Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информационные технологии»

И. А. Мурашко

ЭВМ И ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА

КУРС ЛЕКЦИЙ
по одноименной дисциплине
для студентов специальности 1-40 01 02
«Информационные системы и технологии»
дневной формы обучения

Электронный аналог печатного издания

Гомель 2011

УДК 004.3(075.8)
ББК 32.973.26-04я73
М91

Рекомендовано к изданию научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 10 от 28.06.2010 г.*)

Рецензент: канд. техн. наук, доц. каф. «Автоматизированный электропривод»
ГГТУ им. П. О. Сухого *В. С. Захаренко*

Мурашко, И. А.

М91 ЭВМ и периферийные устройства : курс лекций по одноим. дисциплине для студентов специальности 1-40 01 02 «Информационные системы и технологии» днев. формы обучения / И. А. Мурашко. – Гомель : ГГТУ им. П. О. Сухого, 2011. – 100 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://lib.gstu.local. – Загл. с титул. экрана.

ISBN 978-985-420-988-3.

Курс лекций содержит основной теоретический материал дисциплины «ЭВМ и периферийные устройства». Может использоваться для самостоятельного изучения принципов работы цифровых вычислительных машин.

Для студентов специальности 1-40 01 02 «Информационные системы и технологии» дневной формы обучения.

УДК 004.3(075.8)
ББК 32.973.26-04я73

ISBN 978-985-420-988-3

ПРЕДИСЛОВИЕ

Курс лекций по дисциплине «ЭВМ и периферийные устройства» разработан для студентов специальности 1-40 01 02 «Информационные системы и технологии» (в проектировании и производстве). Он предусматривает изучение аппаратного обеспечения современных ЭВМ и периферийных устройств, так как использование различного рода автоматических и автоматизированных систем на базе ЭВМ невозможно без знания принципов их построения, особенностей работы и технических возможностей. Целью курса лекций является получение студентами базовых знаний о логических основах работы ЭВМ и периферийных устройств, схемотехнике основных функциональных компонентов и аппаратной реализации обработки информации в ЭВМ.

В курсе лекций большое внимание уделяется изучению теоретических основ выполнения логических операций, законов и правил алгебры логики, формам представления логических функций. Кратко рассмотрены проектирования произвольных логических схем и вопросы минимизации логических функций. Приведена классификация базовых элементов памяти ЭВМ – триггеров и рассмотрены схемотехнические особенности реализации триггеров различных типов.

Большое внимание уделено изучению принципов работы и схемных решений для основных функциональных узлов ЭВМ. Приведена классификация функциональных узлов. Рассмотрены вопросы схемотехнической реализации таких функциональных узлов ЭВМ, как регистры, счетчики, сумматоры, компараторы, мультиплексоры, дешифраторы и умножители.

Отдельная глава в книге посвящена памяти ЭВМ. Приведена классификация памяти, рассмотрены схемотехнические особенности различных типов памяти и алгоритмы ее работы. Память в современных компьютерах и периферийных устройствах составляет до 90 % от общего интегральных компонент. Например, в процессоре *Pentium* IV (ядро *McKinley*) 75 % транзисторов задействовано для реализации памяти. В этом процессоре используется 13 типов памяти. Поэтому на долю памяти приходится львиная доля отказов цифровых систем. В связи с этим в рамках курса лекций рассмотрены вопросы тестирования оперативной памяти. Приведена классификация неисправностей и основные маршевые тесты, используемые для их обнаружения.

1. ЛОГИЧЕСКИЕ ОСНОВЫ ЭВМ

1.1. История возникновения алгебры логики

Результатом применения математических методов к проблемам формальной логики стало появление математической логики. Основателем математической логики считается Дж. Буль – автор основополагающих монографий «Математический анализ логики» (1847) и «Исследование законов мысли» (1854). Дальнейшее развитие математическая логика получила в работах английского математика де Моргана. В результате математическая логика оформилась как особая алгебра – *алгебра логики*, или *булева алгебра*. До начала двадцатого века математическая логика оставалась чисто теоретической областью математики. Однако в 1910 г. профессор Петербургского университета П. Эренфест обратил внимание на возможность применения математической логики для описания работы таких технических устройств, как релейно-контактные схемы. Позже, в тридцатых годах, к данному разделу математики проявили интерес не только ученые-математики, но и инженеры (русский инженер В. И. Шестаков (1936), американский математик и инженер К. Э. Шеннон (1938)). Это позволило создать в сороковых годах первую электронную вычислительную машину (ЭВМ). Одной из первых ЭВМ является «*ENIAC*» (США, 1943), созданная Джоном Мокли и Дж. Преспером Эккертом. При построении машины было использовано около 20 тыс. электронных ламп. Эта ЭВМ весила порядка 30 тонн, занимала площадь более 200 м^2 и нуждалась во вспомогательной холодильной установке. Быстродействие ЭВМ составляло около 5000 операций сложения в секунду, оперативная память позволяла хранить 20 десятиразрядных слов.

В настоящее время область применения математической логики расширилась до анализа и синтеза цифровых устройств (в том числе и ЭВМ), без которых невозможна современная жизнедеятельность человека.

1.2. Основные положения алгебры логики

Цифровые устройства работают с сигналами, которые могут принимать ограниченное количество значений, а именно два. В соответствии с ГОСТ 2.743–82, более положительному значению напряжения («*H*»-уровень) соответствует состояние «логическая 1», а менее

положительному значению («*L*»-уровень) соответствует состояние «логический 0». Такое соглашение называется положительной логикой (обратное соотношение называется отрицательной логикой). Положительная логика получила широкое распространение для анализа и синтеза цифровых устройств, поэтому в дальнейшем будет использоваться только положительная логика. Наименования, определения и условные обозначения основных параметров и характеристик цифровых микросхем приведены в ГОСТ 19480–89.

Теоретической основой анализа и проектирования аппаратных средств ЭВМ и периферийных устройств является *алгебра логики*, или *булева алгебра*, которая включает в себя следующие основные понятия:

– *логическая переменная* – переменная, которая может принимать одно из двух значений: *истина* или *ложь* («1» или «0»);

– *логическая константа* – постоянная величина, значением которой может быть: *истина* или *ложь* («1» или «0»);

– *логическая функция* – функция, которая может принимать одно из двух значений: *истина* или *ложь* («1» или «0») в зависимости от текущего значения своих аргументов, в качестве которых используются логические переменные и константы.

Функция f, зависящая от n переменных $x_1, x_2, ..., x_n$, называется булевой, или переключательной, если $f \in \{0, 1\}$ и $x_i \in \{0, 1\}$ ($i = 1, 2, \ldots, n$). Другими словами, булева функция – это функция, значение которой и аргументы принадлежат множеству $\{0, 1\}$.

Логическая функция может быть одного ($n = 1$), двух ($n = 2$) или нескольких ($n > 2$) аргументов. Значение функции определяется комбинацией конкретных значений переменных, от которых она зависит. Данная комбинация аргументов функции называется *набором*. Количество различных наборов N для n переменных определяется по формуле $N = 2^n$. Так, для $n = 3$ существует $2^3 = 8$ различных наборов.

Существует три основных способа задания логической функции – таблицей истинности, словесным описанием или логическим выражением.

Таблица истинности является универсальным средством задания логической функции. Она включает все наборы для заданного количества переменных, определяющих значение логической функции, с указанием значений, которые принимает функция для каждого набора. Пример определения логической функции от трех переменных представлен в табл. 1.1.

Таблица 1.1

Пример задания логической функции $f(x_1, x_2, x_3)$

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Одна таблица истинности может определять несколько логических функций, зависящих от одних и тех же аргументов. Например, пусть имеется четыре функции $y_i = f(x_j)$; $i = \overline{1, 4}$; $j = \overline{1, 3}$. Определить их можно при помощи таблицы истинности (табл. 1.2).

Таблица 1.2

Пример задания логической функции $f(x_1, x_2, x_3)$

x_1	x_2	x_3	y_1	y_2	y_3	y_4
0	0	0	1	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	1	–
0	1	1	0	1	1	–
1	0	0	0	1	1	0
1	0	1	1	1	0	–
1	1	0	0	0	1	1
1	1	1	1	1	1	1

Логическая функция называется «*полностью определенная*», если для нее заданы значения по всем возможным наборам. Функция называется «*частично определенная*», если значения функции заданы не по всем возможным наборам. В табл. 1.2 функции y_1–y_3 являются полностью определенными, а y_4 – частично определенная (иногда вместо «–» записывают x, это означает, что значение функции на данном наборе может быть как «0», так и «1»).

Для n логических переменных (аргументов) функции существует 2^n двоичных наборов. На каждом таком наборе может быть определено значение функции 0 или 1. Если значения функции отличаются хотя бы на одном наборе, то функции разные. Общее число логиче-

ских функций от n аргументов равно $M = (2^2)^n$. Для $n = 1$ $M = 4$, для $n = 2$ $M = 16$, для $n = 3$ $M = 256$. Далее число различных функций очень быстро растет. Практическое значение имеют четыре функции от одной логической переменной и 16 функций от двух переменных, так как любое сложное выражение можно рассматривать как композицию из простейших.

Словесное описание логической функции применимо только в случае сравнительно не сложной логической функции. Например, для функции y_3: значение функции равно нулю только тогда, когда первый и последний аргумент равен единице, а второй аргумент равен нулю.

Логическое выражение – это комбинация логических переменных и констант, связанных элементарными логическими операциями. Логическое выражение позволяет определить любую логическую функцию. Например, логическая функция, определенная в табл. 1.1, может быть записана следующим образом: $f(x_1, x_2, x_3) = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} + \overline{x_1} \cdot \overline{x_2} \cdot x_3 + x_1 \cdot \overline{x_2} \cdot x_3 + x_1 \cdot x_2 \cdot x_3$.

Набор элементарных логических операций, с помощью которого можно задать любую, сколь угодно сложную логическую функцию, называется *базисом*, или *функционально полной системой логических функций*.

Рассмотрим элементарные логические функции от одной или двух логических переменных. Все возможные функции одной переменной приведены в табл. 1.3.

Таблица 1.3

Логические функции от одной переменной

x	y_1	y_2	y_3	y_4
0	0	0	1	1
1	0	1	0	1

Функции $y_1 = 0$ и $y_4 = 1$ – константы «0» и «1», соответственно. Функция y_2 повторяет значение входной переменной x и записывается: $y_2 = x$. Функция y_3 равна инверсному значению входной переменной x и записывается как $y_3 = \overline{x}$. Это функция отрицания (или инверсии), читается как: «y равно не x». Данная функция в том или ином виде присутствует в каждом базисе.

В табл. 1.4 приведены все возможные функции от двух переменных.

Таблица 1.4

Логические функции от двух переменных

Входные переменные		**Функции**															
x_1	x_2	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}	y_{16}
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

В табл. 1.5 приведены основные сведения по функциям от двух переменных.

Рассмотрим основные операции булевой алгебры.

Логическое отрицание (функция НЕ). Логическим отрицанием высказывания x называется такое высказывание $f(x)$, которое истинно, когда x ложно, и наоборот. Функция НЕ записывается следующим образом: $f = \bar{x}$. Условное графическое изображение (УГО) элемента, реализующего функцию НЕ, приведено на рис. 1.1, *а*.

Логическое умножение (конъюнкция). Конъюнкция (функция И) двух переменных x_1 и x_2 − это сложное высказывание, которое истинно тогда и только тогда, когда истинны высказывания x_1 и x_2, и ложно в остальных случаях. Логическая функция конъюнкции имеет вид $f = x_1 \cdot x_2$. Для обозначения операции конъюнкции используются также символы & и Λ. Функция логического умножения (И) от n переменных имеет вид $f(x_1, x_2, \ldots, x_n) = x_1 \cdot x_2, \ldots, x_n$. Условное изображение элемента, реализующего операцию логического умножения, приведено на рис. 1.1, *б*.

Логическое сложение (дизъюнкция). Дизъюнкция (функция ИЛИ) двух переменных x_1 и x_2 − это сложное высказывание, которое истинно тогда, когда истинна хотя бы одна из переменных x_1 или x_2, и ложно, когда они обе ложны. Логическая функция дизъюнкции имеет вид $f = x_1 + x_2$. Для обозначения операции дизъюнкции используется также символ V. Функция логического сложения (ИЛИ) от n переменных имеет вид $f(x_1, x_2, \ldots, x_n) = x_1 + x_2 +, \ldots, + x_n$. Условное изображение элемента, реализующего операцию логического сложения, показано на рис. 1.1, *в*.

Таблица 1.5

Информация по функциям двух переменных

Функция	Название функции	Запись в виде логического выражения
y_1	Константа «0»	$y_1 = 0$
y_2	Конъюнкция (логическое умножение)	$y_2 = x_1 \cdot x_2$ ($y_2 = x_1 \,\&\, x_2$, или $y_2 = x_1 \wedge x_2$)
y_3	Запрет по x_2	$y_3 = x_1 \cdot \overline{x_2}$
y_4	Повторение x_1	$y_4 = x_1$
y_5	Запрет по x_1	$y_5 = \overline{x_1} \cdot x_2$
y_6	Повторение x_2	$y_6 = x_2$
y_7	Неравнозначность (Сумма по модулю два)	$y_7 = x_1 \cdot \overline{x_2} + \overline{x_1} \cdot x_2$ ($y_7 = x_1 \oplus x_2$)
y_8	Дизъюнкция (логическое сложение)	$y_8 = x_1 + x_2$ ($y_2 = x_1 \vee x_2$)
y_9	Стрелка Пирса	$y_9 = \overline{x_1 + x_2}$
y_{10}	Равнозначность ($y_{10} = x_1 \sim x_2$)	$y_{10} = \overline{x_1} \cdot \overline{x_2} + x_1 \cdot x_2$ ($y_{10} = \overline{x_1 \oplus x_2}$)
y_{11}	Инверсия x_2	$y_{11} = \overline{x_2}$
y_{12}	Импликация ($y_{12} = x_2 \rightarrow x_1$)	$y_{12} = \overline{x_1} \cdot \overline{x_2} + x_1$
y_{13}	Инверсия x_1	$y_{13} = \overline{x_1}$
y_{14}	Импликация ($y_{14} = x_1 \rightarrow x_2$)	$y_{12} = \overline{x_1} \cdot \overline{x_2} + x_2$
y_{15}	Штрих Шеффера ($y_{15} = x_1 \mid x_2$)	$y_{15} = \overline{x_1 \cdot x_2}$
y_{16}	Константа «1»	$y_1 = 1$

Сложение по модулю два (операция Исключающее ИЛИ). Сложение по модулю два − это сложное высказывание, которое истинно только тогда, когда истинна только одна из переменных x_1 или x_2. Логическая функция имеет вид $f = x_1 \oplus x_2$. Если число переменных больше двух, то функция истинна на тех наборах, в которых число единиц нечетно. Условное изображение элемента, реализующего операцию Сумма по модулю два, показано на рис. 1.1, *г*. (в англоязычной литературе – *XOR – Exclusively OR*).

