Заключение

Любую работу трудно начинать и еще труднее заканчивать, но приходится...

Наши поздравления уважаемому читателю — надеемся, что вы оказались на этой странице не в силу природного любопытства и нетерпеливости, а в результате изучения всего материала учебника.

Теперь вы вооружены и опасны ☺. Вооружены базовыми знаниями в данной предметной области, а опасны для дилетантов и «незнаек», то есть людей несведущих. И конечно, вы открыты новым знаниям, тому, что у нас впереди. Это очень важно, ведь темпы развития в этой области знаний предельно высоки. Специалист по вычислительным машинам и системам должен быть готов к обучению на протяжении всей профессиональной жизни: поезд новых компьютерных решений движется чрезвычайно стремительно, только успевай впрыгивать на его подножку!

Мы намеренно не употребляли слово «электронные» применительно к вычислительным машинам. Перефразируя известное высказывание, электроника — колыбель вычислительных машин, но нельзя же вечно жить в колыбели! Двадцать первый век принесет массу сюрпризов в области элементной базы ВМ и ВС, а вместе с ее изменениями переменится архитектура и организация вычислительных средств. Вот краткий список тех новаций, которые уже стучатся в дверь: голографическая, твердотельная и протонная память; схемы на базе молекулярных ключей; оптические, квантовые и нанокомпьютеры; электронная цифровая бумага; пластмассовые дисплеи; нейроинформатика, биоинформатика... И это еще далеко не все. Словом, дорога в компьютерный космос открыта, а информационная революция только начинается... Будьте готовы к переменам, и все у вас получится ☺.

Впереди длинный и интересный путь познаний. Удачи Вам, Уважаемый Читатель, на этом пути!

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:

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

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 OCT 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:

$$\begin{array}{l} 010\ \ 101\ \ 101\ \ 101 \\ (2\quad 5\quad 5,\quad 5)_8 = (10101101{,}101)_2; \\ 1010\ \ 1101\ \ 1010 \\ (\mathrm{A}\quad \mathrm{D},\quad \mathrm{A})_{16} = (10101101{,}1010)_2. \end{array}$$

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):

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

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

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

по правилам арифметики системы счисления с основанием h. Так, например, 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 $(x_{r-1}\ldots x_1x_0)_s$. Требуется получить запись числа в системе счисления с основанием h:

$$(X)_h = (y_{w-1}\ldots y_1y_0)_h = y_{w-1}h^{w-1} + y_{w-2}h^{w-2} + \ldots + y_1h^1 + y_0h^0.$$

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).

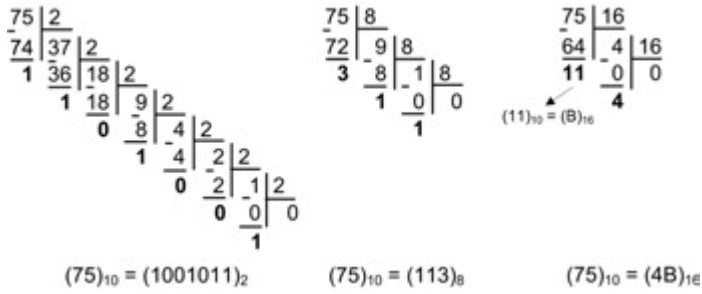


Рис. А.1. Пример перевода целого десятичного числа 75 в 2-ичную, 8-ричную и 16-ричную системы счисления

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 знаках 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.

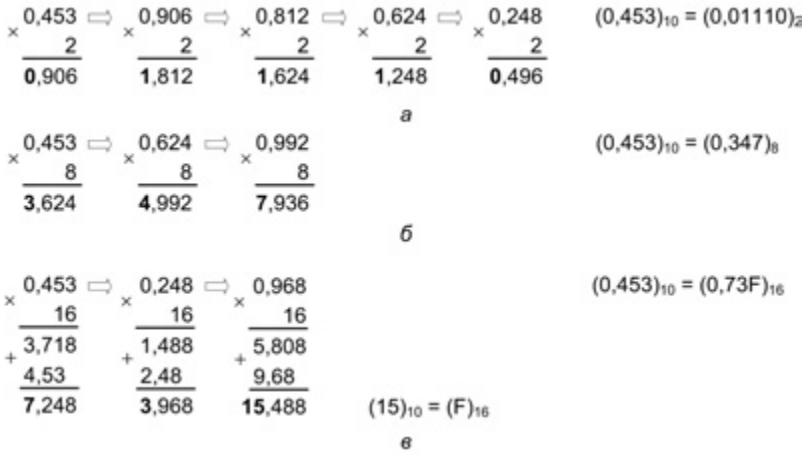


Рис. А.2. Пример перевода десятичной дроби 0,453 в 2-ичную, 8-ричную и 16-ричную системы счисления

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*.

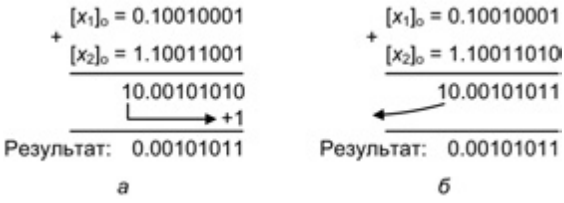


Рис. А.3. Пример сложения чисел: *а* — в обратном коде; *б* — в дополнительном коде

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).

