1} \cdot \overline{x_3}x_4 \vee \overline{x_1}x_2x_3$. Как видно, коэффициент сложности первой тупиковой формы равен 10, а второй — 14, то есть минимальная ДНФ совпадает с первой тупиковой ДНФ.

Минимизация табличным методом

Метод, предложенный в 1952 году Е. Вейчем и усовершенствованный в 1953 году М. Карно, являет собой один из наглядных методов минимизации ФАЛ. В обоих случаях минимизация выполняется с помощью специальных карт, известных как диаграмма Вейча и карта Карно соответственно. Диаграмма Вейча отличается от карты Карно нумерацией строк и столбцов. Если в диаграмме Вейча они нумеруются в порядке возрастания двоичных чисел, например 00, 01, 10, 11, то в карте Карно используется код Грея, предполагающий циклический порядок следования номеров: 00, 01, 11, 10. В результате матрица Карно более удобна в обращении, что станет понятным из последующего изложения.

Карта Карно реализует *координатный способ* представления ФАЛ, при котором таблица истинности заменяется прямоугольной координатной картой состояний, содержащей 2^n клеток (по числу возможных наборов аргументов). Аргументы разбиваются на две группы так, что одна группа определяет координаты столбца карты, а другая — координаты строки. При таком способе каждая клетка соответствует определенному набору аргументов ФАЛ. Внутрь клетки ставится значение функции на данном наборе. Если предполагается получение минимальной ДНФ, то заполняются лишь те клетки, для которых значение функции равно 1. В случае минимальной КНФ — клетки, соответствующие наборам, где значение ФАЛ равно 0.

Двоичные числа, записываемые вокруг карты, характеризуют значения аргументов. В отличие от таблиц истинности, где значения аргументов обычно следуют в естественной последовательности (00, 01, 10, 11), нумерация ячеек в карте Карно выполняется согласно коду Грея, и это является определяющим моментом.

Особенность кода Грея в том, что двоичные коды номеров соседних столбцов и строк отличаются только в одном бите. Существуют карты Карно на 2, 3, 4, 5 и 6 переменных, причем последние стали использоваться достаточно недавно. На рис. Б.3 представлены карты Карно для 2, 3, 4, 5 и 6 аргументов.

Единица в ячейке карты Карно соответствует единичному значению функции в таблице истинности. Благодаря системе нумерации ячеек минтермы, отличающиеся только в одном бите (такие минтермы называются *смежными*, или соседними), располагаются в соседних ячейках по горизонтали или по вертикали. Отметим также, что код Грея является циклическим, то есть с точки зрения «соседства» карта Карно представляет собой пространственную фигуру. Так, карта для 4 аргументов — это тор. Это означает, что одинаково расположенные клетки в левой и правой крайних колонках карты, а также в крайних верхнем и нижнем рядах карты следует рассматривать как смежные. Так как смежные клетки соответствуют наборам, различающимся только в одном бите, к ним можно применить операцию склеивания, что на карте представляется в виде группировки соответствующих клеток.

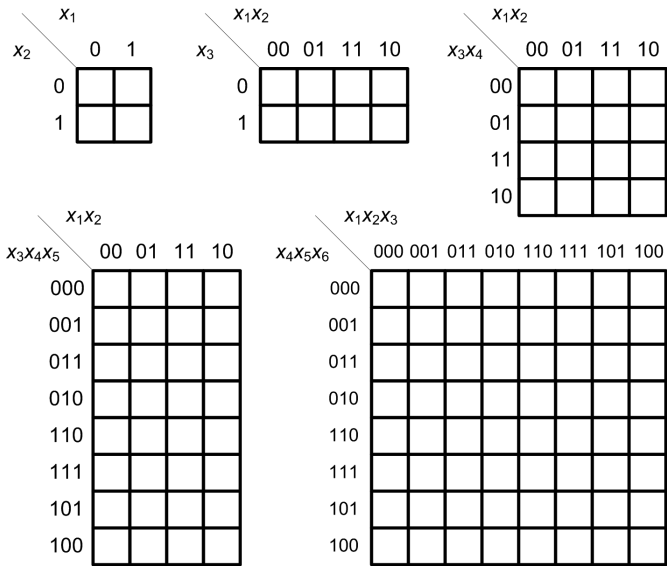


Рис. Б.3. Карты Карно для 2, 3, 4, 5 и 6 аргументов

Процедуру минимизации ФАЛ с помощью карт Карно можно описать следующим образом.

1. Заполнить единицами те клетки карты Карно, для которых значение функции равно 1 (для получения ДНФ), или же нулями те клетки, которые соответствуют нулевым значениям функции (для получения КНФ).
2. В карте Карно группы единиц (для получения ДНФ) или группы нулей (для получения КНФ) необходимо обвести четырехугольными контурами. Этот процесс соответствует операции склеивания или нахождения импликант данной функции.

3. Количество клеток внутри контура должно быть целой степенью двойки (1, 2, 4, 8, 16, ...).
4. При проведении контуров крайние строки карты (верхние и нижние, левые и правые), а также угловые клетки считаются соседними (для карт до 4-х переменных).
5. Каждый контур должен включать максимально возможное количество клеток, тогда он будет отвечать простой импликанте.
6. Все единицы (нули) в карте (даже одиночные) должны быть охвачены контурами. Любая единица (нуль) может входить в контуры произвольное количество раз.
7. Множество контуров, покрывающих все единицы (нули) в карте, образуют тупиковую ДНФ (КНФ). Целью минимизации является нахождение минимальной из множества тупиковых форм.
8. В элементарной конъюнкции (дизъюнкции), которая соответствует одному контуру, остаются только те переменные, значение которых не изменяется внутри обведенного контура. Переменные булевой функции входят в элементарную конъюнкцию (для значений функции 1) без инверсии, если их значение на соответствующих координатах равно 1, и с инверсией — если равно 0. Для значений булевой функции, равных 0, записываются элементарные дизъюнкции, куда переменные входят без инверсии, если их значение на соответствующих координатах равно 0, и с инверсией — если оно равно 1.

На рис. Б.4 приведен пример минимизации булевой функции трех аргументов с получением минимальной ДНФ. В примере все клетки, содержащие единицы, удалось охватить двумя контурами. В горизонтально ориентированном контуре значения переменных x_1 и x_3 совпадают, а значения x_2 различны. Таким образом, переменная x_2 является несущественной (фиктивной) и поэтому ее можно отбросить. Аналогично в вертикальном контуре можно избавиться от переменной x_3.

x_1	x_2	x_3	f	x_1	x_2	x_3	f
0	0	0	0	1	0	0	1
0	0	1	1	1	0	1	1
0	1	0	0	1	1	0	0
0	1	1	1	1	1	1	0

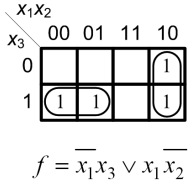


Рис. Б.4. Пример минимизации ФАЛ трех аргументов

Пример на рис. Б.5 иллюстрирует использование одной и той же клетки в нескольких контурах.

x_1	x_2	x_3	f	x_1	x_2	x_3	f
0	0	0	0	1	0	0	1
0	0	1	0	1	0	1	0
0	1	0	1	1	1	0	1
0	1	1	0	1	1	1	1

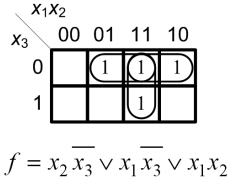


Рис. Б.5. Пример использования общей клетки в нескольких контурах

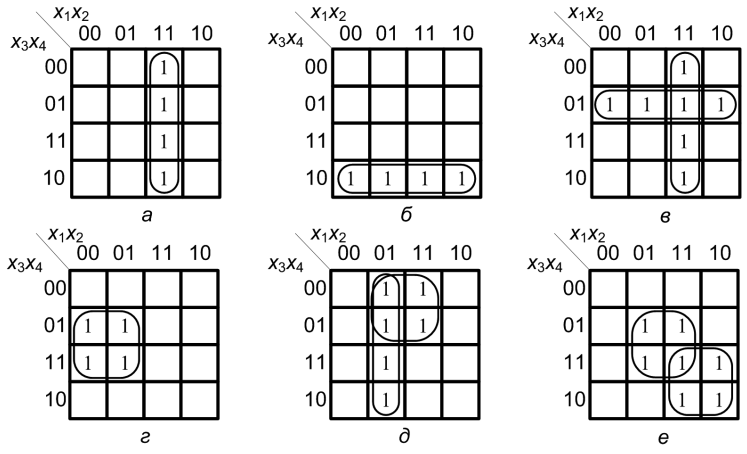


Рис. Б.6. Варианты группировки клеток для функции 4-х аргументов: *а* — $f = x_1x_2$; *б* — $f = x_3\overline{x_4}$; *в* — $f = x_1x_2 \vee \overline{x_3}x_4$; *г* — $f = \overline{x_1}x_4$; *д* — $f = x_2\overline{x_3} \vee \overline{x_1}x_2$; *е* — $f = x_2\overline{x_4} \vee x_1x_3$