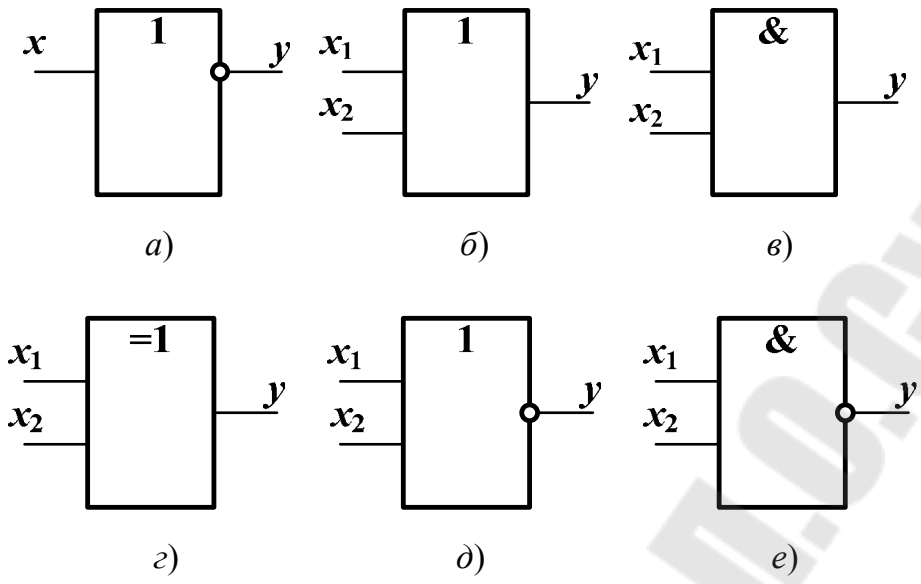


Рис. 1.1. Условные графические обозначения основных логических элементов

Отрицание дизъюнкции (стрелка Пирса). Отрицание дизъюнкции (функция ИЛИ-НЕ) двух переменных x_1 и x_2 – сложное высказывание, истинное только тогда, когда оба аргумента принимают ложное значение. Логическая функция ИЛИ-НЕ имеет вид $f = \overline{x_1 + x_2}$. Условное изображение элемента, реализующего указанную операцию, приведено на рис. 1.1, *д* и называется элементом ИЛИ-НЕ (в англоязычной литературе – *NAND – Not AND* (НЕ-И)).

Отрицание конъюнкции (штрих Шеффера). Отрицание конъюнкции (функция И-НЕ) двух переменных x_1 и x_2 – сложное высказывание, ложное только при истинности обоих аргументов x_1 и x_2. Логическая функция И-НЕ имеет вид $f = \overline{x_1 \cdot x_2}$. Условное изображение элемента, реализующего указанную операцию, приведено на рис. 1.1, *е* и называется элементом И-НЕ (в англоязычной литературе – *NOR – Not OR* (НЕ-ИЛИ)).

Импликация. Импликация двух переменных x_1 и x_2 – сложное высказывание, принимающее ложное значение только в случае, если x_1 истинно, а x_2 ложно.

На основании рассмотренных базовых операций можно строить новые булевы функции с помощью обобщенной операции, называемой операцией суперпозиции. Суперпозиция функций одного аргумента порождает функции одного аргумента. Суперпозиция функций двух аргументов дает возможность строить функции любого числа аргументов. Особенности суперпозиции булевых функций заключаются в следующем:

– одна функция может быть представлена разными формулами;

– каждой формуле соответствует схема соединений элементов;

– между формулами и схемами, их реализующими, существует взаимно однозначное соответствие.

Преобразование формул булевых функций основано на использовании соотношений булевой алгебры, рассматриваемых ниже.

1.3. Основные законы алгебры логики

Наиболее распространенной в алгебре логики является функционально полная система логических функций, которая в качестве базовых логических функций использует функцию одной переменной «НЕ» (функция отрицания) и две функции двух переменных «И» (конъюнкция, или логическое умножение) и «ИЛИ» (дизъюнкция, или логическое сложение). Эта система получила название система *Булевых* функций, или *Булевый базис*. В алгебре логики имеется целый раздел «*алгебра Буля*», посвященный этому базису. Законы алгебры логики позволяют проводить эквивалентные преобразования произвольных логических функций, записанных с помощью операций И, ИЛИ, НЕ, приводить их к удобному для дальнейшего использования виду и упрощать запись.

В алгебре Буля используется следующая приоритетность выполнения операций (в порядке убывания):

– отрицание и выражение в скобках;

– логическое умножение (операция И);

– логическое сложение и сумма по модулю два (операции ИЛИ и Исключающее ИЛИ).

При выполнении преобразований логических функций справедливы следующие соотношения:

1. *Операции с константами*. Основные операции с константами представлены в табл. 1.6.

Таблица 1.6

Операции с константами

Инверсия	Дизъюнкция	Конъюнкция	Сумма по модулю два
$\bar{0} = 1$	$0 + 0 = 0$	$0 \cdot 0 = 0$	$0 \oplus 0 = 0$
$\bar{1} = 0$	$0 + 1 = 1 + 0 = 1$	$0 \cdot 1 = 1 \cdot 0 = 0$	$0 \oplus 1 = 1 \oplus 0 = 1$
$x \oplus 1 = \bar{x}$	$1 + 1 = 1$	$1 \cdot 1 = 1$	$1 \oplus 1 = 0$

Отметим следующие особенности:

– выражение $x+1=1$ всегда истинно;

– выражение $x \cdot 0 = 0$ всегда ложно;

– выражение $x \oplus 1$ формирует инверсию переменной x.

2. *Операции с одинаковыми операндами.*

Сумма (произведение) одинаковых операндов равна этому операнду:

$$x + x + x + \ldots + x = x;$$

$$x \cdot x \cdot x \cdot \ldots \cdot x = x.$$

Правило справедливо для любого количества логических операндов.

3. *Операции с отрицаниями*:

– всегда истинно выражение: $x + \bar{x} = 1$;

– всегда ложно выражение: $x \cdot \bar{x} = 0$;

– правило двойного отрицания: $\bar{\bar{x}} = x$.

4. *Переместительный* (*коммутативный*) *закон*:

– для дизъюнкции: $x_1 + x_2 = x_2 + x_1$;

– для конъюнкции: $x_1 \cdot x_2 = x_2 \cdot x_1$;

– для суммы по модулю два: $x_1 \oplus x_2 = x_2 \oplus x_1$.

Этот закон справедлив для любого количества логических операндов.

5. *Сочетательный* (*ассоциативный*) *закон*:

– для дизъюнкции: $(x_1 + x_2) + x_3 = x_1 + (x_2 + x_3)$;

– для конъюнкции: $(x_1 \cdot x_2) \cdot x_3 = x_1 \cdot (x_2 \cdot x_3)$;

– для суммы по модулю два: $(x_1 \oplus x_2) \oplus x_3 = x_1 \oplus (x_2 \oplus x_3)$.

Этот закон справедлив для любого количества логических операндов.

6. *Распределительный* (*дистрибутивный*) *закон*:

– для дизъюнкции: $x_1 + x_2 \cdot x_3 = (x_1 + x_2) \cdot (x_1 + x_3)$;

– для конъюнкции: $x_1 \cdot (x_2 + x_3) = x_1 \cdot x_2 + x_1 \cdot x_3$.

7. *Правило де Моргана*. Отрицание дизъюнкции (конъюнкции) переменных равно конъюнкции (дизъюнкции) отрицаний этих переменных. Правило де Моргана справедливо для любого числа переменных:

$$\overline{x_1 + x_2} = \bar{x}_1 \cdot \bar{x}_2;$$

$$\overline{x_1 \cdot x_2} = \bar{x}_1 + \bar{x}_2;$$

$$\overline{x_1 + x_2 + \ldots + x_n} = \overline{x_1} \cdot \overline{x_2} \cdot \ldots \cdot \overline{x_n};$$

$$\overline{x_1 \cdot x_2 \cdot \ldots \cdot x_n} = \overline{x_1} + \overline{x_2} + \ldots + \overline{x_n}.$$

8. *Правило склеивания*:

$$x_1 \cdot \overline{x_2} + x_1 \cdot x_2 = x_1;$$

$$(x_1 + \overline{x_2}) \cdot (x_1 + x_2) = x_1.$$

Отметим, что рассмотренные законы и правила в основном совпадают с законами и правилами обычной алгебры. Однако есть и отличия. Например, в обычной алгебре нет законов, аналогичных распределительному закону для дизъюнкции и законам инверсии.

1.4. Формы представления логических функций

Одну и ту же логическую функцию можно представить различными логическими выражениями. Среди множества выражений, которыми представляется логическая функция, особое место занимают две формы:

– *дизъюнктивная нормальная форма* (ДНФ);

– *конъюнктивная нормальная форма* (КНФ).

Дизъюнктивная нормальная форма представляет собой дизъюнкцию элементарных конъюнкций, где под термином *элементарная конъюнкция* имеется в виду конъюнкция логических переменных или их отрицаний. Например, логические выражения $x_1\overline{x_2}x_3$, x_1x_3 являются элементарными конъюнкциями, а выражения вида $\overline{x_1\overline{x_2}x_3}$, $\overline{x_1\overline{x_3}}$ не являются элементарными конъюнкциями. Число переменных, входящих в элементарную конъюнкцию, определяет ранг этой конъюнкции. Примером ДНФ является функция $y = x_1\overline{x_2}x_3\overline{x_4} + x_1\overline{x_2}x_4 + x_3\overline{x_4}$.

Совершенной дизъюнктивной нормальной формой (СДНФ) логической функции от *n* аргументов называется такая ДНФ, в которой все конъюнкции имеют ранг *n*. СДНФ записывается на основании таблицы истинности по следующему правилу: для каждого набора переменных, на котором булева функция принимает единичное значение, записывается конъюнкция ранга *n* и все эти конъюнкции объединяются дизъюнктивно; переменная имеет знак инверсии, если на соответствующем наборе имеет нулевое значение. Например, запишем в СДНФ функцию, представленную в табл. 1.1:

$$f_{\text{СДНФ}} = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} + \overline{x_1} \cdot \overline{x_2} \cdot x_3 + x_1 \cdot \overline{x_2} \cdot x_3 + x_1 \cdot x_2 \cdot x_3 . \qquad (1.1)$$

Элементарные конъюнкции, образующие СДНФ, называют также *конституентами* (составляющими) *единицы*, так как они соответствуют наборам, при которых функция принимает значение, равное единице. Построение СДНФ по таблице истинности называют составлением булевой функции по условиям истинности.

В общем виде запись СДНФ функции *y* имеет вид:

$$y = \sum_{j \in J} x_1 x_2 \ldots x_n ,$$

где *n* – число аргументов функции, *J* – множество наборов, на которых функция принимает единичное значение.

Конъюнктивная нормальная форма (КНФ) представляет собой конъюнкцию элементарных дизъюнкций, где под термином *элементарная дизъюнкция* имеется в виду сумма логических переменных или их отрицаний. Примером логических выражений, являющихся элементарными дизъюнкциями, могут служить $x_1 + x_2 + x_3$, $x_1 + \overline{x_3}$, а выражения вида $x_1 + \overline{\overline{x_2} + x_3}$, $\overline{x_1 + \overline{x_3}}$ не являются элементарными дизъюнкциями. Число переменных, входящих в элементарную дизъюнкцию, определяет ранг этой конъюнкции. Примером ДНФ является функция $y = (x_1 + \overline{x_2} + x_3 + \overline{x_4}) \cdot (x_1 + \overline{x_2} + x_4) \cdot (x_3 + \overline{x_4})$.

Совершенной конъюнктивной нормальной формой (СКНФ) логической функции от *n* аргументов называется такая КНФ, в которой все конъюнкции имеют ранг *n*. СКНФ записывается на основании таблицы истинности по следующему правилу: для каждого набора переменных, на котором булева функция принимает нулевое значение, записывается дизъюнкция ранга *n*, и все эти дизъюнкции объединяются конъюнктивно; переменная имеет знак инверсии, если на соответствующем наборе имеет единичное значение. Например, запишем в СКНФ функцию, представленную в табл.1.1:

$$f_{\text{СКНФ}} = (x_1 + \overline{x_2} + x_3) \cdot (x_1 + \overline{x_2} + \overline{x_3}) \cdot (\overline{x_1} + x_2 + x_3) \cdot (\overline{x_1} + \overline{x_2} + x_3) . \qquad (1.2)$$

Элементарные дизъюнкции, образующие СКНФ, называют *конституентами* (составляющими) *нуля*, так как они соответствуют наборам, при которых функция принимает нулевое значение.

В общем виде СКНФ функции *y* имеет вид:

$$y = \prod_{j \in J} (x_1 + x_2 + \ldots + x_n),$$

где n – число аргументов функции; J – множество наборов, на которых функция принимает нулевое значение.

Элементарные дизъюнкции, образующие СКНФ, называют конституентами нуля, так как они соответствуют наборам, при которых функция принимает нулевое значение.

Наряду с нормальными формами представления функций алгебры логики в вычислительной технике широко используются логические полиномиальные формы. Преобразования над формулами булевых функций иногда удобно выполнять в алгебре Жегалкина. Алгебра Жегалкина включает две двухместные операции: конъюнкцию и сложение по модулю два, а также константу единицу.

Теорема Жегалкина. Любая функция алгебры логики может быть представлена многочленом вида

$$f(x_1, x_2, \ldots, x_n) = k_0 \oplus k_1 x_1 \oplus k_2 x_2 \oplus \ldots \oplus k_n x_n \oplus k_{n+1} x_1 x_2 \oplus k_{n+2} x_1 x_3 \oplus \ldots \oplus k_{n+m} x_1 x_2 \ldots x_m,$$

где $k_i \in \{0, 1\}$.

Теорема позволяет представить любую логическую функцию в виде полиномов. Задача построения полинома Жегалкина сводится к нахождению коэффициентов k_i. Для любых конституент единицы k_1 и k_2 имеет место следующее соотношение – $k_1 + k2 = k1 \oplus k_2$, которое позволяет выполнить переход от СДНФ к полиному Жегалкина. Для этого достаточно заменить в СДНФ символ + (дизъюнкции) на символ $\oplus$ и выполнить подстановку вида $\bar{x} = x \oplus 1$ с последующими преобразованиями в алгебре Жегалкина.