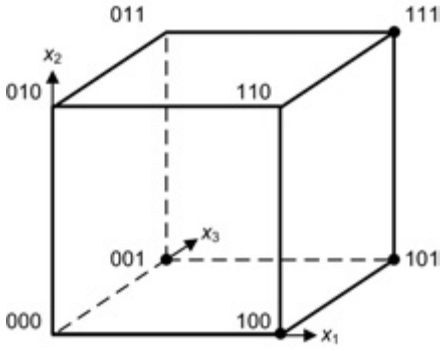


Рис. Б.2. Геометрическое представление булевой функции

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

продолжение ⇗

[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.

1. **Аксиомы конъюнкции**

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

2. **Аксиомы дизъюнкции**

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

3. **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».

4. **Теоремы исключения констант**

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

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

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

Для n переменных:

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

Повторное применение не дает ничего нового.

6. **Теорема противоречия**

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

Невозможно, чтобы противоречащие высказывания были одновременно истинными.

7. **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.

8. **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.

9. **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)**

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

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)**

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

14. **Закон склеивания (исключения)**

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

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 $f = x_4 x_1 \vee x_3 x_2 \vee \overline{x_3} x_1 \vee \overline{x_4} \overline{x_3} \overline{x_2}$, 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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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	$x_2 x_3 \overline{x_4}$ *

продолжение ⇩

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	$\overline{x_1}x_2x_4$				√		√				
6,7	$\overline{x_1}x_2x_3$					√	√				

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* 2* 8*	0001 0010 1000
2	5* 6* 9* 10*	0101 0110 1001 1010
3	7* 14*	0111 1110

Таблица Б.16. Первый этап склеивания членов групп

Группы	Склеиваемый наборы	Результат
0–1	0,1* 0,2* 0,8*	000– 00–0 –000
1–2	1,5 1,9* 2,6* 2,10* 8,9* 8,10*	0–01 –001 0–10 –010 100– 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:

$$\overline{x_2} \cdot \overline{x_3} = A; \quad \overline{x_2} \cdot \overline{x_4} = B; \quad x_3 \overline{x_4} = C;$$
$$\overline{x_1} \cdot \overline{x_3} x_4 = D; \quad \overline{x_1} x_2 x_4 = E; \quad \overline{x_1} x_2 x_3 = F.$$

		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 $f = \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4} \vee x_1 \cdot \overline{x_3}x_4 \vee \overline{x_1}x_2x_3$. 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.

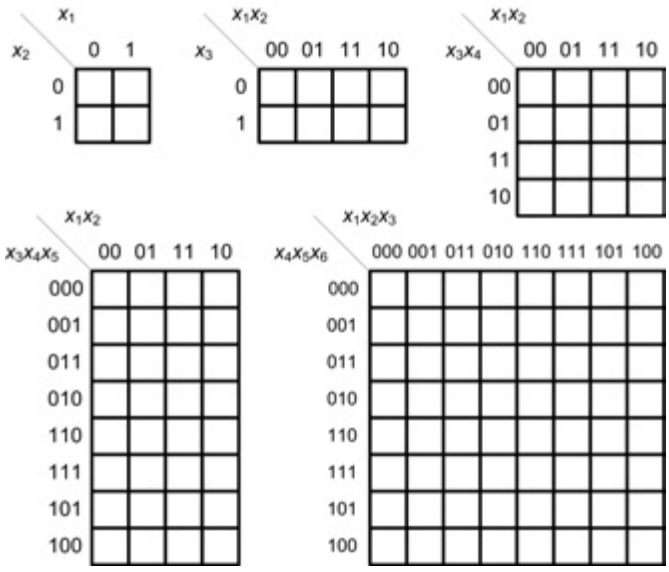


Рис. Б.3. Карты Карно для 2, 3, 4, 5 и 6 аргументов

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

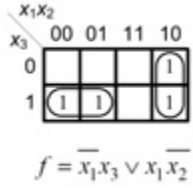


Рис. Б.4. Пример минимизации ФАЛ трех аргументов

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

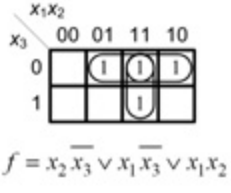


Рис. Б.5. Пример использования общей клетки в нескольких контурах

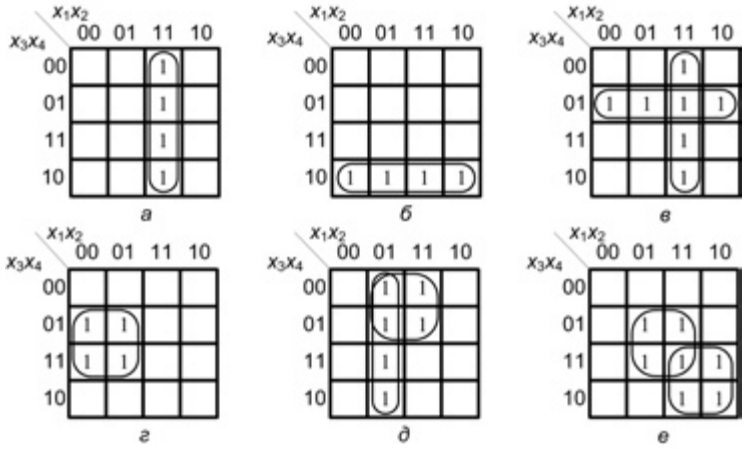


Рис. Б.6. Варианты группировки клеток для функции 4-х аргументов: *а* — $f = x_1x_2$; *б* — $f = x_3\overline{x_4}$; *в* — $f = x_1x_2 \vee \overline{x_3}x_4$; *г* — $f = \overline{x_1}x_4$; *д* — $f = x_2\overline{x_3} \vee \overline{x_1}x_2$; *е* — $f = x_2x_4 \vee x_1x_3$