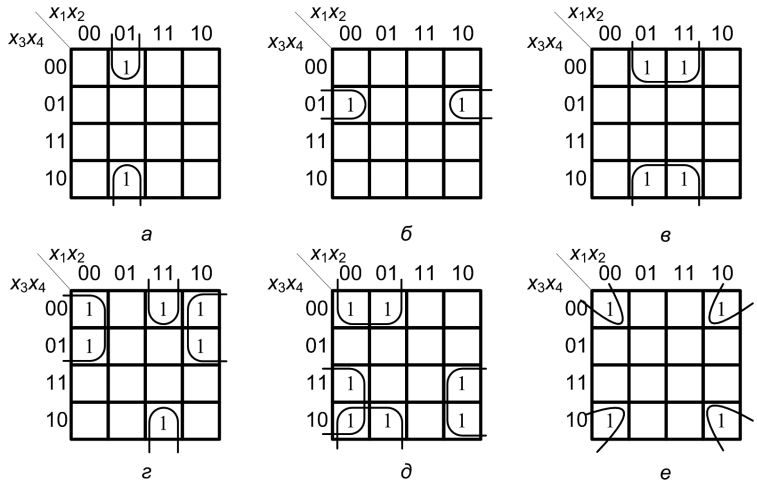
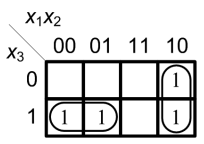


Рис. Б.7. Варианты группировки клеток, использующие циклический характер карт: *а* — $f = \overline{x_1}x_2\overline{x_4}$; *б* — $f = \overline{x_2} \cdot \overline{x_3}x_4$; *в* — $f = x_2\overline{x_4}$; *г* — $f = \overline{x_2} \cdot \overline{x_3} \vee x_1x_2\overline{x_4}$; *д* — $f = x_1 \cdot \overline{x_4} \vee \overline{x_2}x_3$; *е* — $f = \overline{x_2} \cdot \overline{x_4}$

Возможные варианты группировки клеток в картах Карно, в том числе и использующие циклический характер кода Грея, приведены на рис. Б.6 и Б.7.

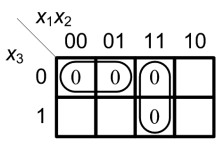
Когда карту Карно заполняют единицами из таблицы истинности, результат записывают в виде дизъюнктивной нормальной формы. В качестве альтернативы возможно заполнение нулями клеток, соответствующих нулевым значениям ФАЛ. В этом случае группируют нули, а результат записывается в виде конъюнктивной нормальной формы. На рис. Б.8 показаны варианты минимизации логической функции трех аргументов с получением минимальных ДНФ (рис. Б.8, *а*) и КНФ (рис. Б.8, *б*).

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



$f = \overline{x_1}x_3 \vee x_1\overline{x_2}$

а

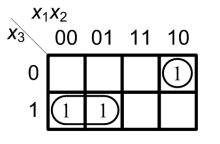


$f = (x_1 \vee x_3)(\overline{x_1} \vee \overline{x_2})$

б

Рис. Б.8. Примеры минимизации ФАЛ с получением минимальной: *а* — ДНФ; *б* — КНФ

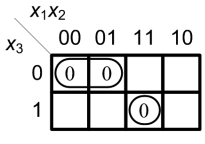
x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



$f = \overline{x_1}x_3 \vee x_1\overline{x_2} \cdot \overline{x_3}$

а

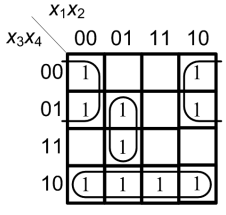
x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



$f = (x_1 \vee x_3)(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$

б

Рис. Б.9. Учет клеток, не имеющих в карте Карно соседних клеток: *а* — ДНФ; *б* — КНФ



$f = \overline{x_1}x_2x_4 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4}$

Рис. Б.10. Минимизация функции из табл. Б.7

Как в случае ДНФ, так и в случае КНФ элементы, не попавшие ни в одну из групп, необходимо добавить в результирующее выражение без каких-либо изменений (рис. Б.9).

В заключение в качестве примера рассмотрим минимизацию функции, которая приведена в табл. Б.7. Соответствующая карта Карно и результат минимизации показаны на рис. Б.10. Отметим, что метод карт Карно применим к минимизации булевых функций до 6-ти переменных: до 4-х переменных на плоскости и до 6-ти — в трехмерной интерпретации.

Минимизация частично определенных функций

Функция алгебры логики с n аргументами, значение которой определено на всех 2^n входных наборах, называется полностью определенной. Для реальных задач более характерен вариант, когда значения функции задаются только на части входных наборов. Логическая функция n аргументов, которая определена не на всех 2^n наборах, называется частично определенной. При задании частично определенной ФАЛ указываются номера лишь тех наборов, где функция равна нулю и единице. Остальные наборы считаются безразличными, или запрещенными. В таблице истинности вместо значения ФАЛ в строках с безразличными наборами записывается символ «*». Значение логической функции на безразличных наборах может доопределяться произвольно, причем независимо для каждого набора. Если число запрещенных комбинаций равно m, то путем доопределения можно получить 2^m различных функций. Естественно, что доопределение целесообразно производить таким образом, чтобы после минимизации ФАЛ имела наименьший возможный коэффициент сложности. Наиболее удобно это производить с помощью карт Карно.

Варианты доопределения частично определенных функций проиллюстрируем на примерах.

Пример 3. Функция трех аргументов задана таблицей истинности.

x_1	x_2	x_3	f	x_1	x_2	x_3	f
0	0	0	0	1	0	0	1
0	0	1	*	1	0	1	1
0	1	0	1	1	1	0	0
0	1	1	*	1	1	1	*

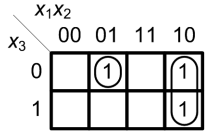
На рис. Б.11 показаны варианты доопределения данной функции.

Если значения функции на всех безразличных наборах принять равным 0, то получаем функцию, приведенную на рис. Б.11, *а*. МДНФ имеет вид $f = \overline{x_1}x_2\overline{x_3} \vee x_1\overline{x_2}$. МДНФ функции, в которой все *=1 (рис. Б.11, *б*) описывается выражением $f = x_3 \vee x_1x_2 \vee x_1\overline{x_2}$. Карта Карно позволяет найти вариант доопределения ФАЛ, при котором МДНФ будет иметь минимальный коэффициент сложности (рис. Б.11, *в*). Соответствующая этому варианту МДНФ имеет вид: $f = x_1\overline{x_2} \vee \overline{x_1}x_2$.

Пример 4. Найти минимальную форму для функции.

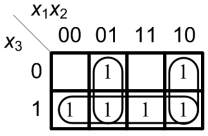
x_1	x_2	x_3	x_4	f	x_1	x_2	x_3	x_4	f
0	0	0	0	1	1	0	0	0	*
0	0	0	1	*	1	0	0	1	1
0	0	1	0	*	1	0	1	0	0
0	0	1	1	0	1	0	1	1	*
0	1	0	0	*	1	1	0	0	0
0	1	0	1	0	1	1	0	1	*
0	1	1	0	1	1	1	1	0	1
0	1	1	1	*	1	1	1	1	*

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



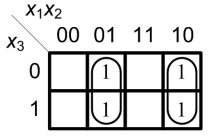
а

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1



б

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



в

Рис. Б.11. Варианты доопределения частично определенной функции: *а* — нулями; *б* — единицами; *в* — оптимальным образом

Оптимальное доопределение функций можно проводить с помощью метода Квайна. Сначала примем условие, при котором значение функции на всех неопределенных наборах равно 1. Проведя первые этапы процедуры минимизации Квайна, получим сокращенную ДНФ:

$$f = \overline{x_1} \cdot \overline{x_4} \vee \overline{x_2} \cdot \overline{x_3} \vee x_2 x_3 \vee x_1 x_4.$$

Составим импликантную матрицу.

	$\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} \cdot \overline{x_4}$	$\overline{x_1} x_2 x_3 \overline{x_4}$	$x_1 \overline{x_2} \cdot \overline{x_3} x_4$	$x_1 x_2 x_3 \overline{x_4}$
$\overline{x_1} \cdot \overline{x_4}$	√	√		
$\overline{x_2} \cdot \overline{x_3}$	√		√	
$x_2 x_3$		√		√
$x_1 x_4$			√	

Матрица позволяет получить минимальный вид результирующей функции $f = \overline{x_2} \cdot \overline{x_3} \vee x_2 x_3$. Этот результат получается при следующем доопределении исходной функции.

x_1	x_2	x_3	x_4	f	x_1	x_2	x_3	x_4	f
0	0	0	0	1	1	0	0	0	1
0	0	0	1	1	1	0	0	1	1
0	0	1	0	0	1	0	1	0	0
0	0	1	1	0	1	0	1	1	0
0	1	0	0	0	1	1	0	0	0
0	1	0	1	0	1	1	0	1	0
0	1	1	0	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1

Тот же результат, как уже отмечалось, может быть достигнут с помощью карты Карно (рис. Б.12).

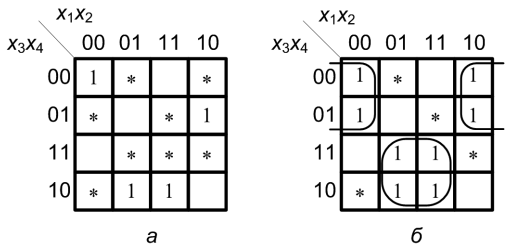


Рис. Б.12. Частично определенная функция: *а* — перед доопределением; *б* — после доопределения $f = \overline{x_2} \cdot \overline{x_3} \vee x_2 x_3$

Минимизация совокупности логических функций

В практике проектирования логических схем очень частым является случай, когда устройство имеет несколько выходов. Состояние каждого выхода в этом случае описывается с помощью отдельной ФАЛ, а общее описание устройства представляется совокупностью (системой) булевых функций. Если каждую ФАЛ этой совокупности минимизировать по отдельности, то общая схема устройства будет состоять из изолированных подсхем, в которых могут присутствовать одинаковые участки. Объединение подобных участков ведет к упрощению схемы в целом, но это возможно, только если уравнения совокупности ФАЛ будут минимизироваться не по отдельности, а совместно. Общая идея минимизации совокупности функций алгебры логики сводится к получению таких выражений, в которых оптимально используются члены, общие для нескольких функций.