1.5. Синтез логических схем по логическим выражениям

Логические схемы строятся на основе логических элементов, набор которых определяется заданным логическим базисом. Для базиса Буля в качестве логических элементов используются элементы, реализующие базовые логические функции *И*, *ИЛИ*, *НЕ*, которые приведены на рис. 1.1, *а–в* соответственно.

При синтезе схемы по логическому выражению логические операции представляются в виде соответствующих логических элементов, связи между которыми определяются последовательностью выполнения логических операций в заданном выражении. В качестве примера рассмотрим синтез логической схемы, определяемой выражением (1.1), в базисе И, ИЛИ, НЕ. Схема имеет три входных сигна-

ла – x_1, x_2, x_3, представленные в прямой и инверсной форме, и один выходной – $f_{СДНФ}$. Реализацию заданного выражения в виде логической схемы можно начать или с первой, или с последней операции. Первой операцией является инверсия входных переменных (используются три элемента НЕ). Следующая операция – формирование конъюнкций, для чего используются четыре трехвходовых элемента И. Последней операцией в заданном выражении является операция логического сложения четырех операндов (используется один четырехвходовый элемент ИЛИ). Схема синтезируемого выражения приведена на рис. 1.2.

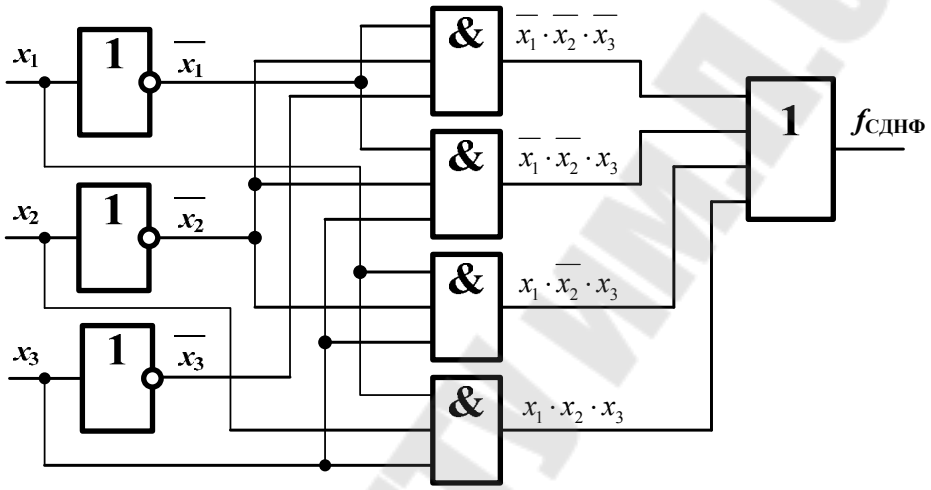


Рис. 1.2. Пример синтеза логической схемы, определяемой выражением (1.1)

Очевидно, среди схем, реализующих данную функцию, есть наиболее простая. Поиск логической формулы, соответствующей этой схеме, представляет большой практический интерес.

1.6. Минимизация логических функций

1.6.1. Параметры минимизации

Сложность логической схемы можно оценить по количеству требуемого оборудования, подсчитав суммарное количество всех элементарных компонентов (например, транзисторов), входящих в эту схему. Для КМОП-технологии: элемент НЕ состоит из двух транзисторов, элемент 2И-НЕ (2ИЛИ-НЕ) из четырех транзисторов, элемент 3И-НЕ (3ИЛИ-НЕ) из шести транзисторов и т. д. Часто сложность схемы оценивают по числу логических элементов, используемых для реализации схемы. Например, схема на рис. 1.2 состоит из восьми элементов (три инвертора, четыре элемента И и один элемент ИЛИ).

Быстродействие комбинационной схемы определяется максимальным временем задержки распространения сигнала от любого из входов до любого выхода. Следовательно, оценить быстродействие логической схемы можно рассчитав максимальное значение суммы времен задержек всех элементов, расположенных между каждым входом и выходом схемы. Например, будем считать, что время задержки всех логических элементов в схеме на рис. 1.2 одинаково и равно t. Тогда максимальное время задержки распространения сигнала равно $3t$.

Следующий параметр, подлежащий минимизации – энергопотребление, которое зависит как от параметров входных сигналов (в частности, от частоты входных сигналов), так и от схемной реализации логической функции. Причем минимизация одного параметра, как правило, конфликтует с другими параметрами. Например, минимизация энергопотребления приводит к снижению быстродействия. Таким образом, при проектировании логических схем стоит задача многокритериальной оптимизации при заданных ограничениях. Получить оптимальное решение по всем параметрам практически невозможно. Поэтому решается задача оптимизации по одному параметру (например, аппаратные затраты). Такая частная оптимизация называется минимизацией.

Далее, рассмотрим только одну оптимизационную задачу – минимизацию логической функции, обеспечивающую минимум аппаратных затрат. Минимизировать функции, т. е. находить наиболее простое выражение для исходной функции, можно различными методами. Рассмотрим наиболее известные из них:

1) расчетный (метод непосредственных преобразований);

2) расчетно-табличный (метод Квайна–Мак-Класки);

3) табличный (метод Вейча–Карно).

В общем случае минимизация проводится в три этапа:

1. Переход от СДНФ (или СКНФ) к сокращенной ДНФ (КНФ) путем производства всех возможных склеиваний друг с другом – сначала конституент, потом всех производных членов более низкого ранга (импликант).

2. Переход от сокращенной к тупиковой нормальной форме (ТДНФ, ТКНФ). Тупиковой называют такую нормальную форму, членами которой являются простые импликанты. Причем среди этих импликант нет ни одной лишней (такой импликанты, удаление которой не влияет на значение истинности этой функции).

3. Переход от тупиковой формы функции к ее минимальной форме (факторизация). Этот этап не является регулярным (формальным), поэтому требует некоторого перебора решений.

Различные методы минимизации отличаются друг от друга путями и средствами реализации того или иного этапа.

1.6.2. Расчетный метод минимизации

При расчетном методе минимизации все этапы выполняются аналитически. Например, пусть задана некоторая функция в СДНФ, которую необходимо минимизировать:

$$f_{\text{СДНФ}} = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} + \overline{x_1} \cdot \overline{x_2} \cdot x_3 + x_1 \cdot \overline{x_2} \cdot x_3 + x_1 \cdot x_2 \cdot x_3 .$$

1. Переход от СДНФ к сокращенной ДНФ, применяя склеивание. Первое и второе слагаемое склеиваем по x_3, третье и четвертое слагаемое склеиваем по x_2. Получим:

$$f_{\text{СДНФ}} = \overline{x_1} \cdot \overline{x_2} + x_1 \cdot x_3 .$$

2. Переход к тупиковой форме путем проверки каждой импликанты. Первое слагаемое принимает единичное значение на наборах 000 и 001, причем на этих наборах второе слагаемое равно нулю. По аналогии второе слагаемое принимает единичное значение на наборах 101 и 111, причем на этих наборах первое слагаемое равно нулю. Следовательно, в выражении нет лишних импликант и оно является тупиковой формой:

$$f_{\text{ТДНФ}} = \overline{x_1} \cdot \overline{x_2} + x_1 \cdot x_3 .$$

3. Переход от тупиковой формы функции к ее минимальной форме.

Так как на предыдущем этапе была получена только одна тупиковая форма, то она является минимальной:

$$f_{\text{МДНФ}} = \overline{x_1} \cdot \overline{x_2} + x_1 \cdot x_3 .$$

1.6.3. Расчетно-табличный метод минимизации

Минимизация этим методом основана на следующей теореме.

Теорема Квайна. Для получения минимальной формы логической функции необходимо в СДНФ произвести все возможные склеивания и поглощения так, чтобы в результате была получена сокра-

щенная ДНФ. Сокращенная ДНФ в общем случае может содержать лишние простые импликанты, которые необходимо выявить на втором этапе минимизации.

На первом этапе выполняется переход от функции, заданной в форме СДНФ, к сокращенной ДНФ, используя операции склеивания и поглощения.

Склеивание и поглощение выполняются до тех пор, пока имеются члены, не участвовавшие в попарном сравнении. Термы, подвергшиеся операции склеивания, отмечаются. Неотмеченные термы представляют собой простые импликанты и включаются в сокращенную ДНФ. Все отмеченные конъюнкции ранга $n-1$ подвергаются вновь операции склеивания до получения термов $n-2$ ранга и так далее до тех пор, пока количество неотмеченных конъюнкций больше 2. В результате выполнения первого этапа получена сокращенная ДНФ. Полученное логическое выражение не всегда оказывается минимальным.

На втором этапе переходят от сокращенной ДНФ к тупиковым ДНФ и на третьем этапе выбирают минимальную ДНФ.

Для формирования тупиковых ДНФ строится *импликантная таблица* (*матрица*), строки которой отмечаются простыми импликантами сокращенной ДНФ, а столбцы – конституентами единицы исходной СДНФ. В строке напротив каждой простой импликанты ставится метка под теми наборами (конституентами единицы), на которых она принимает значение 1. Соответствующие конституенты поглощаются (покрываются) данной простой импликантой.

Из общего числа простых импликант необходимо отобрать их минимальное число, исключив лишние. Формирование тупиковых форм и выбор минимального покрытия начинается с выявления обязательных простых импликант, т. е. таких, которые (и только они) покрывают некоторый исходный набор.

Рассмотрим пример минимизации методом Квайна логической функции:

$$f_{\text{СДНФ}} = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} + \overline{x_1} \cdot \overline{x_2} \cdot x_3 + x_1 \cdot \overline{x_2} \cdot x_3 + x_1 \cdot x_2 \cdot x_3 .$$

Выполним операцию склеивания:

$1 - 2\ (x_3) \quad \rightarrow \quad \overline{x_1} \cdot \overline{x_2}$

$2 - 3\ (x_1) \quad \rightarrow \quad \overline{x_2} \cdot x_3$

$3 - 4\ (x_2) \quad \rightarrow \quad x_1 \cdot x_3$

Нахождение ТДНФ (ТКНФ) производится при помощи специальной таблицы, предложенной У. Квайном. В графах (колонках) указываются конституенты, в строках – импликанты сокращенной формы. Исходная форма минимизируемой функции – СДНФ (или СКНФ).

Таблица 1.7

Таблица Квайна

Импликанты	**Конституенты**			
	$\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3}$	$\overline{x_1} \cdot \overline{x_2} \cdot x_3$	$x_1 \cdot \overline{x_2} \cdot x_3$	$x_1 \cdot x_2 \cdot x_3$
$\overline{x_1} \cdot \overline{x_2}$	+	+		
$\overline{x_2} \cdot x_3$		+	+	
$x_1 \cdot x_3$			+	+

Процесс минимизации осуществляется путем последовательного сопоставления каждой импликанты со всеми конституентами. Если импликанта является частью конституенты, то в таблице в соответствующей клетке ставится знак «+». Каждая конституента может покрываться несколькими импликантами. Задача состоит в том, чтобы вычеркиванием некоторых (лишних) импликант попытаться оставить в каждой колонке только один знак «+» (по крайней мере – минимальное количество). Из табл. 1.7 – лишней является импликанта $\overline{x_2} \cdot x_3$ и тупиковая форма будет иметь вид: $f_{\text{ТДНФ}} = \overline{x_1} \cdot \overline{x_2} + x_1 \cdot x_3$.

Для использования данного метода на ЭВМ Мак-Класки предложил числовое представление логических функций. Десятичный индекс j у символа реализации функции f_j, развернутой в n-разрядное двоичное число, позволяет полностью восстановить вид набора аргументов $[x_i]_j$, который соответствует данной реализации. А вид набора значений аргументов позволяет восстановить вид соответствующей ему конституенты единицы или нуля.

Например, относительно громоздкую буквенную запись функции $f_{\text{СДНФ}} = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} + \overline{x_1} \cdot \overline{x_2} \cdot x_3 + x_1 \cdot \overline{x_2} \cdot x_3 + x_1 \cdot x_2 \cdot x_3$ можно заменить более компактной числовой записью:

$$f_{\text{СДНФ}}(x_1, x_2, x_3) = 000_2 \vee 001_2 \vee 101_2 \vee 111_2 =$$
$$= 0_{10} \vee 1_{10} \vee 5_{10} \vee 7_{10} = \vee(0, 1, 5, 7).$$

При таком представлении логических функций их можно вводить в ЭВМ, обрабатывать в соответствии с правилами булевой ал-

гебры. Отметим, что любая переменная x_i в этом случае будет иметь не два, а три состояния: 0, 1 и отсутствие переменной.

1.6.4. Табличный метод минимизации

При минимизации логической функции от небольшого числа переменных удобным является графический метод представления функции с помощью диаграмм (карт) Вейча и их разновидности – Карно. Карта Вейча представляет собой развертку n-мерного куба на плоскости. При этом вершины куба представляются клетками карты, каждой из которых поставлена в соответствие конституента единицы или нуля. Переменные, обозначающие клетки диаграммы, расставляются таким образом, чтобы наборы, записанные в двух смежных клетках, отличались только одним разрядом. Поскольку такие наборы располагаются в смежных клетках, они получили название соседних наборов. В клетку карты, соответствующую конституенте единицы, заносится 1, иначе − 0.

Таким образом, для минимизации функции она должна быть представлена в форме СДНФ. Минимизация булевой функции с использованием карт в дизъюнктивной (конъюнктивной) форме заключается в объединении единичных (нулевых) клеток в контуры, каждому такому контуру соответствует простая импликанта.

Можно сформулировать следующие правила минимизации:

– число клеток карты в одном контуре должно быть равно 2^n;

– для контура, содержащего 2^n клеток, должно быть n осей симметрии;

– количество контуров должно быть минимально;

– число единиц в контуре должно быть максимально;

– контуры могут пересекаться, т. е. некоторая клетка может входить в несколько контуров.

В принципе диаграмма Вейча–Карно является разновидностью табличной записи некоторой функции, заданной в СДНФ или СКНФ. Она имеет в общем виде вид прямоугольника, разбитого на 2^n клеток. При нечетном n одну сторону образуют $2^{(n-1)/2}$ клеток, а другую – $2^{(n+1)/2}$. При четном n каждая сторона прямоугольника образуется из $2^{n/2}$ клеток. Каждая клетка соответствует одному определенному набору аргументов и, следовательно, одной определенной конституенте.

Для того чтобы при помощи таблиц Вейча–Карно произвести минимизацию, необходимо выполнить одно важное условие: в соседних клетках (в физическом смысле) должны находиться соседние конституенты. Это условие называется условием совмещенного соседства.