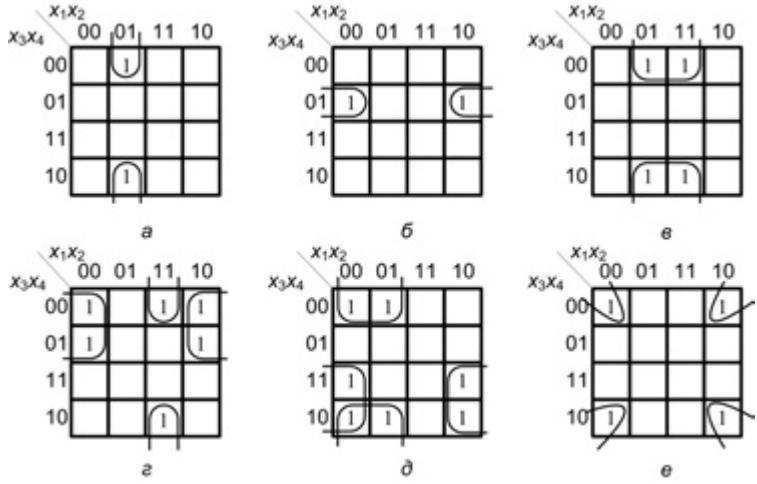


Рис. Б.7. Варианты группировки клеток, использующие циклический характер карт:
а — $f = \overline{x_1}x_2\overline{x_4}$; *б* — $f = \overline{x_2} \cdot \overline{x_3}x_4$; *в* — $f = x_2\overline{x_4}$; *г* — $f = \overline{x_2} \cdot \overline{x_3} \vee x_1x_2\overline{x_4}$;
д — $f = \overline{x_1} \cdot x_4 \vee \overline{x_2}x_3$; *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*).

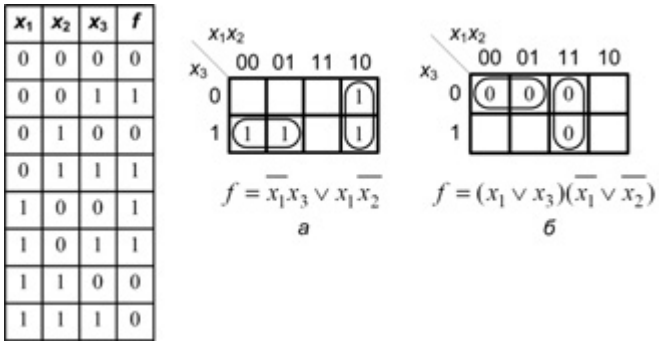


Рис. Б.8. Примеры минимизации ФАЛ с получением минимальной: *а* — ДНФ; *б* — КНФ

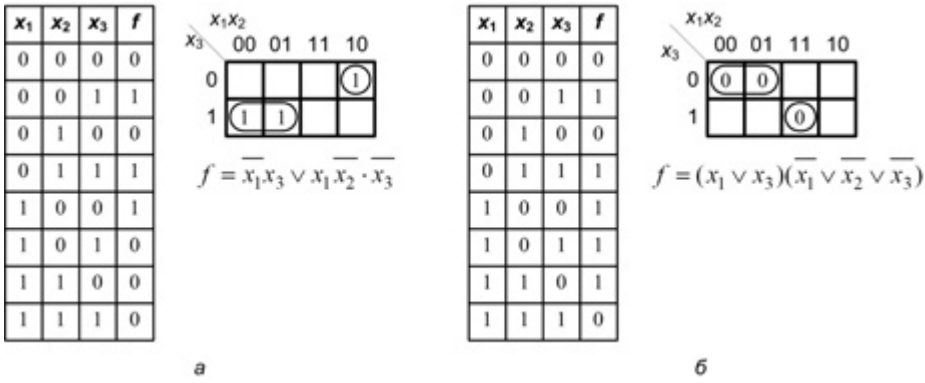


Рис. Б.9. Учет клеток, не имеющих в карте Карно соседних клеток: *а* — ДНФ; *б* — КНФ

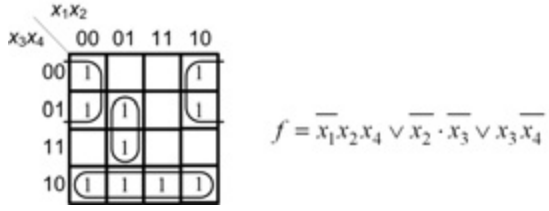


Рис. Б.10. Минимизация функции из табл. Б.7

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	*

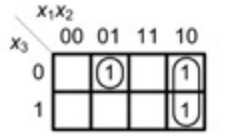
На рис. Б.11 показаны варианты доопределения данной функции.