Минимизация систем ФАЛ может производиться по алгоритму, похожему на метод Квайна, но с небольшими отличиями. Рассмотрим этот алгоритм на примере совокупности двух ФАЛ:

$$f_1 = x_1 \overline{x_2} x_3 \vee x_1 x_2 \overline{x_3} \vee x_1 \overline{x_2} \cdot \overline{x_3}$$
$$f_2 = x_1 x_2 x_3 \vee x_1 x_2 \overline{x_3} \vee x_1 \overline{x_2} x_3.$$

Сначала сформируем множество минтермов всех уравнений системы. Каждый минтерм снабдим признаком, приписав его в виде индекса. В качестве признака используем название функции, в которую входит минтерм: $\{x_1\overline{x_2}x_{3f}, x_1x_2\overline{x_3}_f, x_1\overline{x_2}\cdot\overline{x_3}_f, x_1x_2x_{3f}, x_1x_2\overline{x_3}_f, x_1\overline{x_2}x_{3f}\}$.

Из полученного множества выделим подмножество неодинаковых элементов. Из одинаковых элементов оставим по одному, изменив в них индекс так, чтобы он указывал на все логические выражения, в которые входит данный элемент (такое подмножество называется полным подмножеством членов совокупности ФАЛ). Проделав описанную процедуру, получим: $\{x_1\overline{x_2}x_{3f_1f_2}, x_1x_2\overline{x_3}_{f_1f_2}, x_1\overline{x_2}\cdot\overline{x_3}_{f_1}, x_1x_2x_{3f_2}\}$. Именно это подмножество используется при минимизации совокупности ФАЛ.

Итак, задача заключается в нахождении *минимальной совокупности дизъюнктивных нормальных форм* ФАЛ, под которой понимается та, где полное подмножество членов совокупности содержит минимальное количество букв, а каждая ФАЛ включает минимальное число дизъюнктивных членов. Иными словами, минимальная совокупность должна иметь наименьший возможный суммарный ранг. Следует отметить, что форма представления каждой отдельной ФАЛ, входящей в минимальную совокупность, может быть отлична от минимальной для данной отдельной ФАЛ.

Выпишем и пронумеруем все минтермы с их признаками:

$$1-\ x_1\overline{x_2}x_{3f_1f_2};\ 2-\ x_1x_2\overline{x_3}_{f_1f_2};\ 3-\ x_1\overline{x_2}\cdot\overline{x_3}_{f_1};\ 4-\ x_1x_2x_{3f_2}.$$

Далее, как и в методе Квайна, начинается процедура получения простых импликант. Для получения простых импликант проводятся все возможные склеивания членов подмножества. Полученным после склеивания произведениям приписывается признак, состоящий из общих букв, содержащихся в признаках обоих склеивающихся членов. Если ни одна буква в признаке склеиваемых членов не совпадает, то произведению признак не присваивается:

$$1,3-\ x_1\overline{x_2}_{f_1};\ 1,4-\ x_1x_{3f_2};\ 2,3-\ x_1\overline{x_3}_{f_1};\ 2,4-\ x_1x_{2f_2}.$$

После проведения склеиваний выполняются поглощения, причем операции поглощения можно проводить только между членами с одинаковыми признаками. Минтермы, которые не поглощаются ни одним произведением, являются *простыми импликантами совокупности функций*.

Поглощенными оказываются $x_1\overline{x_2}\cdot\overline{x_3}_{f_1}$ и $x_1x_2x_{3f_2}$. Минтермы $x_1\overline{x_2}x_{3f_1f_2}$ и $x_1x_2\overline{x_3}_{f_1f_2}$ являются простыми импликантами.

Для получения всех простых импликант заданной совокупности функций операции склеивания выполняются над произведениями, полученными в результате склеивания минтермов. При этом произведения, не содержащие признака, в склеивании не участвуют.

Дальнейшее склеивание дает $1,3,2,4-x_1$; $1,4,2,3-x_1$.

Поскольку последние импликанты не имеют признака, то они исключаются.

В нашем примере первый этап на этом завершается, но в общем случае склеивания и поглощения повторяются, пока очередной цикл склеивания станет невозможным.

Таким образом, простыми импликантами совокупности будут: $x_1\overline{x_2}x_{3f_1f_2}$, $x_1x_2\overline{x_3}_{f_1f_2}$, $x_1\overline{x_2}_{f_1}$, $x_1x_{3f_2}$, $x_1\overline{x_3}_{f_1}$, $x_1x_{2f_2}$.

Для нахождения минимальной совокупности, как и в методе Квайна, используется импликантная матрица. В заголовки столбцов матрицы записываются все минтермы, а в горизонтальные входы — все простые импликанты совокупности функций. Каждому минтерму приписывается признак, указывающий, в какие функции этот минтерм входит.

	$x_1\overline{x_2}x_3$		$x_1x_2\overline{x_3}$		$x_1\overline{x_2}\cdot\overline{x_3}$	$x_1x_2x_3$
	f_1	f_2	f_1	f_2	f_1	f_2
$x_1\overline{x_2}x_{3f_1f_2}$	√	√				
$x_1x_2\overline{x_3}_{f_1f_2}$			√	√		
$x_1\overline{x_2}_{f_1}$	√				√	
$x_1x_{3f_2}$		√				√
$x_1\overline{x_3}_{f_1}$			√		√	
$x_1x_{2f_2}$				√		√

Заключительный этап минимизации совокупности ФАЛ состоит в выборе подмножества импликант с минимальным числом букв, покрывающих все столбцы импликантной матрицы. Для рассматриваемого примера — это простые импликанты $x_1\overline{x_2}x_{3f_1f_2}$, $x_1\overline{x_3}_{f_1}$, $x_1x_{2f_2}$. Выделив для функции f_i импликанты с признаком f_i, получим искомую минимальную совокупность ФАЛ:

$$f_1 = x_1\overline{x_2}x_3 \vee x_1\overline{x_3};$$
$$f_2 = x_1\overline{x_2}x_3 \vee x_1x_2.$$

Учитывая, что первый член входит в оба выражения, при вычислении коэффициента сложности совокупности этот элемент нужно учитывать лишь однократно. Поэтому суммарный ранг совокупности равен 7, а коэффициент сложности — 10.

Если бы мы минимизировали каждую из ФАЛ совокупности отдельно, то получили бы систему:

$$f_1 = x_1\overline{x_2} \vee x_1\overline{x_3};$$
$$f_2 = x_1x_3 \vee x_1x_2,$$

в которой ранги первых членов выражений меньше, однако, суммарный ранг системы равен 8, а коэффициент сложности — 12, то есть больше, чем при минимизации совокупности ФАЛ.

Следует заметить, что если упростить хотя бы одну ФАЛ, входящую в минимальную совокупность, то количество букв в полном подмножестве увеличится.

Приложение В

Схемотехнические основы вычислительных машин

Сигналы в цифровой схемотехнике

Цифровой сигнал — это сигнал, который может принимать два значения, рассматриваемые как логическая «1» и логический «0». Устройства, работающие только с цифровыми сигналами, называются *цифровыми устройствами*. При описании цифровых сигналов используются определенные термины (рис. В.1):

- активный уровень сигнала — уровень, порождающий выполнение устройством соответствующей функции;
- пассивный уровень сигнала — уровень, при котором устройство не выполняет никакой функции;
- положительный сигнал (сигнал положительной полярности) — сигнал, активный уровень которого — логическая «1» («0» соответствует отсутствию сигнала);
- отрицательный сигнал (сигнал отрицательной полярности) — сигнал, активный уровень которого — логический «0» («1» соответствует отсутствию сигнала);

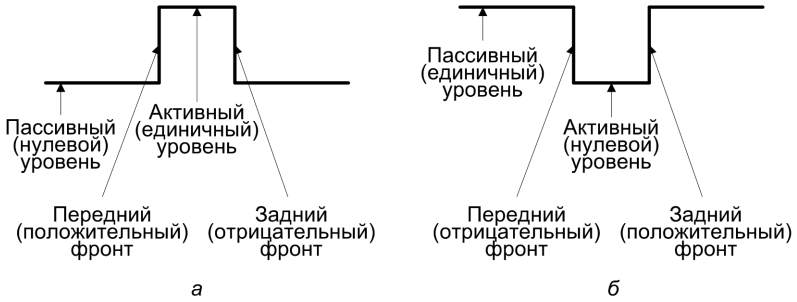


Рис. В.1. Основные параметры цифровых сигналов: *а* — в положительной логике; *б* — в отрицательной логике

- передний фронт сигнала — переход сигнала из пассивного уровня в активный;
- задний фронт сигнала — переход сигнала из активного уровня в пассивный;
- положительный фронт сигнала — переход сигнала из «0» в «1»;
- отрицательный фронт сигнала — переход сигнала из «1» в «0»;
- тактовый сигнал (строб) — управляющий сигнал, определяющий момент выполнения элементом или узлом его функции.