Для нахождения клетки, соответствующей определенному набору аргументов, в качестве координат клетки используются значения аргументов. Строится плоская прямоугольная система координат, ось ординат которой совпадает с левым вертикальным краем диаграммы, а ось абсцисс – с нижним краем. Пронумеруем ряды и столбцы клеток на диаграмме двоичными числами, причем каждый двоичный разряд сопоставим с определенной логической переменной. Тогда ось ординат явится осью изменения сложного аргумента, состоящего из $n/2$ первых логических переменных, а ось абсцисс – $n/2$ последних логических переменных. Если n нечетное, то по оси ординат $(n-1)/2$ – логических переменных, а по оси абсцисс – $(n+1)/2$ переменных. Тогда каждый сложный аргумент изменяется дискретно вдоль соответствующей координатной оси, пробегая все числовые значения от 0 = = 000…0 до $2^{n/2} - 1 = 111\ldots1$.

Следовательно, в построенной системе координат можно положение любой клетки на диаграмме точно и однозначно определить парой полуслов с разрядностью $n/2$ (или $(n-1)/2$ и $(n+1)/2$). Каждое слово, состоящее из двух координатных полуслов, будет соответствовать определенной конституэнте, так как всегда можно сделать обратный переход от цифр 0 и 1 к символам x_i и $\overline{x_i}$ соответственно ($i = 1, 2, \ldots, n$).

Таким образом, имея на диаграмме по осям ординат и абсцисс номера – координаты клеток в двоичной системе счисления, не заполняя самих клеток, всегда можно быстро и легко указать, какой конституэнте соответствует та или иная клетка и наоборот. Однако если эти координаты будут проставляться в обычном двоичном коде, то условие совмещенного соседства будет выполняться не всегда. Для выполнения этого условия координаты следует проставлять в двоичном циклическом *коде Грея*. Этот код имеет такую особенность, что представленные в нем изображения двух чисел, имеющих различие по величине на единицу младшего разряда (или просто на 1 для целых чисел), отличаются друг от друга только в одном разряде. Пример представления чисел в двоичном коде и коде Грея представлен в табл. 1.8.

Анализ таблицы позволяет сформулировать правило преобразования любого n-разрядного двоичного числа ($A = a_{n-1}, a_{n-2}, \ldots, a_0$) в n-разрядный код Грея ($B = b_{n-1}, b_{n-2}, \ldots, b_0$):

– самая старшая цифра в коде Грея совпадает с самой старшей значащей цифрой этого же числа в двоичном коде, т. е. $b_{n-1} = a_{n-1}$;

– остальные цифры: $b_i = a_i \oplus a_{i+1},\ i = \overline{0, n-2}$.

Таблица 1.8

Код Грея

Число			Число		
Десятичное	**Двоичное**	**Грея**	**Десятичное**	**Двоичное**	**Грея**
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

Например, переведем двоичное число 110101 в код Грея. Здесь $n = 6$, $b_5 = a_5 = 1$, $b_4 = a_4 \oplus a_5 = 1 \oplus 1 = 0$, $b_3 = a_3 \oplus a_4 = 0 \oplus 1 = 1$, $b_2 = a_2 \oplus a_3 = 1 \oplus 0 = 1$, $b_1 = a_1 \oplus a_2 = 0 \oplus 1 = 1$, $b_0 = a_0 \oplus a_1 = 1 \oplus 0 = 0$. Получили: $B = 101110$.

Применение кода Грея для нумерации клеток по осям ординат и абсцисс в диаграмме Вейча–Карно обеспечивает автоматическое размещение в соседних клетках соседних членов СДНФ (или СКНФ) любой логической функции. В самих клетках необходимо только проставить значение истинности реализации функции на каждом из набора аргументов. Причем в случае задания функции в СДНФ – достаточно проставить только «1», а в случае СКНФ – только «0».

Процесс минимизации с помощью диаграмм Вейча–Карно подчиняется следующим правилам:

Правило 1. 2^i смежных клеток, расположенных в виде прямоугольника, соответствуют одной элементарной конъюнкции (дизъюнкции), ранг которой r меньше ранга конституенты n на i единиц.

Правило 2. В любой диаграмме соседними клетками являются не только смежные клетки, но и клетки, расположенные на противоположных концах любой строки и любого столбца.

Импликанта, соответствующая некоторой группе заполненных клеток, будет содержать в себе символы тех переменных, значения которых совпадают у всех объединенных клеток. Пользуясь переходом $0 \to \overline{x_i}$, $1 \to x_i$ для функций, представленных в СДНФ, или $1 \to \overline{x_i}$, $0 \to x_i$ для СКНФ, можем записать выражения для каждого члена функции или импликанты.

Построим карту Карно для $f_{\text{СДНФ}} = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} + \overline{x_1} \cdot \overline{x_2} \cdot x_3 + x_1 \cdot \overline{x_2} \cdot x_3 + x_1 \cdot x_2 \cdot x_3$ (рис. 1.3). Имеем две группы заполненных клеток, которые покрывают все единицы. Получили $f_{\text{ТДНФ}} = \overline{x_1} \cdot \overline{x_2} + x_1 \cdot x_3$.

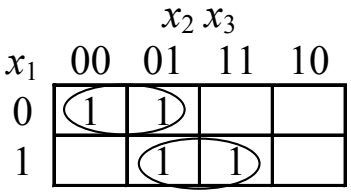


Рис. 1.3. Карта Карно

Рассмотрим пример минимизации произвольной логической функции *f* (табл. 1.9) при помощи карт Карно.

Таблица 1.9

Таблица истинности функции *f*

$x_1x_2x_3x_4x_5$	*f*	$x_1x_2x_3x_4x_5$	*f*	$x_1x_2x_3x_4x_5$	*f*	$x_1x_2x_3x_4x_5$	*f*
00000	1	01000	0	10000	1	11000	0
00001	0	01001	0	10001	1	11001	0
00010	1	01010	0	10010	0	11010	1
00011	0	01011	1	10011	0	11011	0
00100	1	01100	1	10100	1	11100	0
00101	0	01101	1	10101	1	11101	0
00110	0	01110	0	10110	1	11110	1
00111	1	01111	1	10111	1	11111	1

В данном случае число переменных $n = 5$, поэтому карта Карно будет состоять из $2^5 = 32$ клеток. Представим эти клетки в виде матрицы 8 x 4 (рис. 1.4).

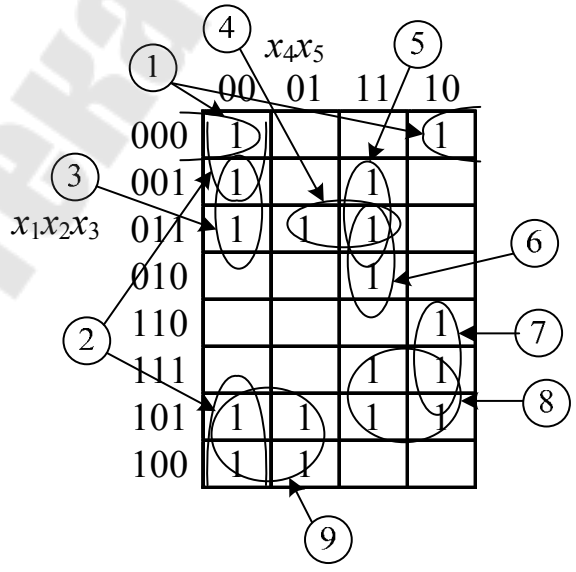


Рис. 1.4. Карта Карно для функции *f*

Запишем тупиковую ДНФ исходной функции на основании карты Карно (цифрами проставлены номера простых импликант):

$$f_{\text{ТДНФ}} = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} \cdot \overline{x_5} + \overline{x_2} \cdot \overline{x_4} \cdot \overline{x_5} + \overline{x_1} \cdot x_3 \cdot \overline{x_4} \cdot \overline{x_5} + \overline{x_1} \cdot x_2 \cdot x_3 \cdot x_5 +$$

$$+ \overline{x_1} \cdot x_3 \cdot x_4 \cdot x_5 + \overline{x_1} \cdot x_2 \cdot x_4 \cdot x_5 + x_1 \cdot x_2 \cdot x_4 \cdot \overline{x_5} + x_1 \cdot x_3 \cdot x_4 + x_1 \cdot \overline{x_2} \cdot \overline{x_4}.$$

Данный метод минимизации применим при относительно небольшом числе аргументов логической функции ($n < 8$).

2. ФУНКЦИОНАЛЬНЫЕ УЗЛЫ ЭВМ

2.1. Классификация функциональных узлов

Функциональные узлы предназначены для выполнения некоторых типовых операций в ЭВМ, таких, как хранение и сдвиг данных, арифметические операции, преобразование двоичных кодов и т. п. Это позволяет представить ЭВМ в виде совокупности модулей, выполняющих определенные функции. Упрощенная классификация функциональных узлов представлена на рис. 2.1. Все узлы делятся на три категории: комбинационные схемы, последовательностные схемы и вспомогательные схемы. На данном рисунке не показана память ЭВМ и процессорные элементы, так как они будут рассмотрены в последующих темах.

Комбинационные схемы содержат только комбинационную логику, запоминающие элементы в этих схемах отсутствуют. Поэтому значения выходных сигналов у них зависят только от текущего значения входных сигналов и не зависят от значений входных сигналов в предыдущие моменты времени. Последовательные схемы содержат как запоминающие элементы, так и комбинационную логику. Поэтому значения выходных сигналов у них зависят как от текущего значения входных сигналов, так и от значений входных сигналов в предыдущие моменты времени. Вспомогательные схемы предназначены для выполнения вспомогательных функций. Среди этих функций отметим усиление или умощнение сигналов, преобразование логических уровней. К ним также относятся схемы управления нецифровыми устройствами (например, светодиодами, двигателями и т. п.). Кроме того, к ним относятся разнообразные интерфейсные схемы, предназначенные для реализации внутренних и внешних интерфейсов ЭВМ. Часто эти схемы комбинируют со схемами, выполняющими некоторую логическую функцию.

Рассмотрим правила построения условных графических обозначений (УГО) функциональных элементов вычислительной техники, определяемых ГОСТ 2.743–82. К элементам вычислительной техники относят элементы, предназначенные для преобразования и обработки сигналов, представленных в двоичном виде. Кроме того, сюда относят и элементы, не выполняющие логическую функцию, но применяемые в логических цепях, например, резистор, конденсатор, генератор, транзистор и др.

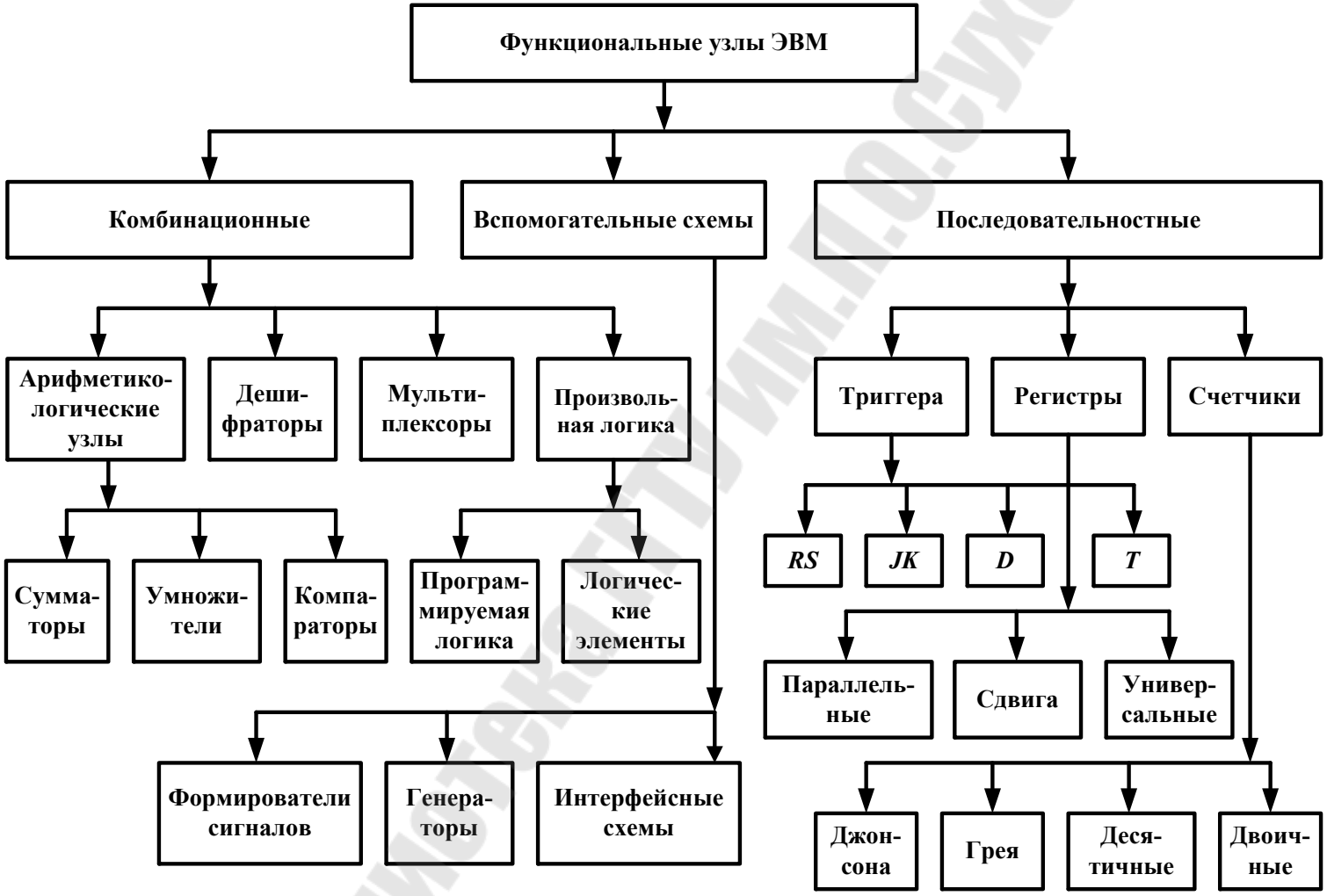


Рис. 2.1. Классификация функциональных узлов ЭВМ

Условное графическое обозначение элементов строят на основе прямоугольника. В самом общем виде УГО может содержать основное и два дополнительных поля, расположенных по обе стороны от основного (рис. 2.2). Ширина основного поля должна быть не менее 10 мм, дополнительных – не менее 5 мм (при большом числе знаков в обозначениях функции и входов-выходов элемента эти размеры соответственно увеличивают). Размер прямоугольника по ширине зависит от наличия дополнительных полей и количества помещенных в них знаков, по высоте – от числа выводов, интервалов между ними и числа строк информации в основном и дополнительных полях.