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.

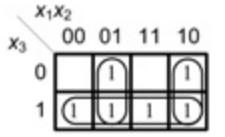
x_1	x_2	x_3	x_4	f	x_1	x_2	x_3	x_4	f
0	0	0	0	1	1	0	0	0	*
0	0	0	1	*	1	0	0	1	1
0	0	1	0	*	1	0	1	0	0
0	0	1	1	0	1	0	1	1	*
0	1	0	0	*	1	1	0	0	0
0	1	0	1	0	1	1	0	1	*
0	1	1	0	1	1	1	1	0	1
0	1	1	1	*	1	1	1	1	*

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



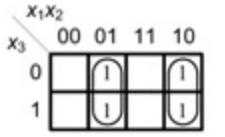
а

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).

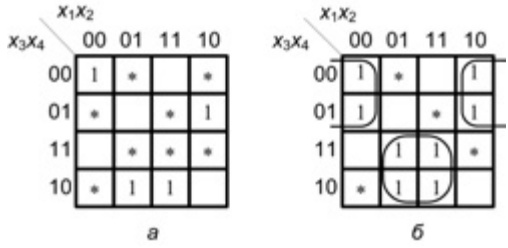


Рис. Б.12. Частично определенная функция: *а* — перед доопределением; *б* — после доопределения $f = \overline{x_2} \cdot \overline{x_3} \vee x_2 x_3$

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: $\{x_1\overline{x_2}x_{3f_1}, x_1x_2\overline{x_3}_{f_1}, x_1\overline{x_2}\cdot\overline{x_3}_{f_1}, x_1x_2x_{3f_2}, x_1x_2\overline{x_3}_{f_2}, x_1\overline{x_2}x_{3f_2}\}$.

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: $x_1\overline{x_2}x_{3\,f_1f_2}$, $x_1x_2\overline{x_3}_{\,f_1f_2}$, $x_1\overline{x_2}_{\,f_1}$, $x_1x_{3\,f_2}$, $x_1\overline{x_3}_{\,f_1}$, $x_1x_{2\,f_2}$.

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
$x_1\overline{x_2}x_{3\,f_1f_2}$	√	√				
$x_1x_2\overline{x_3}_{\,f_1f_2}$			√	√		
$x_1\overline{x_2}_{\,f_1}$	√				√	
$x_1x_{3\,f_2}$		√				√
$x_1\overline{x_3}_{\,f_1}$			√		√	
$x_1x_{2\,f_2}$				√		√

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 $x_1\overline{x_2}x_{3\,f_1f_2}$, $x_1\overline{x_3}_{\,f_1}$, $x_1x_{2\,f_2}$. 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);

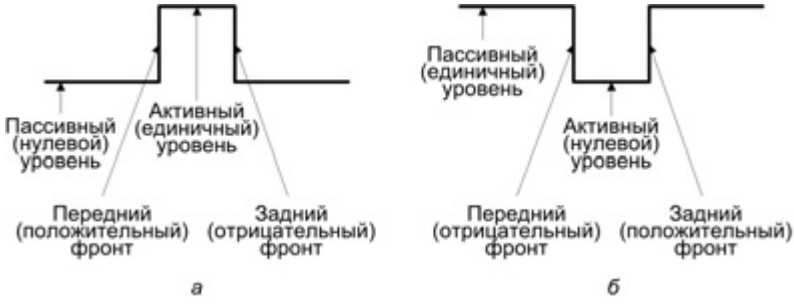


Рис. В.1. Основные параметры цифровых сигналов: *а* — в положительной логике; *б* — в отрицательной логике

- 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.

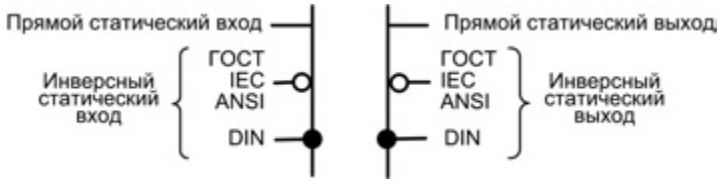


Рис. В.2. Типовое изображение входов и выходов на условном обозначении логических элементов

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.

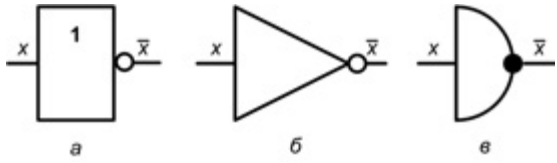


Рис. В.3. Условные обозначения логического элемента «НЕ» в стандартах:
а — ГОСТ и IEC; *б* — ANSI; *в* — DIN

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.

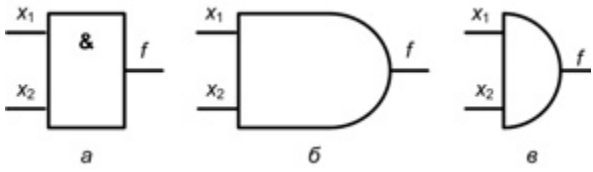


Рис. В.4. Условные обозначения логического элемента «И» в стандартах:
а — ГОСТ и IEC; *б* — ANSI; *в* — DIN

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 международном стандартом IEC.