Логические элементы

Между аналитической формой представления булевых функций и их схемной реализацией существует взаимно однозначное соответствие: каждой элементарной ФАЛ соответствует схемный аналог. Электронные схемы, выполняющие простейшие логические операции, называются *логическими элементами*. Для реализации разнообразных логических функций достаточно иметь логические элементы, обеспечивающие минимальный логический базис.

Основные обозначения на схемах

В настоящее время для обозначения логических элементов используются несколько стандартов. Наиболее распространенными являются международный (IEC), российский (ГОСТ), американский (ANSI) и европейский (DIN). ГОСТ на уровне логических элементов практически идентичен международному стандарту. Здесь логические элементы изображаются в виде прямоугольников с соответствующими надписями внутри. В двух последних стандартах каждый вид логического элемента представляется особым символом. Отметим, что стандарт DIN фактически является стандартом, принятым в Германии, однако иногда он используется и в других европейских странах, хотя широкого распространения пока не получил.

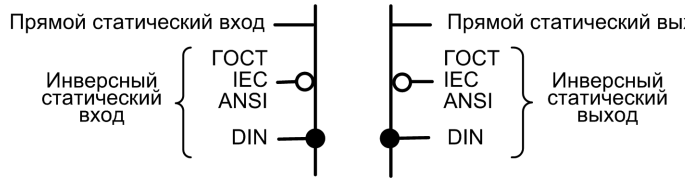


Рис. В.2. Типовое изображение входов и выходов на условном обозначении логических элементов

Возможное обозначение входов и выходов логических элементов показано на рис. В.2.

Логический элемент «НЕ»

Логический элемент «НЕ» (инвертор) имеет всего один вход и один выход. Выходной сигнал инвертора принимает всегда противоположное значение по отношению к значениям входного сигнала. Схема «НЕ» реализует операцию $F = \overline{x}$ (читается как «не *x*») и на электрических схемах изображается в одном из вариантов, приведенных на рис. В.3.

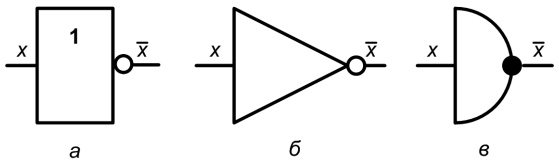


Рис. В.3. Условные обозначения логического элемента «НЕ» в стандартах:
а — ГОСТ и IEC; *б* — ANSI; *в* — DIN

Кружок служит указателем инверсии[1].

Логический элемент «И»

Логические элементы «И» реализуют функцию конъюнкции (логического умножения). Минимальное число входов равно двум. В общем случае такие элементы описываются логическим выражением вида $f = x_1 \wedge x_2 \wedge ... \wedge x_n$ и изображаются, как показано на рис. В.4.

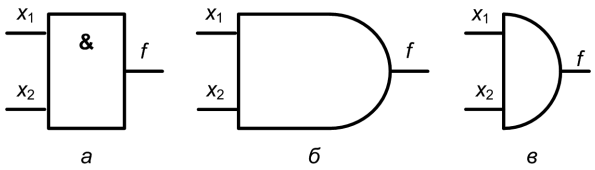


Рис. В.4. Условные обозначения логического элемента «И» в стандартах:
а — ГОСТ и IEC; *б* — ANSI; *в* — DIN

Сигнал на выходе логического элемента «И» соответствует логической единице, когда на все n входов ($n \geq 2$) поданы сигналы логической единицы. По этой причине такие элементы называют схемами совпадения. Иногда используется еще одно название — «конъюнкторы».

Переместительный и сочетательный законы булевой алгебры предопределяют возможность построения схемы «И» на большое число входов с использованием конъюнкторов, имеющих меньшее количество входов. Так, 6-входовую схему «И» можно описать выражением $f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_6$ и реализовать на базе двухвходовых схем «И», если согласно упомянутым законам представить данное выражение в виде $f = ((((x_1 \wedge x_2) \wedge x_3) \wedge x_4) \wedge x_5) \wedge x_6$ либо $f = ((x_1 \wedge x_2) \wedge (x_3 \wedge x_4)) \wedge (x_5 \wedge x_6)$. Соответственно получаем два варианта схемы (рис. В.5).

Логически оба варианта эквивалентны. Следует, однако, учитывать, что сигнал на выходе логической схемы появляется с некоторой задержкой по отношению к моменту подачи входных сигналов. По этой причине второй вариант (см. рис. В.5, *б*) предпочтителен в плане быстродействия, так как в нем входные сигналы проходят через меньшее число элементов.

---

[1] ГОСТ разрешает и иные варианты размещения указателя инверсии. Здесь же будем придерживаться варианта, совпадающего с международным стандартом IEC.

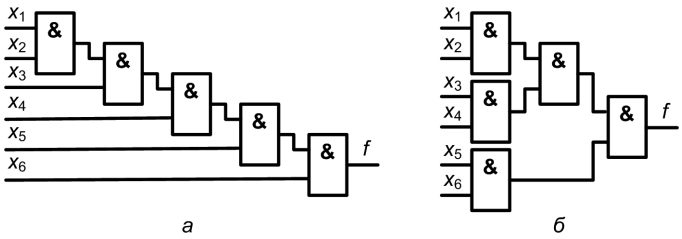


Рис. В.5. Варианты построения 6-входовой схемы «И» на базе двухвходовых схем: *а* — каскадный; *б* — параллельный

Логический элемент «ИЛИ»

Логический элемент «ИЛИ» — это электронная логическая схема, выходной сигнал которой соответствует логическому нулю, когда логическому нулю равны сигналы на всех входах схемы. Схема обеспечивает логическое сложение входных сигналов $(f = x_1 \vee x_2 \vee ... \vee x_n)$.

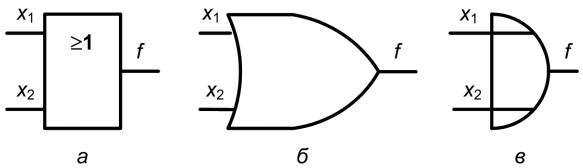


Рис. В.6. Условные обозначения логического элемента «ИЛИ» в стандартах: *а* — ГОСТ и IEC; *б* — ANSI; *в* — DIN

Возможные графические обозначения показаны на рис. В.6.

Логический элемент «И-НЕ»

Электронная логическая схема, в которой выходной сигнал соответствует логическому «0», когда сигналы на всех входах равны логической «1». Схема реализует инверсию логического произведения всех входных сигналов, то есть логическую операцию $f = \overline{x_1 \wedge x_2 \wedge ... \wedge x_n}$. Условные графические обозначения логического элемента «И-НЕ» показаны на рис. В.7.

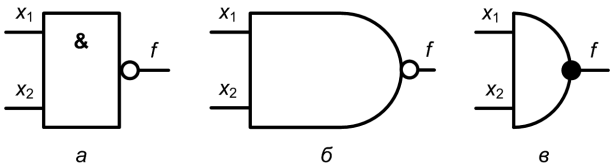


Рис. В.7. Условные обозначения логического элемента «И-НЕ»: *а* — ГОСТ и IEC; *б* — ANSI; *в* — DIN

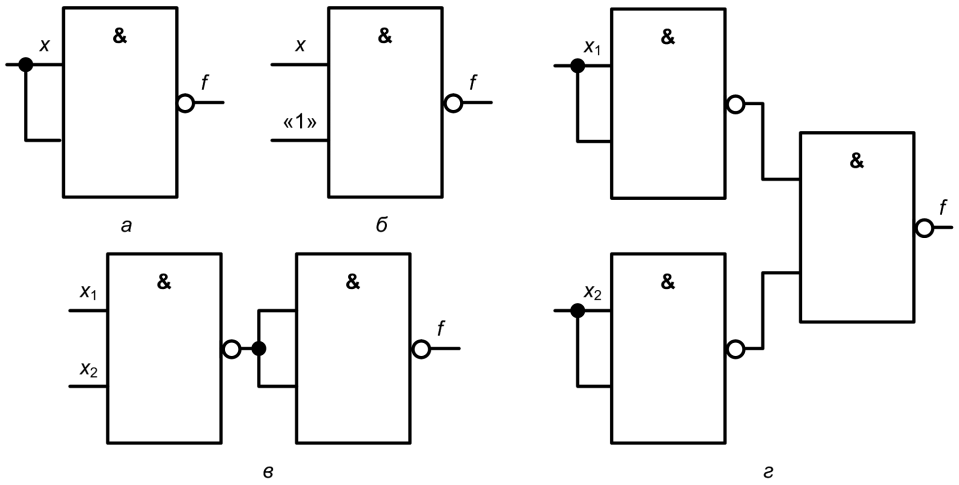


Рис. В.8. Реализация на базе логического элемента «И-НЕ» функций:
а, б — «НЕ»; *в* — «И»; *г* — «ИЛИ»

Имея элемент «И-НЕ», можно реализовать элементы «НЕ», «И», «ИЛИ», как это показано на рис. В.8.

Элемент «ИЛИ-НЕ»

Электронная логическая схема, в которой выходной сигнал соответствует логической «1», когда сигналы на всех входах равны логическому «0». Схема реализует инверсию логической суммы, то есть логическую операцию «ИЛИ-НЕ» ($f = \overline{x_1 \vee x_2 \vee ... \vee x_n}$), и обозначается, как показано на рис. В.9.