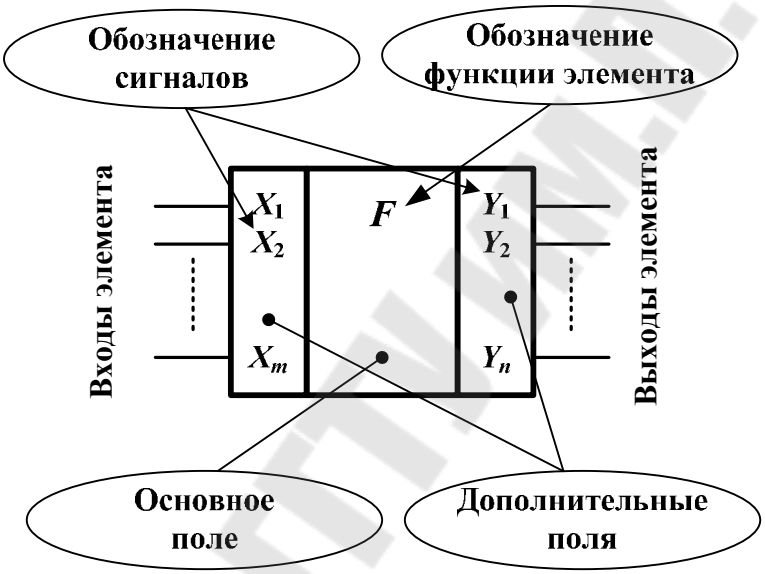


Рис. 2.2. Условное графическое обозначение элементов ЭВМ

Согласно стандарту, расстояние между выводами – 5 мм, между выводом и горизонтальной стороной обозначения (или границей зоны) – не менее 2,5 мм и кратно этой величине. При разделении групп выводов интервалом величина последнего должна быть не менее 10 мм и кратна 5 мм. Выводы элементов делятся на входы, выходы, двунаправленные выводы и выводы, не несущие информации. Входы изображают слева, выходы – справа, остальные выводы – с любой стороны УГО. При необходимости разрешается поворачивать УГО на 90° по часовой стрелке, т. е. располагать входы сверху, а выходы – снизу.

Функциональное назначение элемента указывают в верхней части основного поля УГО (см. рис. 2.2). Его составляют из прописных букв латинского алфавита, арабских цифр и специальных знаков, записываемых без пробелов (число знаков в обозначении функции не

ограничивается). Обозначения основных функций и их производных приведены в табл. 2.1. В последующих строках – соответствующую информацию по ГОСТ 2.708–81.

Таблица 2.1

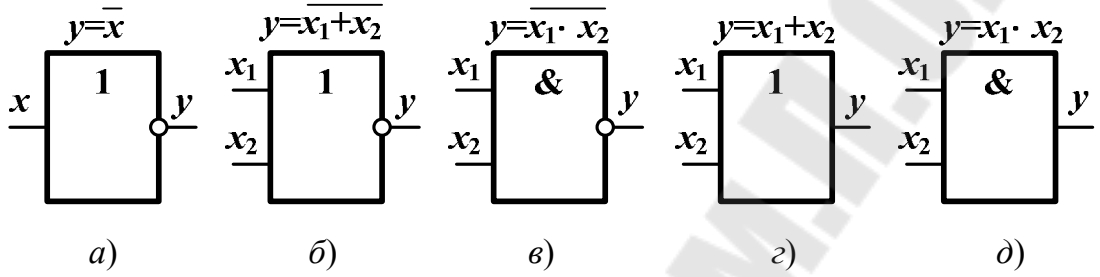
Обозначения основных функций элементов ЭВМ

Номер	Наименование	Обозначение
1	Логический элемент	
	НЕ	1
	И	&
	ИЛИ	1
	Исключающее ИЛИ	=1
2	Вычислитель:	*CP*
	секция вычислителя	*CPS*
	вычислительное устройство	*CPU*
	с плавающей точкой	*FPU*
	арифметико-логическое устройство	*ALU*
	микропроцессор	*MP*, *MPU*
3	Дешифратор	*DC*
4	Демультиплексор	*DX*
5	Мультиплексор	*MX*
6	Компаратор	*CMP*
7	Регистр	*RG*
8	Сумматор	*SM*
9	Счетчик двоичный	*ST*2
	десятичный	*ST*10
10	Триггер одноступенчатый	*T*
	Триггер двухступенчатый	*TT*
11	Память:	
	постоянное запоминающее устройство (ПЗУ)	*ROM*
	программируемое пользователем постоянное запоминающее устройство (ППЗУ)	*PROM* *EPROM*
	перепрограммируемое постоянное запоминающее устройство (в том числе *Flash*)	*EEPROM*
	оперативное запоминающее устройство (ОЗУ) с произвольной выборкой	*RAM*
	статическое ОЗУ с произвольной выборкой	*SRAM*
	динамическое ОЗУ с произвольной выборкой	*DRAM*
	ассоциативное запоминающее устройство	*CAM*

2.2. Комбинационные узлы ЭВМ

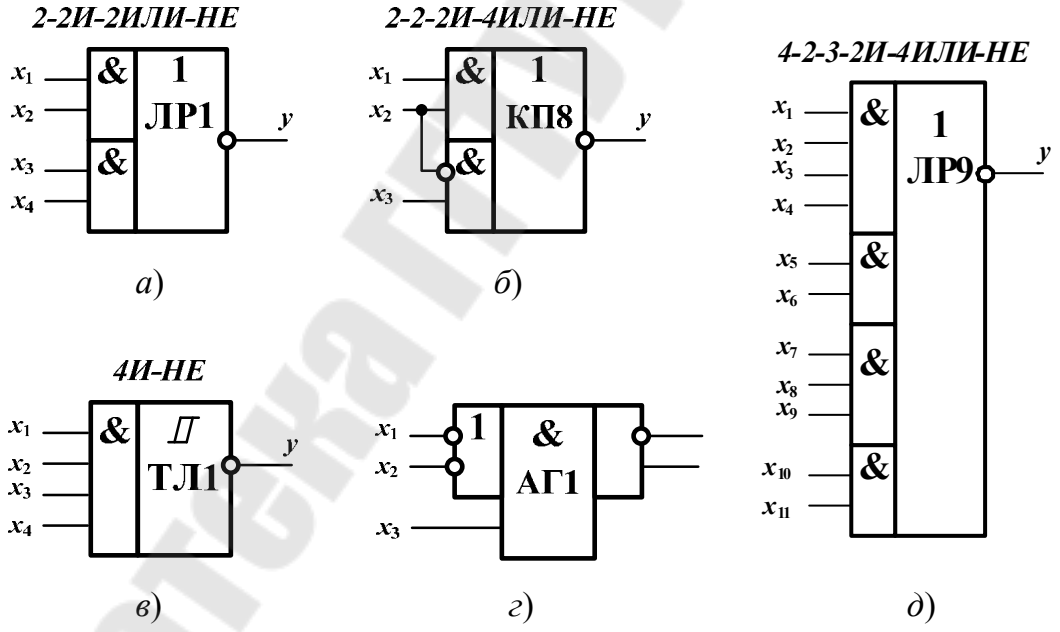
2.2.1. Произвольная логика

Произвольная логика содержит элементы, реализующие некоторые наперед заданные логические функции. Промышленностью выпускается достаточно разнообразный набор логических элементов, включающий в себя как простейшие, так и достаточно сложные элементы. Примеры этих элементов приведены на рис. 2.3 и 2.4.



а) *б*) *в*) *г*) *д*)

Рис. 2.3. Простейшие логические элементы ЭВМ



а) *б*) *в*) *г*) *д*)

Рис. 2.4. Примеры сложных логических элементов ЭВМ

Простейшие логические элементы входят в состав практически всех серий микросхем, выпускаемых промышленностью. Приведем примеры только для наиболее распространенных серий микросхем – ТТЛ (транзисторно-транзисторная логика) К155, ТТЛШ (транзисторно-транзисторная логика с диодами Шотки) КР1533 и КМОП (Комплиментарная МОП-технология) КР1561. Микросхема КР1533ЛН1

или КР1561ЛН2 – шесть инверторов, КР1533ЛЕ1 или КР1561ЛЕ5 – четыре элемента 2ИЛИ-НЕ, КР1533ЛА3 или КР1561ЛА7 – четыре элемента 2И-НЕ, КР1533ЛЛ1 – четыре элемента 2ИЛИ, КР1533ЛИ1 – четыре элемента 2И.

Кроме простейших логических элементов, в состав серий микросхем входят различные элементы, которые позволяют упростить реализацию произвольной логической функции или выполняют некоторую вспомогательную функцию. Например, микросхема К155ЛР1 содержит два элемента, реализующих функцию 2-2И-2ИЛИ-НЕ (рис. 2.4, *а*), на основе которых легко реализовать мультиплексор «2 в 1». В частности, микросхема К134КП8 содержит три таких мультиплексора (рис. 2.4, *в*). Микросхема КР531ЛР9 содержит один элемент 4-2-3-2И-4ИЛИ-НЕ (рис. 2.4, *д*), на основе которого легко реализовать достаточно сложную логическую функцию.

Микросхема КР1533ТЛ1 (рис. 2.4, *б*) содержит два логических элемента 4И-НЕ, кроме того, эти элементы выполняют функцию триггера Шмита (позволяют формировать более крутые фронты сигналов). Микросхема К155АГ1 (рис. 2.4, *г*) содержит одновибратор с логическим элементом на входе и предназначена для построения времязадающих элементов.

Успехи в области интегральной технологии привели к созданию больших интегральных схем БИС с тысячами логических элементов на одном кристалле, что позволило создавать компактные и надежные средства вычислительной техники.

Однако при этом возникла проблема: как на схемах с таким количеством элементов изготовлять устройства, реализующие различные функции или решающие разные задачи. Ведь спроектированная БИС решает одну конкретную задачу. Процесс проектирования новой БИС достаточно трудоемок и дорог, поэтому экономически оправдан лишь при массовом производстве. Возникшее противоречие нашло разрешение на путях разработки БИС/СБИС с программируемой и репрограммируемой структурой.

Первыми представителями указанного направления явились программируемые логические матрицы (ПЛМ), в англоязычной литературе применяют термин *Programmable Logic Array* (*PLA*), *Programmable Array Logic* (*PAL*) – программируемая матричная логика и базовые матричные кристаллы (БМК), называемые также вентильными матрицами, или *Gate Array* (*GA*). *PLA* и *PAL* в английской терминологии объединяются также термином *Programmable Logic Devices* (*PLD*). Развитие БИС/СБИС с программируемой и репрограммируе-

мой структурой оказалось настолько перспективным направлением, что привело к созданию новых эффективных средств разработки цифровых систем, таких, как *CPLD* (*Complex PLD*) и *FPGA* (*Field Programmable GA*).

Программируемые логические матрицы появились в середине 70-х годов. Основная идея ПЛМ заключается в использовании двух программируемых матриц – матрицы элементов И и матрицы элементов ИЛИ. Простейшая ПЛМ представлена на рис. 2.5. Она имеет три входных сигнала, четыре выходных и восемь промежуточных термов. В чистом (незапрограммируемом) виде обе матрицы содержат плавкие перемычки, которые соединяют входные сигналы (прямые и инверсные) со всеми термами (в матрице И) и каждый из термов с каждым выходным сигналом в матрице ИЛИ. Перемычки показаны на рис. 2.5 в виде выносок. При программировании перемычки расплавляются импульсами тока. Сохраняются лишь те перемычки, которые необходимы для реализации требуемой логической функции (на рис. 2.5 сохраненные перемычки обозначены крестиками).

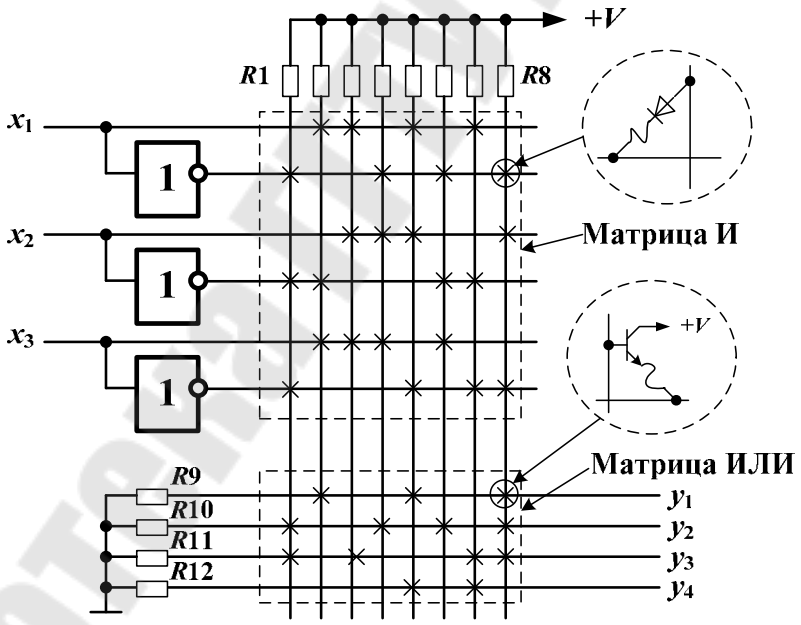


Рис. 2.5. Пример простейшей ПЛМ

Таким образом, в примере на рис. 2.5 реализованы четыре функции от трех переменных:

$$y_1 = x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + \overline{x_1}\overline{x_2}\overline{x_3},$$

$$y_2 = \overline{x_1}\cdot\overline{x_2}\cdot\overline{x_3} + \overline{x_1}\cdot x_2\cdot x_3 + \overline{x_1}\cdot\overline{x_2}\cdot x_3 + \overline{x_1}\cdot x_2\cdot\overline{x_3},$$

$$y_3 = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} + x_1 \cdot x_2 \cdot x_3 + \overline{x_1} \cdot \overline{x_2} \cdot x_3 + x_1 \cdot \overline{x_2} \cdot \overline{x_3},$$

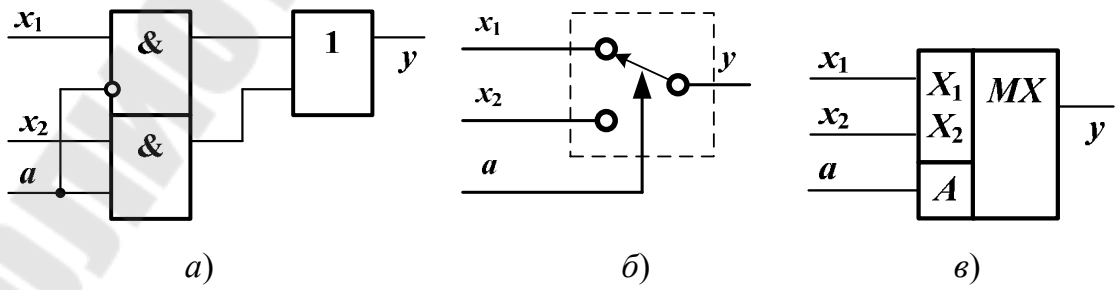
$$y_4 = x_1 \cdot x_2 \cdot \overline{x_3} + x_1 \cdot \overline{x_2} \cdot \overline{x_3}.$$

Как правило, в ПЛМ входят также блоки входных и выходных буферных каскадов. Входные буферы преобразуют однофазные входные сигналы в парафазные сигналы и формируют сигналы необходимой мощности для питания матрицы элементов И. Выходные буферы обеспечивают необходимую нагрузочную способность выходов, могут содержать некоторую логику, выходной регистр или логику отключения выходов от внешней шины.