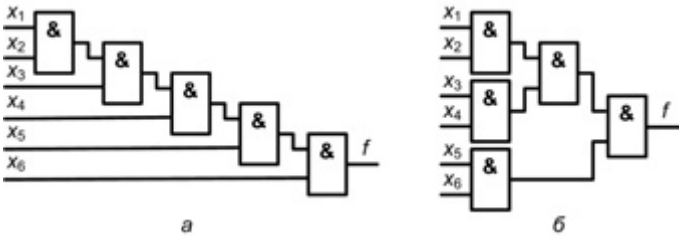


Рис. В.5. Варианты построения 6-входовой схемы «И» на базе двухвходовых схем: *а* — каскадный; *б* — параллельный

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 ($f = x_1 \vee x_2 \vee ... \vee x_n$).

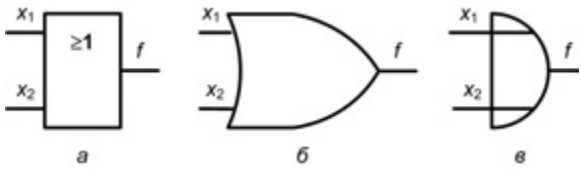


Рис. В.6. Условные обозначения логического элемента «ИЛИ» в стандартах: *а* — ГОСТ и IEC; *б* — ANSI; *в* — DIN

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.

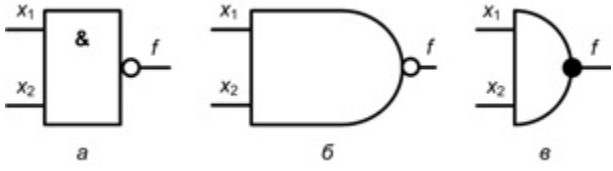


Рис. В.7. Условные обозначения логического элемента «И-НЕ»: *а* — ГОСТ и IEC; *б* — ANSI; *в* — DIN

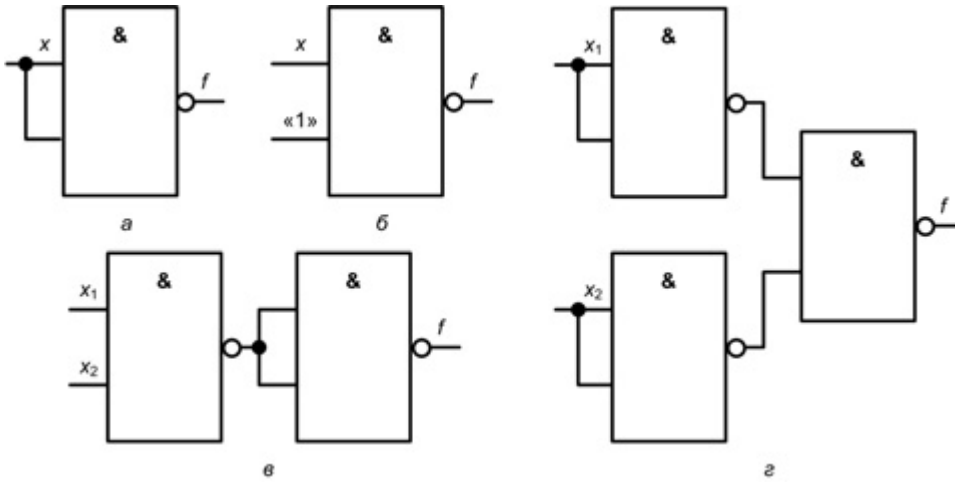


Рис. В.8. Реализация на базе логического элемента «И-НЕ» функций: *а, б* — «НЕ»; *в* — «И»; *г* — «ИЛИ»

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.

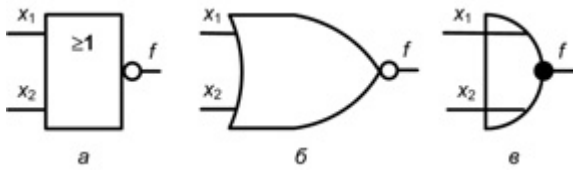


Рис. В.9. Условные обозначения логического элемента «ИЛИ-НЕ»: *а* — ГОСТ и IEC; *б* — ANSI; *в* — DIN

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».

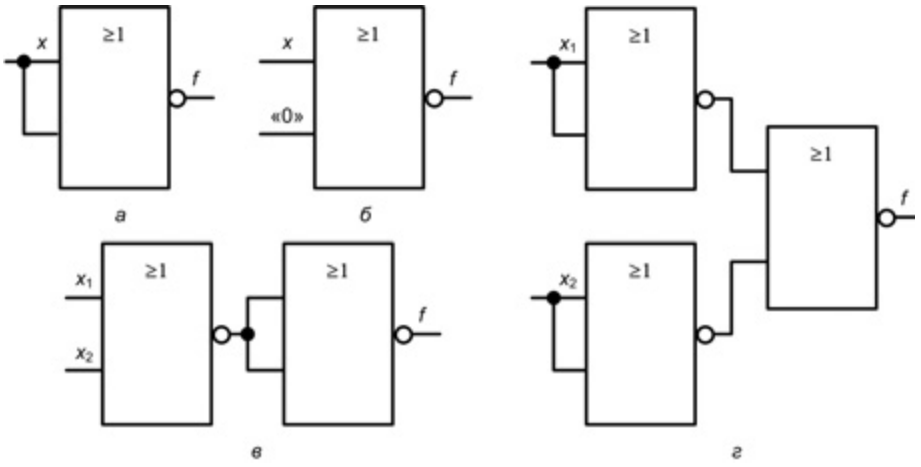


Рис. В.10. Реализация на базе логического элемента «ИЛИ-НЕ» функций: *а, б* — «НЕ»; *в* — «ИЛИ»; *г* — «И»