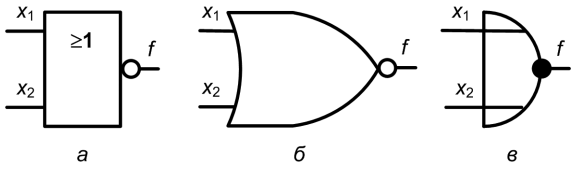


Рис. В.9. Условные обозначения логического элемента «ИЛИ-НЕ»: *а* — ГОСТ и IEC; *б* — ANSI; *в* — DIN

Элемент «ИЛИ-НЕ» можно трансформировать в элементы «НЕ», «И», «ИЛИ», как это показано на рис. В.10.

Набор логических элементов, реализующий операции того или иного минимального базиса, называется *минимальным элементным базисом*. В современной микроэлектронике таким базисом служат элементы «И-НЕ» либо «ИЛИ-НЕ».

Использование только элементов минимального базиса часто приводит к увеличению сложности устройств и ухудшает их основные эксплуатационные параметры. Поэтому во многих случаях используются расширенные (избыточные) элементы

базиса, в которых кроме элементов «И-НЕ», «ИЛИ-НЕ» используются схемы, выполняющие функции «И-ИЛИ-НЕ», «И», «ИЛИ», «Исключающее ИЛИ».

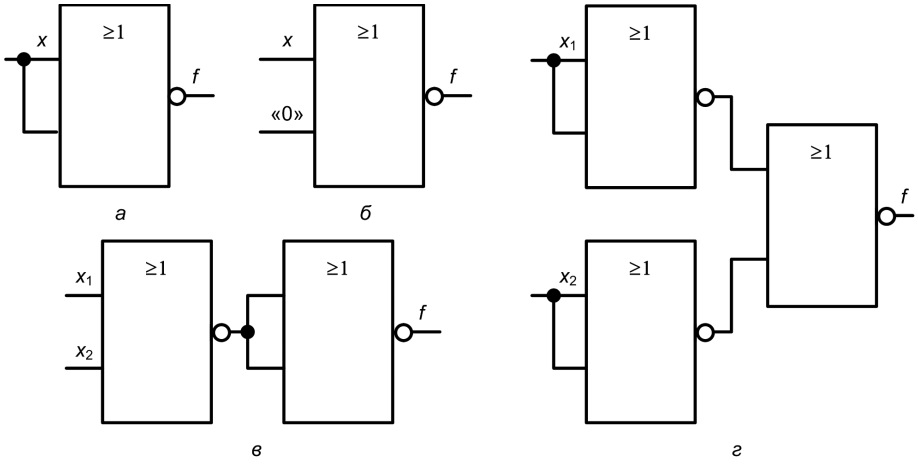


Рис. В.10. Реализация на базе логического элемента «ИЛИ-НЕ» функций: *а, б* — «НЕ»; *в* — «ИЛИ»; *г* — «И»

Логический элемент «Исключающее ИЛИ»

В общем случае речь идет о схеме, на выходе которой сигнал соответствует логической единице, когда значение логической единицы имеет нечетное число аргументов. Под логическим элементом «Исключающее ИЛИ» понимают схему с двумя входами. Схемы с большим числом входов обычно рассматриваются как самостоятельные логические устройства, известные под названием «схем контроля четности». Приведенному описанию соответствует логическая функция «неравнозначность». Функция равносильна операции сложения по модулю 2. Для двух аргументов она описывается выражением $f = x_1 \oplus x_2 = x_1 \wedge \overline{x_2} \vee \overline{x_1} \wedge x_2$. Графические изображения логического элемента «Исключающее ИЛИ» показаны на рис. В.11.

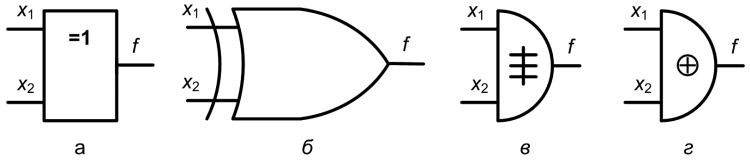


Рис. В.11. Условные обозначения логического элемента «Исключающее ИЛИ» в стандартах: *а* — ГОСТ и IEC; *б* — ANSI; *в*, *г* — DIN

Логический элемент «Эквивалентность»

В некоторых микросхемах встречается логический элемент, реализующий функцию эквивалентности или логической равнозначности ($f = \overline{x_1 \oplus x_2} = x_1 \wedge x_2 \vee \overline{x_1} \wedge \overline{x_2}$).

Условные графические обозначения подобных логических элементов показаны на рис. В.12.

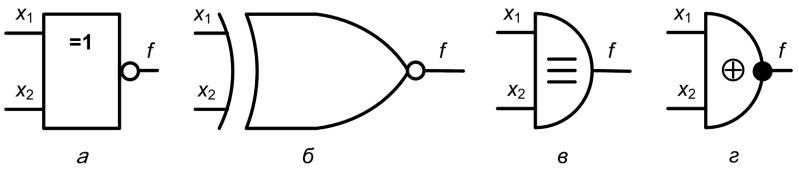


Рис. В.12. Условные обозначения логического элемента «Исключающее ИЛИ» с инверсным выходом в стандартах: *а* — ГОСТ и IEC; *б* — ANSI; *в*, *г* — DIN

Смысловое значение этой функции в том, что она принимает значение логической «1» при четном и значение логического «0» при нечетном числе единичных значений ее аргументов. В многовходовом варианте ($n > 2$) реализующие ее схемы получили название «схем контроля нечетности».

Положительная и отрицательная логика

Напряжения на входах и выходах логических элементов могут принимать два уровня: высокий (H — high) и низкий (L — low). Если высокий уровень соответствует логической «1», а низкий уровень — логическому «0», то принято считать, что логический элемент работает с *положительной логикой*. Если высокий уровень соответствует логическому «0», а низкий уровень — логической «1», то элемент работает с *отрицательной логикой*.

Пусть имеется таблица истинности логического элемента «ИЛИ».

x_1	x_2	f
0	0	0
0	1	1
1	0	1
1	1	1

Для элемента с положительной логикой эту таблицу можно переписать в следующем виде:

x_1	x_2	f
L	L	L
L	H	H
H	L	H
H	H	H

Однако этот же элемент реализует также и определенную логическую функцию для отрицательной логики. Для отрицательной логики таблицу истинности логического элемента можно переписать в виде:

x_1	x_2	f_1
1	1	1
1	0	0
0	1	0
0	0	0

Эта таблица истинности соответствует логическому элементу «И». Докажем это алгебраически: $f = x_1 \vee x_2$. Перейдем от положительной логики к отрицательной: $f_1 = \overline{f} = \overline{x_1} \vee \overline{x_2} = \overline{x_1 \wedge x_2} = x_1 \wedge x_2$. Таким образом, один и тот же элемент для положительной логики является элементом «ИЛИ», а для отрицательной логики — элементом «И». Очевидно, что элемент, являющийся элементом «И» для положительной логики, будет являться элементом «ИЛИ» — для отрицательной логики. Аналогично элемент «И-НЕ» трансформируется в «ИЛИ-НЕ», а «ИЛИ-НЕ» — в «И-НЕ». Таким образом, логическую функцию элемента можно изменить, не затрагивая его структуры, а лишь поменяв логику сигналов на входах и выходах.

Чтобы явно указать использование отрицательной логики, несколько видоизменяют условные графические изображения логических элементов (рис. В.13).

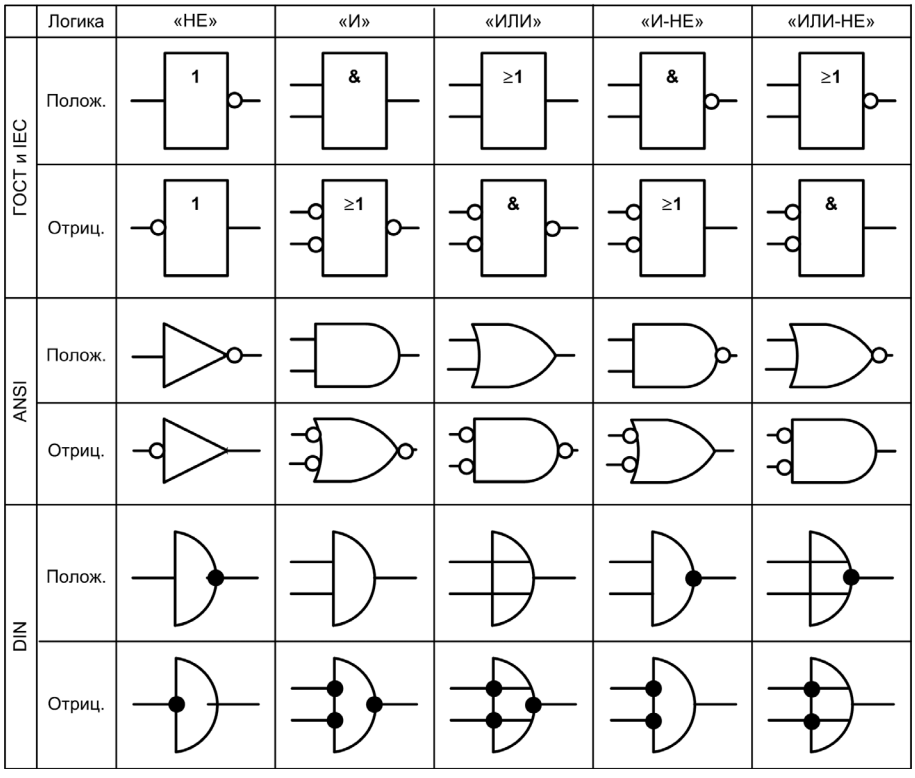


Рис. В.13. Условные графические изображения логических элементов в положительной и отрицательной логике

Элементы памяти

Состояние выходных сигналов рассмотренных логических элементов определяется лишь текущим состоянием входов и не зависит от предыдущего состояния схемы. Иными словами, логические элементы не обладают свойством памяти и не могут быть использованы для запоминания информации. Функцию элемента памяти в вычислительной техники выполняют *последовательностные логические устройства*, в которых выходной сигнал зависит не только от текущих входных логических значений, но и от тех, которые действовали на входе в предыдущие моменты времени.