Основными параметрами ПЛМ являются: число входов *m*, число выходов *n* и число термов *l*, а также наличие выходного регистра. ПЛМ реализует систему из *n* логических функций от *m* переменных, представленных в ДНФ и содержащих не более чем *l* различных термов. Основными производителями СБИС *CPLD* и *FPGA* являются *Xilinx* и *Altera*.

2.2.2. Мультиплексоры

Мультиплексор – функциональный узел, осуществляющий передачу сигнала с любого информационного входа на один выход. Входы мультиплексора подразделяются на информационные и адресные. Адресные входы определяют номер информационного входа, с которого информация попадает на выход. Схема простейшего мультиплексора, содержащего два информационных входа, представлена на рис. 2.6, *а*. На рис. 2.6, *б* представлено упрощенное представление принципа работы этого мультиплексора. Значение адресного входа *a* определяет позицию переключателя: при $a = 0$ переключатель находится в верхнем положении. Поэтому на выход попадает информация с входа x_1. При $a = 1$ переключатель находится в нижнем положении, поэтому на выход попадает информация со входа x_2. На рис. 2.6, *в* представлено УГО этого мультиплексора, в табл. 2.2 представлена таблица истинности.



а) *б*) *в*)

Рис. 2.6. Мультиплексор «2 в 1»: *а* – схема; *б* – принцип работы; *в* – функциональное обозначение

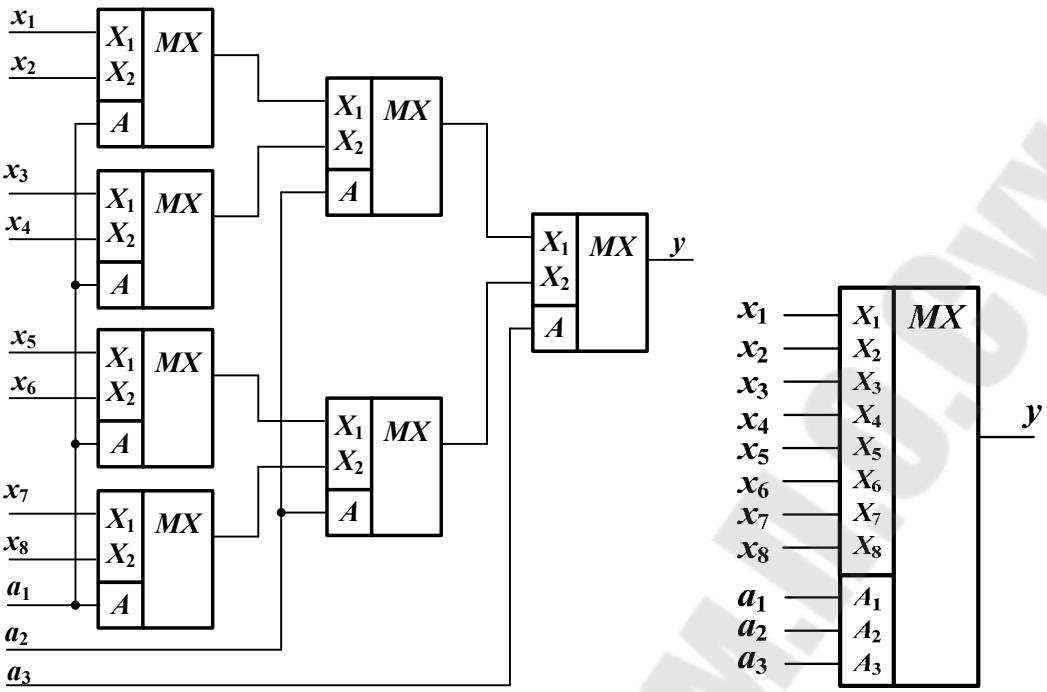
Таблица 2.2

Таблица истинности мультиплексора «2 в 1»

a	x_1	x_2	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

В общем случае мультиплексор при помощи n адресных входов позволяет выбирать один из 2^n информационных сигналов. Промышленностью выпускаются мультиплексоры, имеющие 2, 4, 8 и 16 информационных входов. Для реализации мультиплексоров большей разрядности применяют различные схемы. Наибольшее распространение получила пирамидальная схема наращивания разрядности. Рассмотрим пример построения мультиплексора «8 в 1» из мультиплексоров «2 в 1» (рис. 2.7). В общем случае первый ярус пирамиды состоит из k мультиплексоров. Значение k определяется из соотношения $k = n/m$, где n – требуемое число входов, m – число входов имеющихся мультиплексоров. Таким образом, первый ярус содержит k мультиплексоров «m в 1». Следующий ярус содержит k/m мультиплексоров «m в 1» и т. д. Последний ярус содержит один мультиплексор «m в 1». В нашем случае: $n = 8$, $m = 2$, поэтому будет три яруса, причем в первом ярусе $k = 4$ мультиплексора «2 в 1», во втором ярусе – два, в третьем ярусе – один. На рис. 2.7, *а* приведена пирамидальная схема мультиплексора, а на рис. 2.7, *б* – его условное графическое обозначение.

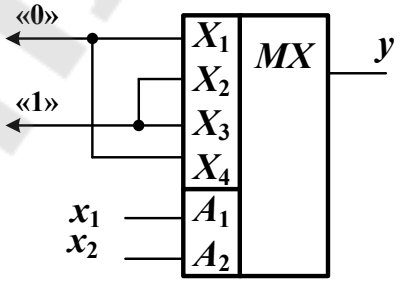
На основе мультиплексоров строят универсальные логические модули (УЛМ). Универсальность этих модулей заключается в том, что они могут быть настроены на реализацию произвольной логической функции. Самым простым способом настройки УЛМ является фиксация информационных входов. При этом на адресные входы мультиплексора подаются аргументы функции, а на информационные входы – таблица настройки. Рассмотрим пример настройки мультиплексора на реализацию функции $y = x_1 \oplus x_2$. Функция имеет два аргумента, поэтому можно использовать мультиплексор с двумя адресными входами «4 в 1». Таблица истинности представлена на рис. 2.8, *а*, а реализация данной функции – на рис. 2.8, *б*.



а) *б*)

Рис. 2.7. Наращивание разрядности мультиплексора (*а*) и условное графическое обозначение мультиплексора «8 в 1» (*б*)

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



а) *б*)

Рис. 2.8. Применение мультиплексора для реализации произвольной функции: *а* – таблица истинности; *б* – логическая схема

Основное достоинство данного способа – простота реализации произвольной функции от фиксированного числа аргументов. Общее число функций от n аргументов определяется как 2^{2^n}, поэтому с ростом n растет очень быстро. Применение мультиплексоров позволяет легко реализовать любую из возможных функций. Заметим, что с ростом числа адресных входов мультиплексора возрастают и аппаратные затраты на его реализацию, поэтому на практике рассмотренный подход применим для функций, имеющих сравнительно небольшое число

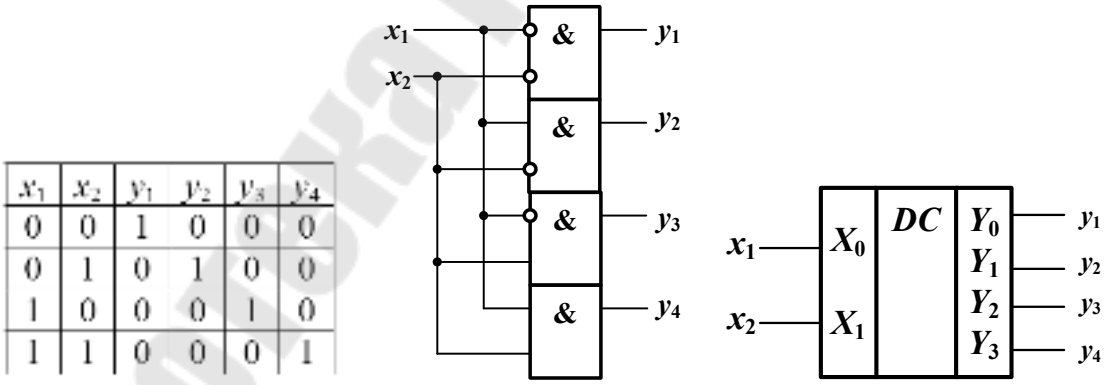
аргументов. Для увеличения числа аргументов функции используют различные варианты наращивания. В частности, применяют пирамидальные структуры, аналогичные тем, которые используются для наращивания разрядности мультиплексоров.

2.2.3. Дешифраторы

Дешифраторы относятся к преобразователям кодов. Двоичные дешифраторы преобразуют двоичный код в код «1 из *N*». В данном коде из *N* позиций только одна позиция занята единицей, а все остальные – нулевые. Например, код «1 из 4» содержит следующие кодовые комбинации:

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1.

Таким образом, дешифратор, который имеет *n* входов, должен иметь 2^n выходов. Причем только на одном из выходов дешифратора (который соответствует кодовой комбинации на входе) будет уровень логической единицы, а на всех остальных – логический ноль (рис. 2.9, *а*). Если часть кодовых комбинаций не используется, то дешифратор называется неполным и имеет меньшее число выходов. Схема дешифратора 2–4 и его УГО приведены на рис 2.9, *б*, *в*, соответственно.



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

а) *б*) *в*)

Рис. 2.9. Простейший дешифратор «2–4»:
а – таблица истинности; *б* – логическая схема; *в* – УГО.

В общем случае для наращивания разрядности дешифраторов используют различные подходы. По аналогии с наращиванием разрядности мультиплексора, применяют пирамидальные структуры (рис. 2.10).

В этом случае дешифратор должен содержать дополнительный вход разрешения работы *E*. Пример реализации дешифратора «4–16» на дешифраторах «2–4» представлен на рис. 2.10, *а*. На рис. 2.10, *б* приведено УГО этого дешифратора.

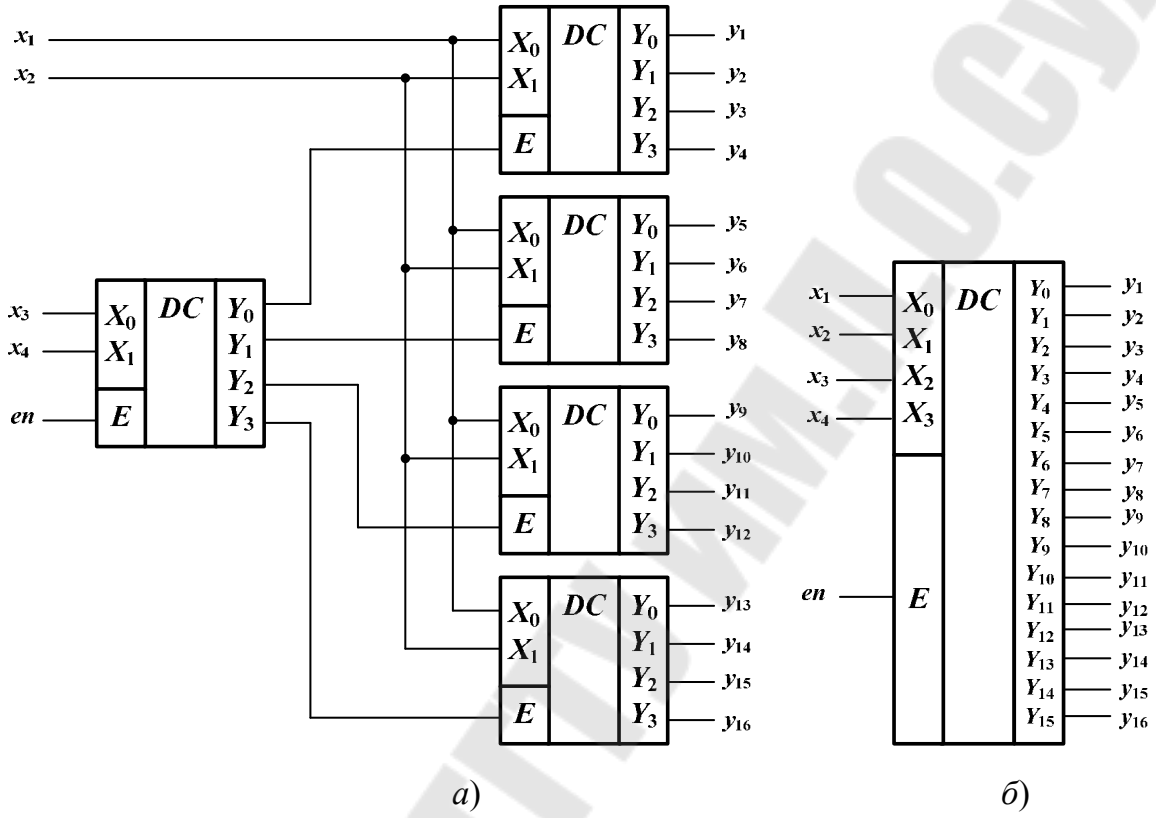


Рис. 2.10. Наращивание разрядности дешифратора (*а*)
и УГО дешифратора «4–16» (*б*)

Однако многокаскадные структуры снижают быстродействие схемы. Поэтому для достижения максимального быстродействия стремятся минимизировать количество логических элементов, расположенных между входом в схему и ее выходом. Пример дешифратора «6–64», спроектированного по данному принципу, представлен на рис. 2.11.

Как компромисс между требованиями по быстродействию и аппаратными затратами применяют предварительную дешифрацию. На рис. 2.12 приведена схема дешифратора «6–64», спроектированного с использованием предварительной дешифрации.