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.

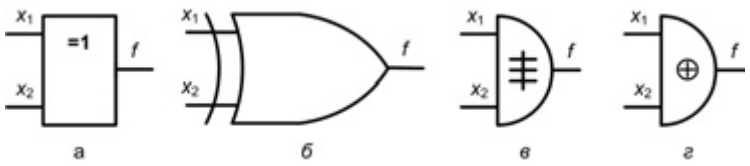


Рис. В.11. Условные обозначения логического элемента «Исключающее ИЛИ» в стандартах: *а* — ГОСТ и IEC; *б* — ANSI; *в*, *г* — DIN

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.

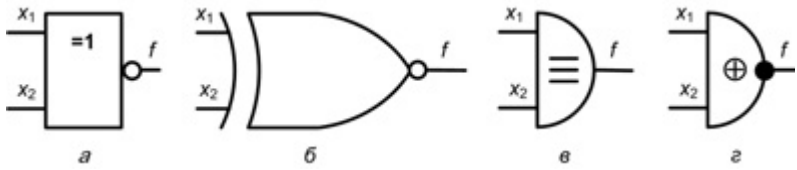


Рис. В.12. Условные обозначения логического элемента «Исключающее ИЛИ» с инверсным выходом в стандартах: *а* — ГОСТ и IEC; *б* — ANSI; *в*, *г* — DIN

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).

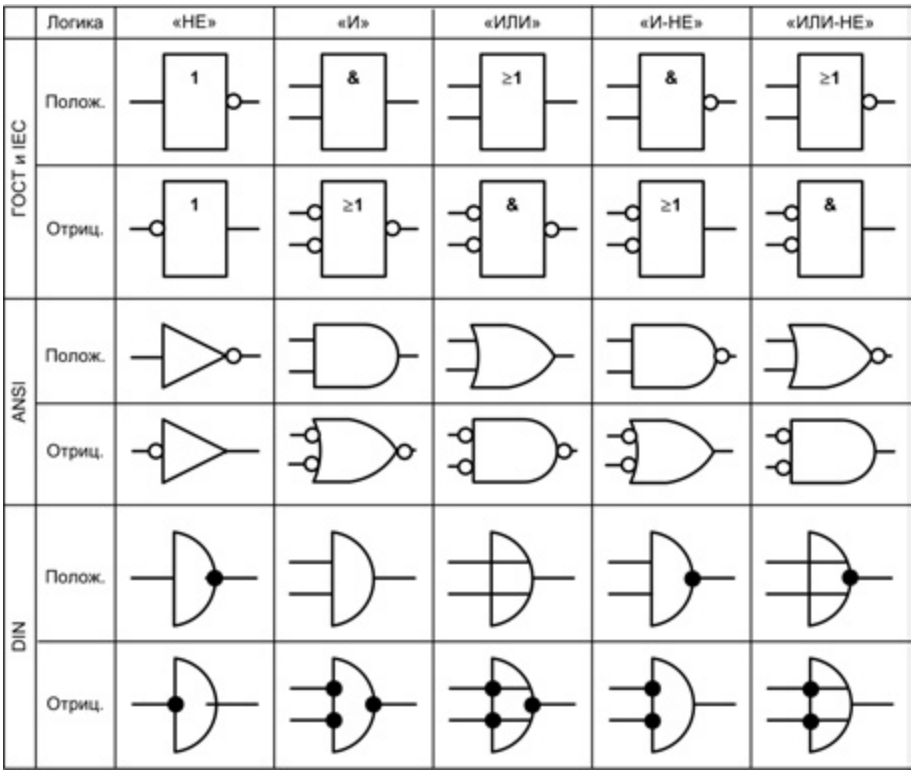


Рис. В.13. Условные графические изображения логических элементов в положительной и отрицательной логике

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).

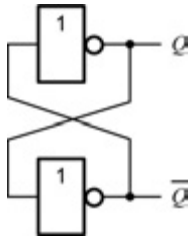


Рис. В.14. Простейший бистабильный элемент

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.

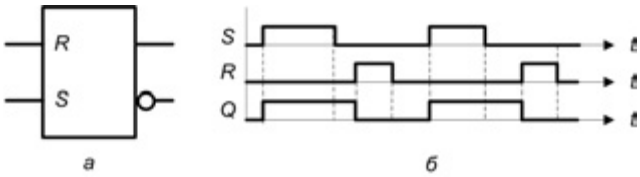


Рис. В.16. Асинхронный *RS*-триггер: *а* — условное графическое обозначение; *б* — временная диаграмма работы

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.