Простейшая последовательностная схема состоит из пары схем «НЕ», охваченных петлей обратной связи (рис. В.14).

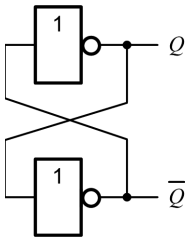


Рис. В.14. Простейший бистабильный элемент

Схему часто называют *бистабильной*, поскольку у нее есть два устойчивых состояния. Так как элемент не имеет свободных входов, им невозможно управлять и изменять его состояние. В реальных последовательных устройствах бистабильный элемент строится не на схемах «НЕ», а на схемах «И-НЕ» либо «ИЛИ-НЕ». Появляющиеся при этом дополнительные входы позволяют реализовать логику изменения состояния элемента. Простейшим последовательностным элементом является триггер.

Триггеры

Триггер представляет собой устройство, которое находится в одном из двух устойчивых состояний и скачкообразно переходит из одного состояния в другое под воздействием внешнего управляющего сигнала. Триггер может служить элементом памяти, способным хранить 1 бит информации. В основе любого триггера лежит схема из двух логических элементов, которые охвачены положительными обратными связями (см. рис. В.14). Триггеры являются основными составными элементами в большинстве последовательностных устройств.

Выходы триггеров

Основной выход триггера, определяющий его состояние, обозначают Q. Обычно у триггеров есть и инверсный выход — $\overline{Q}$. В том случае, когда в триггере хранится

логическая 1 ($Q = 1$), говорят, что триггер установлен. Если триггер хранит логический 0 ($Q = 0$), считается, что триггер сброшен.

Для описания работы триггера аналогично комбинационным схемам могут быть использованы таблицы истинности или логические выражения. Особенностью такого описания является использование в качестве дополнительной входной переменной предыдущего значения выходного сигнала триггера.

Входы триггеров

Состояние триггера может меняться только при воздействии внешних сигналов. Для обеспечения возможности изменения состояния триггера организуют цепи записи информации. В настоящее время разработано большое количество типов триггеров, которые различаются способами записи информации. Способ записи можно определить по обозначению информационных входов:

- *R* (Reset — сброс) — вход установки триггера в нулевое состояние ($Q = 0$);
- *S* (Set — установка) — вход установки триггера в единичное состояние ($Q = 1$);
- *K* (Kill — аннулирование) — вход сброса универсального триггера ($Q = 0$);
- *J* (Jerk — толчок) — вход установки универсального триггера ($Q = 1$);
- *T* (Toggle — переключатель) — вход триггера, по которому он меняет свое текущее состояние на противоположное;
- *D* (Delay — задержка) — информационный вход переключения триггера в состояние, соответствующее логическому уровню на этом входе;
- *C* (Clock — такт) — управляющий или синхронизирующий вход.

Кроме этих основных входов некоторые триггеры могут снабжаться входом *V*. Вход *V* блокирует работу триггера, при этом триггер продолжает сохранять ранее записанную в него информацию.

Входы триггеров могут быть прямыми (статическими или динамическими) и инверсными (статическими или динамическими). *Статические входы* реагируют на уровень входного сигнала, а *динамические* — на его перепад. Типовое изображение статических входов на условном обозначении логических элементов приводилось на рис. В.2. Изображение динамических входов в разных стандартах практически совпадает и показано на рис. В.15.

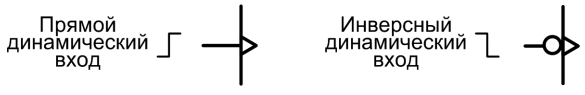


Рис. В.15. Типовое изображение динамических входов на условном обозначении триггеров

Классификация триггеров

Триггеры классифицируют по логическому функционированию и способу записи информации.

По логическому функционированию различают *RS*-, *JK*-, *D*-, *DV*-, *T*- и *TV*-триггеры. Название типа триггера отражает вид используемых входов. Кроме того, используются комбинированные триггеры, где совмещается одновременно несколько типов, и триггеры со сложной логикой, в которых группы входов связаны между собой логическими зависимостями.

При классификации *по способу записи информации* выделяют несколько признаков.

Первый признак — момент реакции на входной сигнал. Здесь различают *асинхронные* (нетактируемые) и *синхронные* (тактируемые) триггеры. Асинхронный триггер изменяет свое состояние непосредственно в момент изменения сигнала на его информационных входах, то есть его реакция на изменение входного сигнала подобна реакции комбинационного элемента. Синхронный триггер изменяет свое состояние лишь в строго определенные моменты времени, соответствующие действию на его синхронизирующем входе *C* активного сигнала. При пассивном значении сигнала *C* триггер на изменение информационных сигналов не реагирует.

Вторым признаком служит способ восприятия тактовых сигналов. По этому признаку различают триггеры, *управляемые уровнем* (со статическим управлением, или стробируемые) и *управляемые фронтом* (с динамическим управлением, или тактируемые). Управление уровнем означает, что при одном уровне тактового сигнала триггер «прозрачен» — воспринимает входные сигналы и реагирует на них, а при другом уровне — остается в неизменном состоянии, не реагируя на входные сигналы. При управлении фронтом разрешение на переключение триггера дается только в момент перепада тактового сигнала (на фронте или спаде). В остальное время триггер не воспринимает входные сигналы и остается в неизменном состоянии.

Наконец, третий признак — характер процесса переключения триггера. Здесь различают одноступенчатые и двухступенчатые триггеры. В одноступенчатом триггере переключение в новое состояние происходит сразу, а в двухступенчатом — по этапам. Двухступенчатые триггеры содержат два триггера — входной и выходной. На первом этапе информация заносится во входной триггер, а на втором этапе — переносится в выходной. Подробнее двухступенчатые триггеры рассматриваются ниже.

RS-триггеры

RS-триггер — это элемент с двумя входами (*S* и *R*) и двумя выходами — прямым (*Q*) и инверсным ($\overline{Q}$). Если на вход *S* такого триггера подать логический сигнал «1» (на входе $R = 0$), то выходной сигнал *Q* примет значение «1». Если подать «1» на вход *R* (на входе $S = 0$), выходной сигнал *Q* примет значение «0». При $Q = 1$ и $\overline{Q} = 0$ считается, что триггер находится в единичном состоянии, а при $Q = 0$ и $\overline{Q} = 1$ — в нулевом. Одновременная подача сигналов $R = 1$ и $S = 1$ запрещена, поскольку в этом случае устройство утрачивает свойства триггера (на выходах *Q* и $\overline{Q}$ будут одинаковые значения, что невозможно по определению).

Асинхронный RS-триггер снабжен только двумя информационными входами: входом сброса *R* и входом установки *S*. По сути это простейший элемент памяти,

который может быть реализован на элементах «ИЛИ-НЕ» либо «И-НЕ». Условное обозначение триггера и временная диаграмма приведены на рис. В.16.

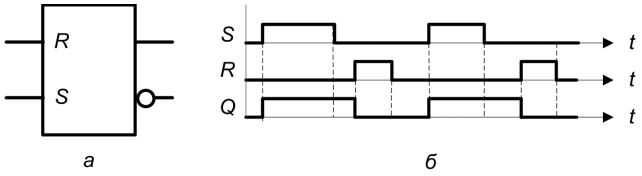


Рис. В.16. Асинхронный *RS*-триггер: *а* — условное графическое обозначение; *б* — временная диаграмма работы

Основной недостаток асинхронных триггеров — незащищенность перед опасными состязаниями сигналов, когда сигналы, поступающие на разные информационные входы триггера, проходят по разным цепям через различное число элементов. Из-за этого между сигналами возможны временные сдвиги, что может привести к ложным срабатываниям триггеров.

Синхронный RS-триггер срабатывает лишь при наличии дополнительного сигнала синхронизации на входе С. Срабатывание триггера может происходить по уровню сигнала (статическое управление) либо по фронту или срезу сигнала (динамическое управление). Условное графическое обозначение и временная диаграмма для синхронного RS-триггера приведены на рис. В.17.