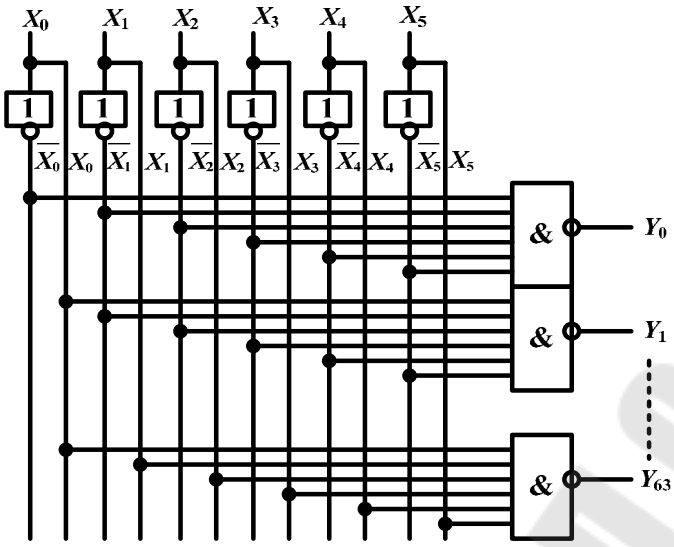


Рис. 2.11. Дешифратор «6–64»

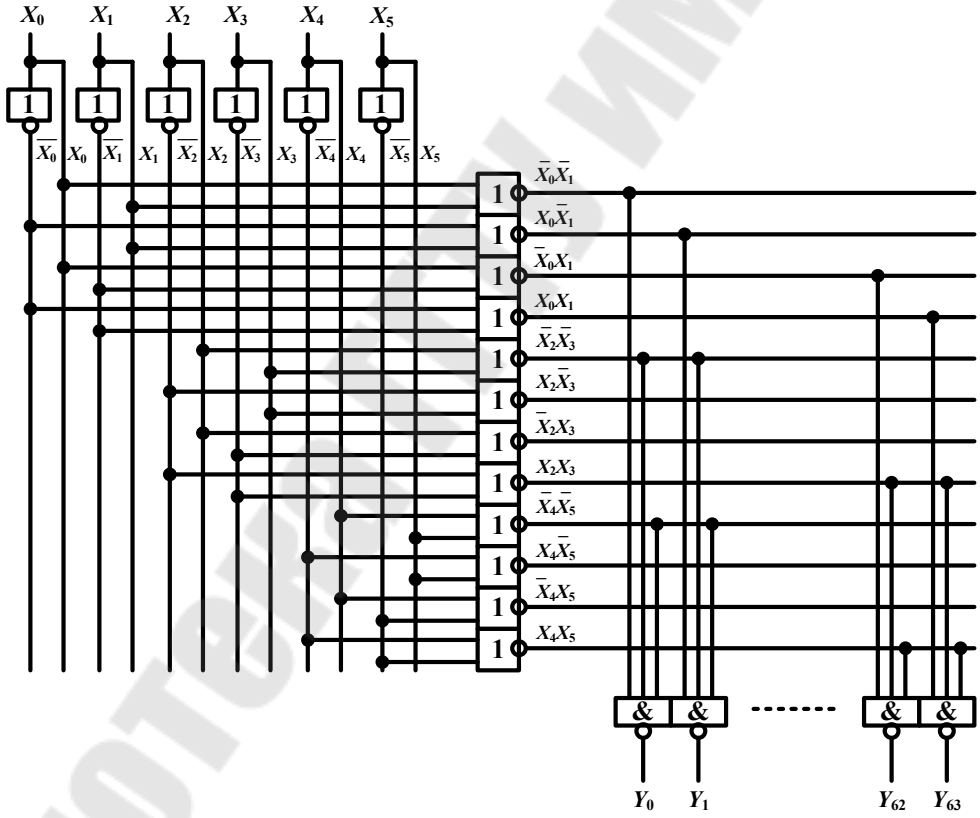


Рис. 2.12. Дешифратор «6–64» с предварительной дешифрацией

Сравним схемы, приведенные на рис. 2.11 и 2.12 по быстродействию, аппаратным затратам и энергопотреблению. При сравнении будем считать, что задержка распространения сигнала у всех логических элементов одинакова и равна $t = 1$ нс. Инвертор реализуется на

двух транзисторах, двухвходовой элемент ИЛИ-НЕ – на четырех транзисторах, трехвходовой элемент И-НЕ – на четырех транзисторах, шестивходовой элемент И-НЕ – на семи транзисторах (данные взяты из библиотеки КМОП элементов *AMI*3*LS* (*American Microsystems*, *Inc*.)).

Максимальная задержка распространения сигнала для схемы на рис. 2.11 составляет 2*t*, соответственно максимальная частота работы – 500 МГц. Этот параметр для схемы на рис. 2.12 составляет 3*t*, соответственно максимальная частота работы – 333 МГц. Для реализации дешифратора на рис. 2.11 требуется шесть инверторов и 64 шестивходовых элемента И-НЕ, т. е. общее число транзисторов равно $K_1 = 6 \cdot 2 + 64 \cdot 7 = 460$ транзисторов. Для реализации дешифратора на рис. 2.12 требуется шесть инверторов, 12 двухвходовых элементов ИЛИ-НЕ и 64 трехвходовых элемента И-НЕ. Таким образом, аппаратные затраты составят $K_2 = 6 \cdot 2 + 12 \cdot 4 + 64 \cdot 4 = 316$ транзисторов.

Получили, что схема на рис. 2.12 позволяет снизить примерно на 30 % аппаратные затраты на реализацию дешифратора «6–64», при этом быстродействие снижается на 33 %.

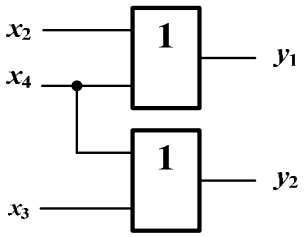
Сравним дешифраторы на рис. 2.11 и 2.12 с точки зрения переключаемой емкости, которая определяет динамическую потребляемую мощность. Изменение одного разряда адреса для схемы на рис. 2.9 приводит к изменению логического уровня на 65 входах логических элементов (одно переключение на входе инвертора и 64 переключения на входах элементов И-НЕ). Для схемы на рис. 2.12 изменение одного разряда адреса приводит к 37 переключениям на входах логических элементов (одно переключение на входе инвертора, четыре переключения на входах элементов ИЛИ-НЕ и 32 переключения на входах элементов И-НЕ). Таким образом, применение предварительной дешифрации позволяет не только снизить аппаратные затраты, но и уменьшить потребляемую мощность.

Шифратор – функциональный узел ЭВМ, выполняющий функцию, обратную дешифратору. Шифратор имеет 2^n входов и n выходов. При подаче логической единицы на один из его входов на выходе формируется двоичный код, соответствующий номеру этого входа. Другими словами, шифратор преобразует код «1 из N» в двоичный код. Поэтому только один из входов шифратора может быть активен. На рис. 2.13 приведена таблица истинности, логическая схема и УГО простейшего шифратора «4–2». Заметим, что в соответствии с таблицей истинности логическая схема определяется выражениями

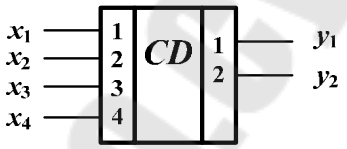
$y_1 = x_2 + x_4$ и $y_2 = x_3 + x_4$, среди аргументов которых нет x_1. Таким образом, код для состояния, когда активизирован первый вход, совпадает с кодом, когда ни на одном входе нет активных сигналов.

x_1	x_2	x_3	x_4	y_1	y_2
1	0	0	0	0	0
0	1	0	0	1	0
0	0	1	0	0	1
0	0	0	1	1	1

а)



б)



в)

Рис. 2.13. Простейший шифратор «4–2»:
а – таблица истинности; *б* – логическая схема; *в* – УГО

Кроме двоичных шифраторов существуют и приоритетные шифраторы, выполняющие более сложную функцию. Основное отличие приоритетных шифраторов – возможность активизации сразу нескольких входов. Соответственно, задача приоритетного шифратора заключается в том, чтобы сформировать на своих выходах код, соответствующий активизированному входу с наивысшим приоритетом. Простейший способ задания приоритета заключается в том, что вывод с большим номером имеет более высокий приоритет. Соответственно, при одновременном появлении активных сигналов на нескольких входах приоритетный шифратор должен сформировать код, соответствующий активизированному входу с максимальным номером.

2.2.4. Сумматоры

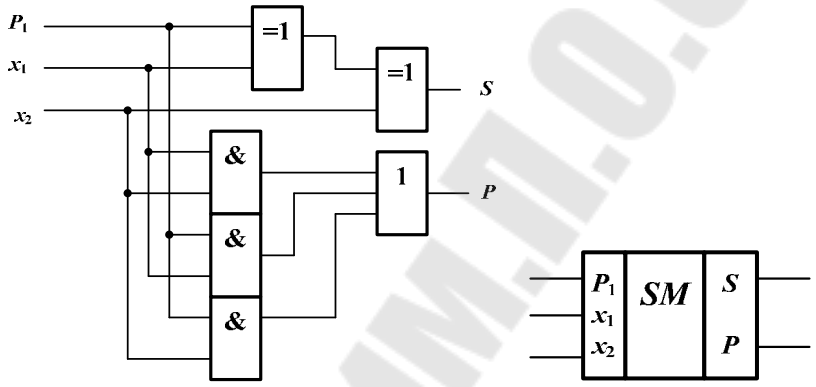
Сумматоры выполняют арифметическое операции, такие, как сложение и вычитание двоичных чисел. Они являются неотъемлемой частью любого арифметико-логического устройства (АЛУ), которое в свою очередь является ядром любого процессора. По аналогии с десятичной арифметикой, суммирование чисел в двоичной арифметике происходит поразрядно, начиная с младшего разряда. При этом, кроме суммы, каждый разряд должен вырабатывать сигнал переноса в старший разряд. Таким образом, одноразрядный сумматор имеет три входа – два слагаемых (x_1 и x_2) и перенос из предыдущего разряда (P_1) и два выхода – суммы (S) и переноса в старший разряд (P). Таблица истинности одноразрядного сумматора приведена на рис. 2.14, *а*. Запишем логические выражения для суммы и переноса:

$$S = \overline{x_1} \cdot \overline{x_2} \cdot p_1 + x_1 \cdot \overline{x_2} \cdot \overline{p_1} + x_1 \cdot x_2 \cdot p_1 = x_1 \oplus x_2 \oplus p_1,$$

$$P = \overline{x_1} \cdot x_2 \cdot p_1 + x_1 \cdot \overline{x_2} \cdot p_1 + x_1 \cdot x_2 \cdot \overline{p_1} + x_1 \cdot x_2 \cdot p_1 = x_1 \cdot x_2 + x_1 \cdot p_1 + x_2 \cdot p_1.$$

На основании этих выражений строим схему сумматора (рис. 2.14, *б*). Условное графическое обозначение сумматора приведено на рис. 2.14, *в*.

x_1	x_2	P_1	S	P
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



а) *б*) *в*)

Рис. 2.14. Простейший одноразрядный сумматор:
а – таблица истинности; *б* – логическая схема; *в* – УГО

По аналогии с сумматорами легко реализовать вычитатели, однако в этом нет смысла, так как вычитание выполняется через сложение с применением дополнительных либо обратных кодов. Многоразрядный двоичный сумматор строится на основе одноразрядных сумматоров. Пример четырехразрядного сумматора представлен на рис. 2.15. На вход сумматора поступают два четырехразрядных числа – $A = a_0a_1a_2a_3$ и $B = b_0b_1b_2b_3$ (младший разряд слева). На выходе сумматора формируется четырехразрядная сумма $S = s_0s_1s_2s_3$ и сигнал переноса P.

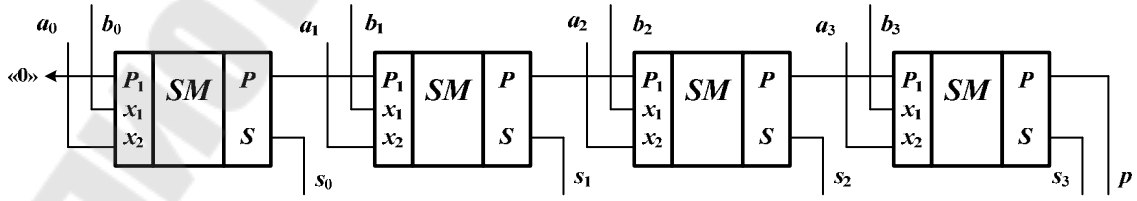


Рис. 2.15. Простейший четырехразрядный сумматор

Приведенная схема сумматора называется сумматором с последовательным переносом. Схема проста, но обладает существенным

недостатком – низкое быстродействие из-за последовательного распространения переноса. Худшим является случай, когда перенос, возникший в младшем разряде, распространяется до самого старшего разряда формируемой суммы.

В общем случае в устройствах вычислительной техники используются различные сумматоры, различающиеся по принципу действия (параллельные или последовательные), по числу обрабатываемых разрядов (полусумматоры, полные сумматоры, многоразрядные сумматоры), по методу формирования суммы (комбинационные, накопительные), по способу организации переноса между разрядами (последовательный перенос, параллельный перенос, комбинированные).

Пример сумматора, который позволяет накапливать результат, представлен на рис. 2.16.

Рис. 2.16. Накапливающий сумматор

Ко входам комбинационного сумматора подключены регистры слагаемых *A* и *B*, а к выходу сумматора подключен регистр суммы *C*. Выход регистра *C* через мультиплексор «2–1» подключен ко входу регистра *B*. Таким образом, сумматор позволяет суммировать результат предыдущего сложения с новым значением A, т. е. накапливать результат.

2.2.5. Компараторы

Компараторы, или устройства сравнения, определяют отношения между двумя двоичными словами. Основными отношениями являются «равно», «больше» и «меньше». Остальные отношения («не равно», «меньше или равно», «больше или равно») легко могут быть получены на основании этих соотношений. Как правило, операция сравнения реализована в большинстве выпускаемых арифметико-логических устройств. Однако выпускаются и специализированные микросхемы, реализующие функцию сравнения, например КР1561ИП2 (рис. 2.17).

Определим функции сравнения для компаратора. При этом в качестве основных функций будем использовать две – «равно» и «больше». Пусть $f(A = B) = 1$, если $A = B$ и $f(A = B) = 0$ при $A \neq B$. По аналогии, $f(A > B) = 1$, если $A > B$ и $f(A > B) = 0$ при $A \leq B$. Тогда для остальных отношений можно записать: $f(A \neq B) = \overline{f(A = B)}$, $f(A < B) = f(B > A)$, $f(A \geq B) = \overline{f(B > A)}$, $f(A \leq B) = \overline{f(A > B)}$.