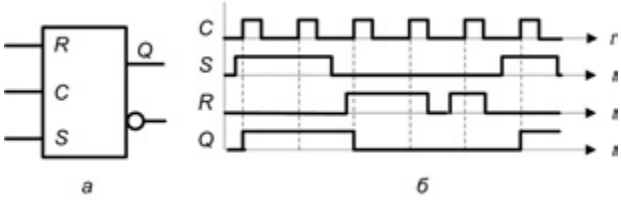


Рис. В.17. Синхронный *RS*-триггер: *а* — условное графическое обозначение; *б* — временная диаграмма работы

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.

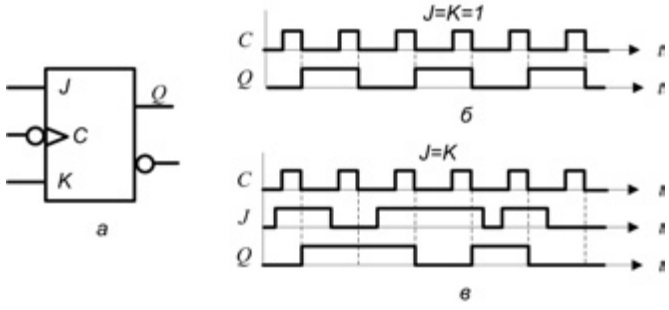


Рис. В.18. *JK*-триггер с динамическим управлением: *а* — условное графическое обозначение; *б* — временная диаграмма работы в асинхронном режиме *T*-триггера; *в* — временная диаграмма работы в синхронном режиме *T*-триггера

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.

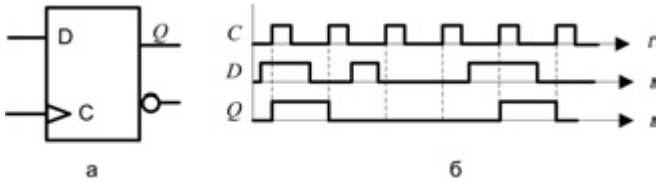


Рис. В.19. *D*-триггер с динамическим управлением: *а* — условное графическое обозначение; *б* — временная диаграмма работы

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.

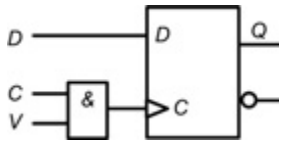


Рис. В.20. Логическая схема *DV*-триггера

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.

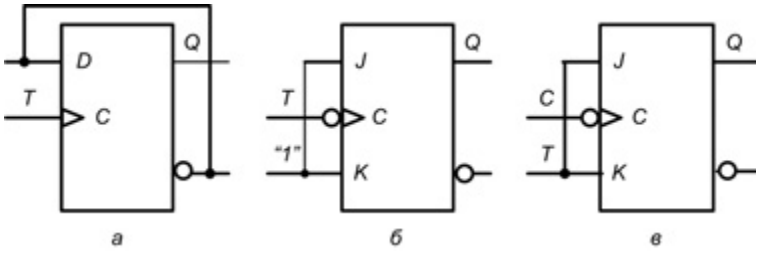


Рис. В.21. Варианты реализации *T*-триггеров: *а* — асинхронный на базе *D*-триггера; *б* — асинхронный на базе *JK*-триггера; *в* — синхронный на базе *JK*-триггера

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.

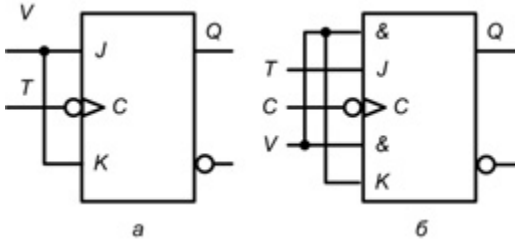


Рис. В.22. Варианты реализации *TV*-триггеров: а — асинхронного; б — синхронного

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.

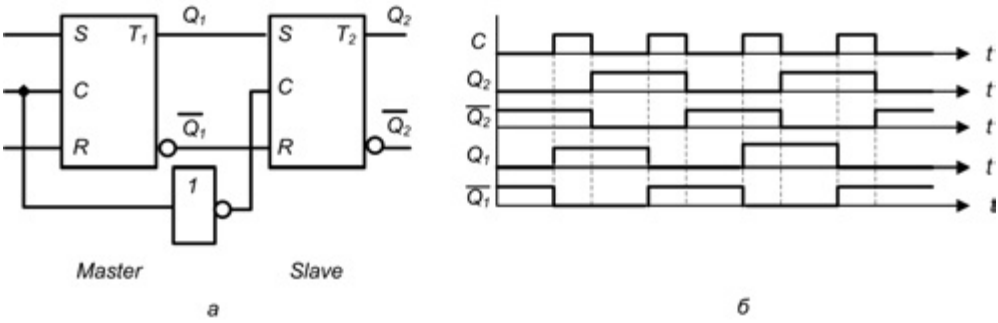


Рис. В.23. Двухступенчатый триггер: *а* — логическая схема; *б* — диаграмма работы

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.