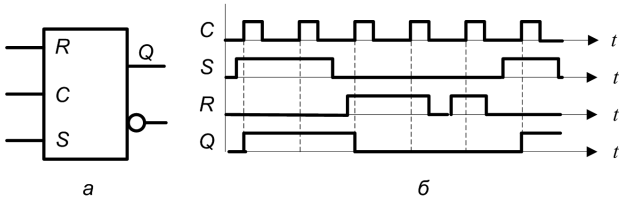


Рис. В.17. Синхронный *RS*-триггер: *а* — условное графическое обозначение; *б* — временная диаграмма работы

Синхронный триггер является более помехозащищенным, чем асинхронный, однако полностью исключить возможность возникновения на входах недопустимой кодовой комбинации данный триггер не позволяет.

JK-триггеры

JK-триггер по своей структуре сложнее *RS*-триггера. Прежде всего, это всегда синхронный (тактируемый) триггер. Как и в *RS*-триггере, *JK*-триггер имеет отдельные входы установки J (аналог S) и сброса K (аналог R). Если $J = 1$ и $K = 0$, то тактовым импульсом можно добиться переключения из 0 в 1. Если $K = 1$, $J = 0$, то тактовым импульсом триггер переключается из 1 в 0. В отличие от *RS*-триггера комбинация $J = K = 1$ не просто разрешена, но специально предусмотрена. При этой комбинации входных сигналов триггер превращается в *T*-триггер, переключаясь по каждому импульсу на входе *C*. *JK*-триггер часто называют универсальным триггером.

Реальные микросхемы строятся в виде триггеров с инверсным динамическим управлением, то есть все переключения в триггере происходят по заднему фронту тактирующего импульса. Графическое обозначение *JK*-триггера с динамическим управлением и временные диаграммы в режиме *T*-триггера приведены на рис. В.18.

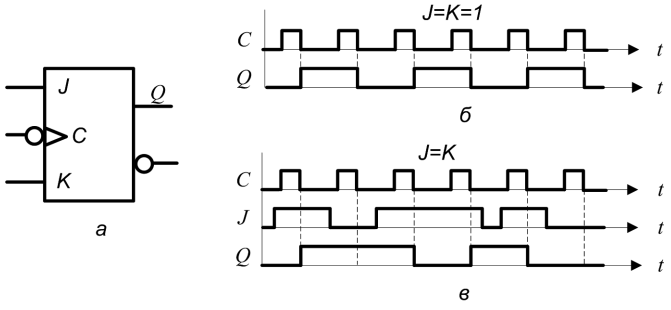


Рис. В.18. *JK*-триггер с динамическим управлением: *а* — условное графическое обозначение; *б* — временная диаграмма работы в асинхронном режиме *T*-триггера; *в* — временная диаграмма работы в синхронном режиме *T*-триггера

D-триггеры

Одним из наиболее распространенных видов триггеров является D-триггер. Он имеет один информационный вход D и один тактирующий вход C. По сигналу синхронизации в D-триггер переписывается информация, которая в данный момент присутствует на входе D. По определению такой триггер может быть только синхронным. Информация на выходе остается неизменной вплоть до прихода следующего импульса синхронизации. Как правило, в интегральном исполнении выпускаются D-триггеры с динамическим управлением, в которых перезапись информации с входа D происходит по фронту тактирующего сигнала. На рис. В.19 приведено условное обозначение и временная диаграмма работы *D*-триггера.

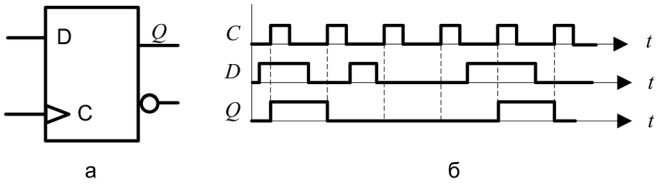


Рис. В.19. *D*-триггер с динамическим управлением: *а* — условное графическое обозначение; *б* — временная диаграмма работы

DV-триггеры

DV-триггер представляет собой модификацию *D*-триггера (рис. В.20). В *D*-триггере записанная информация не может храниться более одного периода синхронизации. *DV*-триггер при $V = 1$ функционирует как обычный *D*-триггер, а при $V = 0$ — переходит в режим хранения информации независимо от смены сигналов на входе *D*.

Наличие входа *V* расширяет функциональные возможности *D*-триггера, позволяя в нужные моменты сохранять информацию на выходах в течение требуемого числа тактовых периодов.

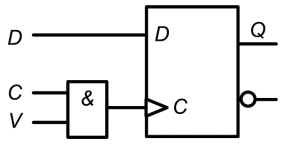


Рис. В.20. Логическая схема *DV*-триггера

Т-триггеры

При построении счетчиков возникает необходимость в триггере, меняющем свое состояние с каждым сигналом на входе Т. Т-триггер, или счетный триггер, имеет один информационный вход Т. Диаграммы функционирования Т-триггера в асинхронном и синхронном режимах приводились при рассмотрении JK-триггеров.

Несколько вариантов построения *T*-триггера показаны на рис. В.21.

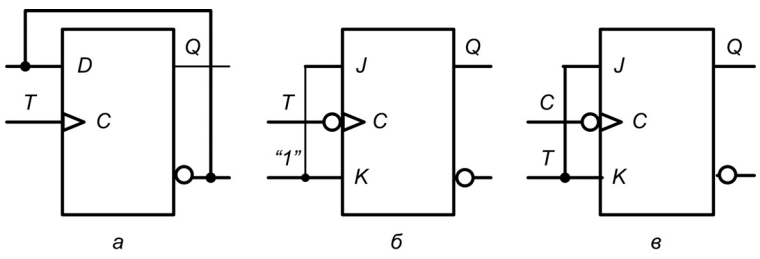


Рис. В.21. Варианты реализации *T*-триггеров: *а* — асинхронный на базе *D*-триггера; *б* — асинхронный на базе *JK*-триггера; *в* — синхронный на базе *JK*-триггера

TV-триггеры

Триггер *TV*-типа кроме счетного входа *T* имеет управляющий вход *V* для разрешения приема информации. Асинхронные и синхронные *TV*-триггеры могут быть получены на базе *JK*-триггера. Ниже показаны схемы асинхронного (рис. В.22, *а*) и синхронного (рис. В.22, *б*) *TV*-триггеров.

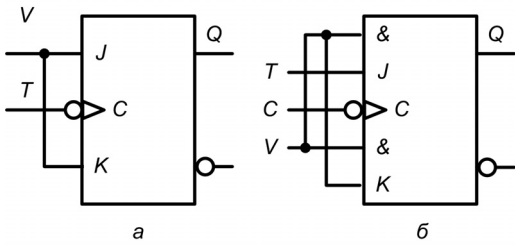


Рис. В.22. Варианты реализации *TV*-триггеров: а — асинхронного; б — синхронного

Двухступенчатые триггеры

При построении схем, представляющих собой цепочку взаимосвязанных триггеров, например регистров сдвига, выходные сигналы предшествующего триггера являются входными сигналами для последующего триггера. В этих условиях необходимо, чтобы значения выходных сигналов триггера на то время, пока производится их запись в другой триггер, не изменялись. Данная проблема решается с помощью двухступенчатых триггеров.

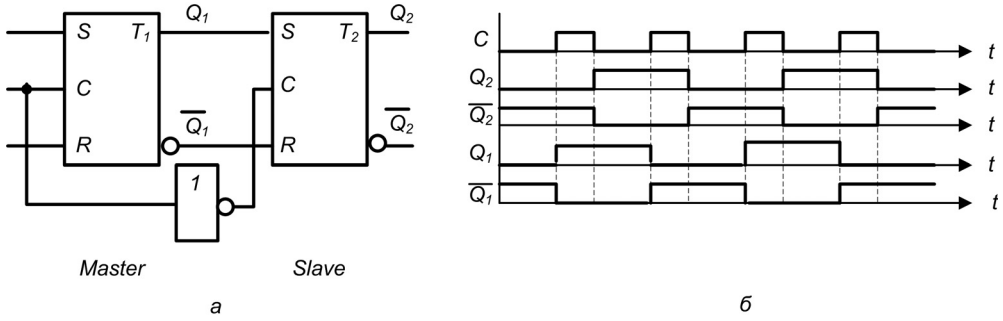


Рис. В.23. Двухступенчатый триггер: *а* — логическая схема; *б* — диаграмма работы

Двухступенчатый триггер состоит из двух последовательно соединенных ступеней, причем каждая ступень содержит по синхронному *RS*-триггеру (рис. В.23, *а*). Первая ступень — ведущая, или М-секция (М происходит от английского Master — хозяин) принимает информацию с входных линий *S* и *R*. Состояние выходов ведущей ступени подается на вторую ступень — ведомую, или *S*-секцию (*S* происходит от английского Slave — раб).

Состояние выходов ведущего триггера изменяется в момент появления положительного импульса синхронизации, и эти изменения будут переданы на входы ведомого триггера. Однако никакие изменения на выходе ведомого триггера не будут происходить до тех пор, пока не появится положительный сигнал инвертированного импульса синхронизации, то есть отрицательный (задний) фронт исходного синхроимпульса. Следовательно, изменения на выходах Q_2 и $\overline{Q_2}$ не произойдут до тех пор, пока не завершится импульс синхронизации. На рис. В.23, *б* показаны временные диаграммы работы триггера.

Двухступенчатый триггер в стандарте ГОСТ обычно обозначают двумя буквами *ТТ*.

Приложение Г

Синтез и анализ комбинационных схем

Комбинационными схемами называются схемные реализации логических функций любого вида, логическое состояние выходов которых зависит только от комбинации логических сигналов на входах в данный момент времени. Таким образом, к комбинационным относят схемы, не обладающие свойством памяти.

Обычно комбинационные схемы реализуются на основе ДНФ и КНФ. В качестве базиса может быть использован любой функционально полный базис, но для удобства изложения мы будем в дальнейшем ориентироваться на булев базис.

При работе с цифровыми устройствами обычно решают две задачи: создание логической схемы по аналитическому описанию соответствующей ФАЛ; получение аналитического описания ФАЛ по логической схеме устройства. Первая задача носит название *задачи синтеза*, вторая — *задачи анализа*.

Синтез комбинационных схем

Обычно исходным пунктом для синтеза служит описание ФАЛ в виде минимальной ДНФ или минимальной КНФ.

Синтез логических устройств в булевом базисе

Булевым называют базис, образуемый элементами «И», «ИЛИ», «НЕ».

При схемной реализации ДНФ каждой элементарной конъюнкции соответствует элемент «И», число входов которого определяется количеством переменных в данной конъюнкции. Все выходы элементов «И» объединяются на элементе «ИЛИ», причем число входов этого элемента равно числу элементарных конъюнкций в ДНФ.

При схемной реализации КНФ каждой элементарной дизъюнкции соответствует элемент «ИЛИ», число входов которого определяется количеством переменных в данной дизъюнкции. Все выходы элементов «ИЛИ» объединяются на элементе «И», причем число входов этого элемента равно числу элементарных дизъюнкций в КНФ.

Инверсия переменных реализуется путем подключения логических элементов «НЕ» на те входы логических элементов «И» («ИЛИ»), куда должны быть поданы переменные с инверсией.

Таким образом, задача синтеза сводится к построению логической схемы, реализующей заданную функцию алгебры логики. Для построения логической схемы необходимо элементы, реализующие логические операции, указанные в ФАЛ, располагать в порядке, заданном булевым выражением. В качестве примера рассмотрим функцию четырех переменных, приведенную в табл. Б.7. Напомним, что минимальная ДНФ для этой функции имеет вид $f = \overline{x_1}x_2x_4 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4}$. Логическую схему будем строить в базисе «И», «ИЛИ», «НЕ». Из выражения видно, что понадобятся 4 схемы «НЕ» (для получения инверсных значений аргументов), одна трехвходовая схема «И» (для первой конъюнкции), две двухвходовых схемы «И» (для получения второй и третьей конъюнкций) и одна трехвходовая схема «ИЛИ». В соответствии с МДНФ получаем логическую схему, приведенную на рис. Г.1.

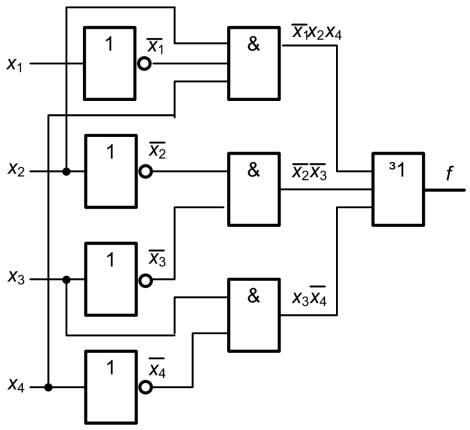


Рис. Г.1. Логическая схема устройства в базисе «И», «ИЛИ», «НЕ»

Синтез логических устройств в заданном базисе

Для сокращения номенклатуры используемых микросхем вместо булева базиса часто используют функционально полные наборы «И-НЕ» либо «ИЛИ-НЕ». Для этого ДНФ или КНФ записывают в заданном базисе.

Если задан базис «И-НЕ», то путем двойного инвертирования исходного выражения или его части (с последующим применением теоремы де Моргана) логическая функция приводится к виду, содержащему только операции логического умножения и инвертирования. В качестве примера синтезируем устройство, реализующее логическую функцию из предшествующего примера. Сначала переведем минимальную ДНФ в базис «И-НЕ».

$$f = \overline{x_1}x_2x_4 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4} = \overline{\overline{x_1}x_2x_4 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4}} = \overline{\overline{x_1}x_2x_4 \cdot \overline{x_2} \cdot \overline{x_3} \cdot x_3\overline{x_4}}.$$

Теперь, используя полученное представление функции, изобразим логическую схему устройства (рис. Г.2). Приведем ее в стандарте ANSI.

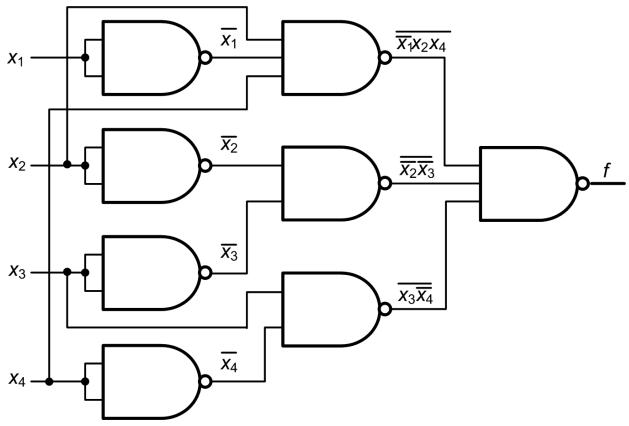


Рис. Г.2. Логическая схема устройства в базисе «И-НЕ»

Поступая аналогично, получим выражение в базисе «ИЛИ-НЕ»:

$$f = \overline{x_1}x_2x_4 \vee \overline{x_2}\cdot\overline{x_3} \vee x_3\overline{x_4} = \overline{\overline{x_1}x_2x_4 \vee \overline{x_2}\cdot\overline{x_3} \vee x_3\overline{x_4}} =$$

$$= x_1 \vee \overline{x_2} \vee \overline{x_4} \vee \overline{x_2} \vee x_3 \vee \overline{x_3} \vee x_4.$$

Соответствующая логическая схема, изображенная в стандарте DIN, показана на рис. Г.3.

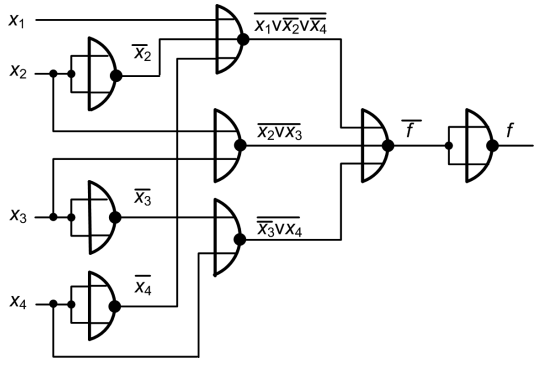


Рис. Г.3. Логическая схема устройства в базисе «ИЛИ-НЕ»

Анализ комбинационных схем

Задача анализа заключается в определении функции *f*, реализуемой заданной логической схемой. При решении данной задачи обычно придерживаются следующей последовательности действий.

9. Схема разбивается на ярусы. Ярусам присваиваются последовательные номера.
10. Выходы каждого логического элемента обозначаются названием искомой функции, снабженным цифровым индексом, где первая цифра — номер яруса, а остальные цифры — порядковый номер элемента в ярусе.
11. Для каждого элемента записывается аналитическое выражение, связывающее его выходную функцию с входными переменными. Выражение определяется логической функцией, реализуемой данным элементом.
12. Производится подстановка одних функций в другие, пока не получится булева функция, выраженная через входные переменные.

Используя приведенную последовательность действий, произведем анализ ранее синтезированной схемы (рис. Г.1). Сначала разобьем ее на ярусы. Пронумеровав получившиеся ярусы, введем обозначения для каждой выходной функции (рис. Г.4).

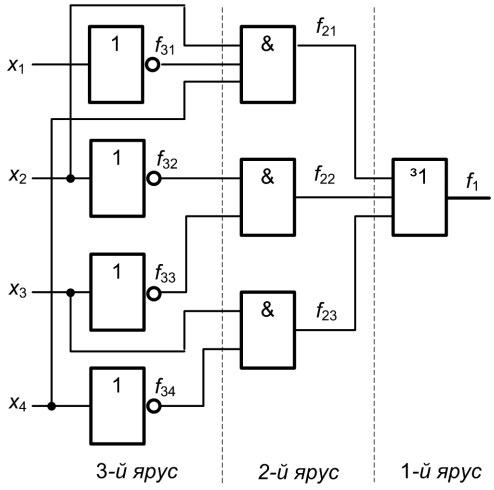


Рис. Г.4. Логическая схема устройства с разбивкой на ярусы

Запишем все функции, начиная с 1-го яруса:

$$f_1 = f_{21} \vee f_{22} \vee f_{23};$$
$$f_{21} = f_{31}x_2x_4, \quad f_{22} = f_{32}f_{33}, \quad f_{23} = x_3f_{34};$$
$$f_{31} = \overline{x_1}, \quad f_{32} = \overline{x_2}, \quad f_{33} = \overline{x_3}, \quad f_{34} = \overline{x_4}.$$

Теперь запишем все функции, подставляя входные переменные x_1, x_2, x_3 и x_4:

$$f_{21} = \overline{x_1}x_2x_4, \quad f_{22} = \overline{x_2} \cdot \overline{x_3}, \quad f_{23} = x_3\overline{x_4}.$$

В итоге получим выходную функцию:

$$f = f_1 = \overline{x_1}x_2x_4 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4}.$$