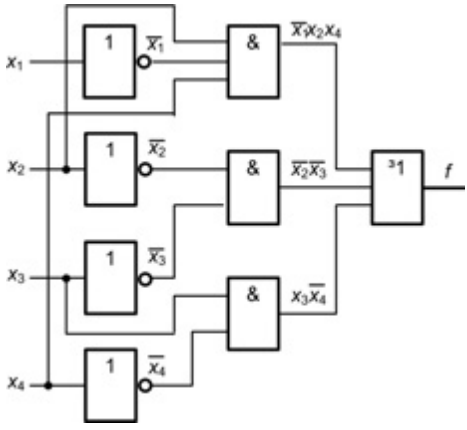


Рис. Г.1. Логическая схема устройства в базисе «И», «ИЛИ», «НЕ»

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».

$$f = \overline{x_1}x_2x_4 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4} = \overline{\overline{x_1}x_2x_4 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4}} = \overline{\overline{x_1}x_2x_4} \cdot \overline{\overline{x_2} \cdot \overline{x_3}} \cdot \overline{x_3\overline{x_4}}.$$

Теперь, используя полученное представление функции, изобразим логическую схему устройства (рис. Г.2). Приведем ее в стандарте ANSI.

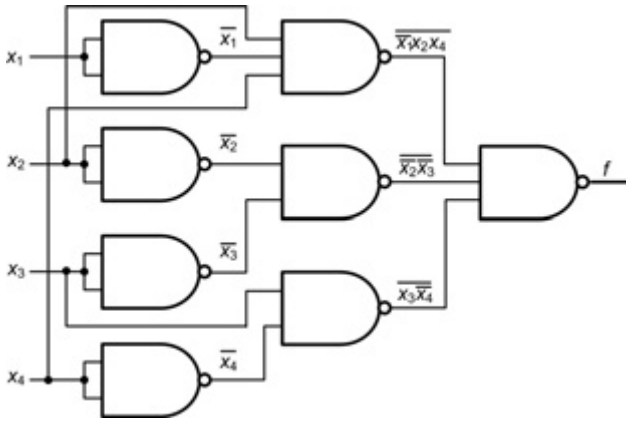


Рис. Г.2. Логическая схема устройства в базисе «И-НЕ»

Поступая аналогично, получим выражение в базисе «ИЛИ-НЕ»:

$$f = \overline{x_1}x_2x_4 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4} = \overline{\overline{\overline{x_1}x_2x_4 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4}}} =$$

$$= \overline{\overline{x_1 \vee \overline{x_2} \vee \overline{x_4}} \vee \overline{x_2 \vee x_3} \vee \overline{\overline{x_3} \vee x_4}}.$$

Соответствующая логическая схема, изображенная в стандарте DIN, показана на рис. Г.3.

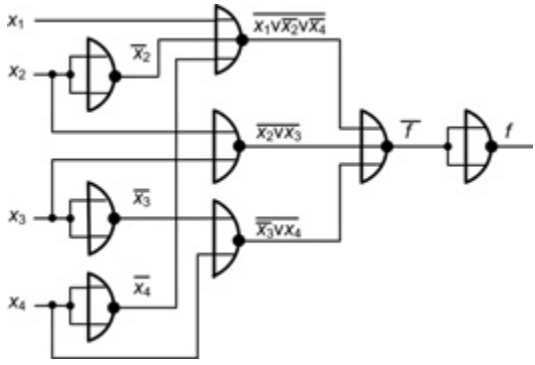


Рис. Г.3. Логическая схема устройства в базисе «ИЛИ-НЕ»

Анализ комбинационных схем

Задача анализа заключается в определении функции f, реализуемой заданной логической схемой. При решении данной задачи обычно придерживаются следующей последовательности действий.

1. Схема разбивается на ярусы. Ярусам присваиваются последовательные номера.
2. Выходы каждого логического элемента обозначаются названием искомой функции, снабженным цифровым индексом, где первая цифра — номер яруса, а остальные цифры — порядковый номер элемента в ярусе.
3. Для каждого элемента записывается аналитическое выражение, связывающее его выходную функцию с входными переменными. Выражение определяется логической функцией, реализуемой данным элементом.
4. 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).

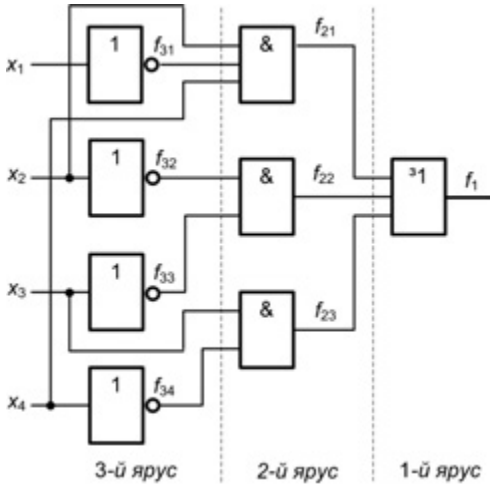


Рис. Г.4. Логическая схема устройства с разбивкой на ярусы

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:

$$f = f_1 = \overline{x_1}x_2x_4 \vee \overline{x_2} \cdot \overline{x_3} \vee x_3\overline{x_4}.$$