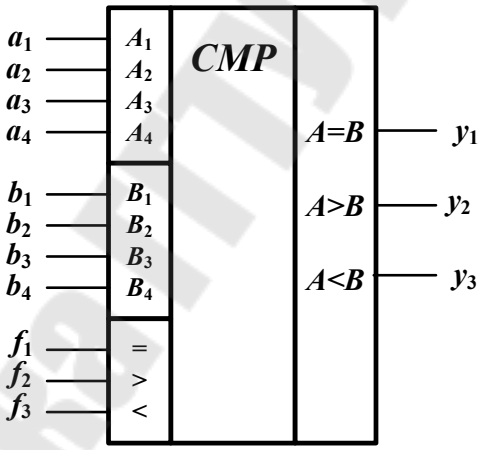


Рис. 2.17. Компаратор

Рассмотрим работу компаратора для двух основных функций. Устройства сравнения на равенство строятся на основе поразрядных операций над одноименными разрядами обоих слов. Слова равны, если равны все одноименные их разряды, т. е. если в обоих нули или единицы. Признак равенства i-го разряда – $r_i = a_i b_i + \overline{a_i b_i} = \overline{a_i \oplus b_i}$, признак неравенства – $\overline{r_i} = a_i \oplus b_i$. Числа не равны, если активен признак неравенства хотя бы для одного разряда. Поэтому, выполнив поразрядное сравнение на неравенство при помощи сумматора по модулю два (элемент Исключающее ИЛИ), необходимо проверить, нет ли

активного признака хотя бы для одного разряда. Проверку выполним при помощи *n*-разрядного элемента ИЛИ-НЕ (рис. 2.18). На схеме если на одном или нескольких выходах элементов Исключающее ИЛИ присутствует «1», то на выходе элемента ИЛИ-НЕ будет «0». Значение выхода будет равно «1» только в том случае, когда на всех входах элемента ИЛИ-НЕ будет «0», т. е. все разряды сравниваемых чисел совпадают.

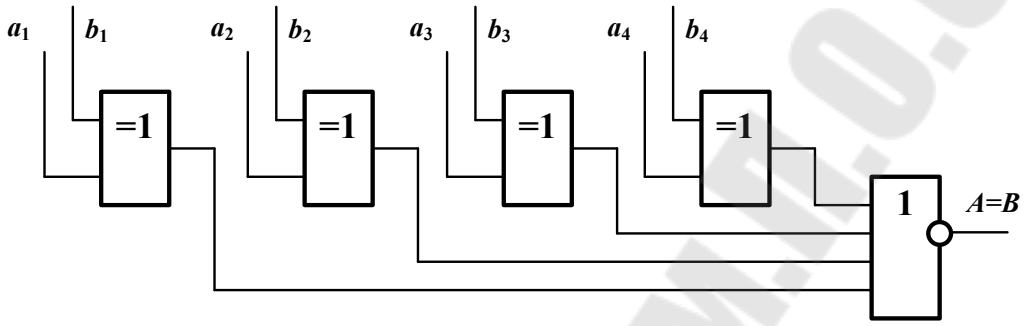


Рис. 2.18. Компаратор, выполняющий сравнение чисел на равенство

Устройства сравнения на «больше» могут быть реализованы различными способами. Одним из наиболее простых способов является использование *n*-разрядного сумматора для сравнения *n*-разрядных слов. Основная идея заключается в следующем. Если число A больше чем B, то их разность больше нуля. Отрицательные числа в ЭВМ представляются в обратном или дополнительном коде, причем самый старший разряд числа содержит знак числа – «0» для положительного и «1» для отрицательного числа. Поэтому после нахождения разности чисел остается проанализировать только один разряд – знаковый. Если его значение равно «0», то $A > B$, в противном случае $A \le B$. Операция вычитания осуществляется на сумматорах, при этом вычитаемое должно быть представлено в обратном или дополнительном коде.

Сравнение на «больше» может быть также реализовано на логике, на основе следующих рассуждений. Компаратор на «больше» для одноразрядных слов требует реализации функции $f(A > B) = a\bar{b}$ (рис. 2.19, *а*). Компаратор для двухразрядных слов также легко может быть реализован на основе таблицы истинности (рис. 2.19, *б*). Однако при возрастании разрядности на единицу размер таблицы истинности увеличивается в четыре раза, что ведет к значительному увеличению аппаратных затрат. Поэтому остановимся только на основных принципах синтеза многоразрядных компараторов на «больше».

Пусть требуется сравнить двухразрядные слова. Если старшие разряды a_1 и b_1 не равны, то результат известен независимо от младших разрядов: при $a_1 = 1$ и $b_1 = 0$ имеем $A > B$, а при $a_1 = 0$ и $b_1 = 1$ – $A < B$. Если же $a_1 = b_1$, то результат еще неизвестен и требуется анализ следующего разряда по тому же алгоритму. Поэтому для двухразрядных слов можно записать: $f(A > B) = a_1\bar{b}_1 + r_1 a_0 \bar{b}_0$. Данный подход может быть использован и для слов произвольной разрядности. К анализу следующего разряда нужно переходить только при равенстве предыдущих (старших) разрядов.

a	b	$f(A>B)$
0	0	0
0	1	0
1	0	1
1	1	0

а)

a_1	a_0	b_1	b_0	$f(A>B)$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

б)

Рис. 2.19. Таблицы истинности компараторов:
а – одноразрядного; *б* – двухразрядного

2.2.6. Умножители

Операция умножения, которая широко используется в обработке данных, является достаточно сложной с точки зрения аппаратной реализации. Прежде всего это связано со структурой математических выражений, описывающих операцию умножения.

Пусть имеются два целых n-разрядных двоичных числа без знаков – $A=a_{n-1}a_{n-2}\ldots a_1a_0$ и $B = b_{n-1}b_{n-2}\ldots b_1b_0$. Их перемножение выполняется по известной схеме «умножения столбиком», т. е. формируют-

ся промежуточные произведения вида $A \cdot b_i$, где $b_i \in \{0, 1\}$, $i = \overline{0, n-1}$, сдвинутые на i позиций влево. Например, для четырехразрядных чисел $A = 1001$ и $B = 0011$:

$A = 1\ 0\ 0\ 1$

$*$

$B = 0\ 0\ 1\ 1$

$A \cdot b_0 = A$	$1\ 0\ 0\ 1$
$A \cdot b_1 = A$	$1\ 0\ 0\ 1$
$A \cdot b_2 = 0$	$0\ 0\ 0\ 0$
$A \cdot b_0 = A$	$0\ 0\ 0\ 0$
$A \cdot B =$	$0\ 0\ 0\ 1\ 1\ 0\ 1\ 1$

Получили, что для выполнения умножения необходимо выполнить ряд операций сдвига и сложения, что приводит к снижению быстродействия вычислительных устройств. Поэтому для быстродействующих устройств были разработаны различные схемы *умножителей*, позволяющие выполнять операцию умножения за время, соизмеримое с временем выполнения таких операций, как сложение, вычитание, сравнение и т. п. Одним из первых способов реализации операции умножения было использование постоянного запоминающего устройства (ПЗУ), в котором записаны результаты умножения. В качестве адреса использовались значения сомножителей. Недостатком данного способа являются большие аппаратные затраты на реализацию ПЗУ. Так, для реализации операции умножения четырехразрядных чисел требуется ПЗУ объемом 256 байт, а для восьмиразрядных чисел – 64 Кслова или 128 Кбайт. Поэтому были спроектированы специализированные микросхемы умножителей. Как правило, эти умножители входят в микропроцессорные комплекты БИС. Например, в серии КР1802 есть умножитель 8 х 8 (КР1802ВР3), 12 х 12 (КР1802ВР4), 16 х 16 (КР1802ВР5). НПО «Интеграл» выпускает умножитель 32 х 32 (КА1843ВР1), который выполняет умножение за 250 нс.

Рассмотрим пример синтеза двухразрядного параллельного умножителя, структурная схема которого приведена на рис. 2.20, таблица истинности умножителя представлена в табл. 2.3.

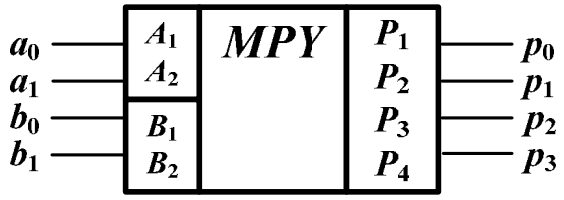


Рис. 2.20. Двухразрядный умножитель

Таблица 2.3

Таблица истинности двухразрядного умножителя

A		*B*		*P*			
a_1	a_0	b_1	b_0	p_3	p_2	p_1	p_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

На вход умножителя поступают двухразрядные числа A и B, на выходе формируется четырехразрядное произведение P. На основании таблицы истинности составляем функции для разрядов произведения P в СДНФ:

$$p_3 = a_1 a_0 b_1 b_0,$$

$$p_2 = a_1 \overline{a_0} b_1 \overline{b_0} + a_1 \overline{a_0} b_1 b_0 + a_1 a_0 b_1 \overline{b_0},$$

$$p_1 = \overline{a_1} a_0 b_1 \overline{b_0} + \overline{a_1} a_0 b_1 b_0 + a_1 \overline{a_0}\,\overline{b_1} b_0 + a_1 \overline{a_0} b_1 b_0 + a_1 a_0 \overline{b_1} b_0 + a_1 a_0 b_1 \overline{b_0},$$

$$p_0 = \overline{a_1} a_0 \overline{b_1} b_0 + \overline{a_1} a_0 b_1 b_0 + a_1 a_0 \overline{b_1} b_0 + a_1 a_0 b_1 b_0.$$

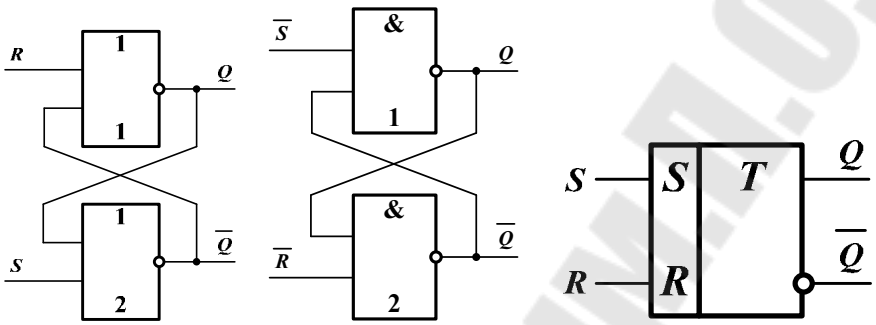
На основании полученных выражений после минимизации реализуется схема умножителя.

2.3. Последовательностные узлы ЭВМ

2.3.1. Триггеры

В последовательностных схемах выходные сигналы зависят не только от комбинаций входных сигналов, но и от значений выходных

сигналов в предшествующий момент времени. Простейшей последовательностной схемой является *триггер* (или элементарный цифровой автомат), содержащий, как правило, элемент памяти и схему управления. Основу триггера составляет элемент памяти, выполненный на двух логических элементах И-НЕ (ИЛИ-НЕ). Схема управления определяет логику работы триггера. Пример триггера с простейшей схемой управления, реализованного на элементах И-НЕ и ИЛИ-НЕ (так называемый *RS*-триггер), приведен на рис. 2.21.



а) *б*) *в*)

Рис. 2.21. RS-триггер на логике ИЛИ-НЕ (*а*), И-НЕ (*б*) и его функциональное обозначение (*в*)

Триггер имеет два входа: S – установка и R – сброс и два выхода: прямой – Q и инверсный – $\overline{Q}$. Состояние триггера определяется значением его прямого выхода. Кроме того, различают состояния триггера в смежные моменты времени – t (текущий момент времени) и $t+1$ (следующий момент времени). Соответственно, состояние триггера в текущий момент времени будем обозначать Q_t, а состояние триггера в следующий момент времени – Q_{t+1}.

Рассмотрим работу схемы на рис. 2.21, *а*. Пусть $Q_t = 1$ (соответственно $\overline{Q_t} = 0$), а на входах S и R присутствуют уровни нуля. Тогда единица на выходе Q поступает на первый вход второго элемента 2ИЛИ-НЕ и формирует на его выходе значение нуля. В результате на обоих входах первого элемента 2ИЛИ-НЕ присутствуют нули, соответственно на его выходе будет единица. В данном состоянии триггер может находиться неограниченно долго. Изменить состояние триггера можно, подав на его управляющие входы комбинацию $R = 1$, $S = 0$. Единица на входе R переключит триггер в нулевое состояние ($Q_t = 0$). Соответственно, на оба входа второго элемента 2ИЛИ-НЕ поступят

нули, соответственно на его выходе будет единица ($\overline{Q_t} = 1$). Комбинация сигналов $S = R = 1$ не определена.

Схема на рис. 2.21, *б* работает аналогичным образом. Отличие заключается в том, что активным уровнем на входах $\overline{S}$ и $\overline{R}$ является ноль, соответственно комбинация $\overline{S} = \overline{R} = 1$ соответствует режиму хранения. Подав на входы: $\overline{S} = 0$, $\overline{R} = 1$, получим на выходе первого элемента 2И-НЕ значение единицы ($Q_t = 1$). На оба входа второго элемента 2И-НЕ поступают единицы, поэтому на его выходе будет ноль ($\overline{Q_t} = 0$). Изменить состояние триггера можно путем подачи на его входы комбинации $\overline{S} = 1$, $\overline{R} = 0$. Для этого триггера комбинация $\overline{S} = \overline{R} = 0$ не определена. В соответствии с данным описанием составим таблицу состояний триггера (табл. 2.4).

Таблица 2.4

Таблица состояний *RS*-триггера

Триггер на элементах 2ИЛИ-НЕ					**Режим**	**Триггер на элементах 2И-НЕ**				
S	R	Q_t	Q_{t+1}	$\overline{Q_{t+1}}$		$\overline{S}$	$\overline{R}$	Q_t	Q_{t+1}	$\overline{Q_{t+1}}$
0	0	Q	Q	$\overline{Q}$	**Хранение**	1	1	Q	Q	$\overline{Q}$
0	1	Q	0	1	**Сброс**	1	0	Q	0	1
1	0	Q	1	0	**Установка**	0	1	Q	1	0
1	1	Q	x	x	**Не определено**	0	0	Q	x	x

В общем случае для входов триггеров приняты следующие стандартные обозначения:

– *R* (*Reset* – сброс) – вход установки триггера в состояние «0»;

– *S* (*Set* – установка) – вход установки триггера в состояние «1»;

– *K* (*Kill* – внезапное отключение) – вход установки универсального триггера в состояние «0»;

– *J* (*Jerk* – внезапное включение) – вход установки универсального триггера в состояние «1»;

– *T* (*Toggle* – переключать) – счетный вход триггера;

– *D* (*Delay* – задержка) – информационный вход триггера;

– *C* (*Clock* – синхронизировать) – вход синхронизации;

– *V* (*Valve* – вентиль) – разрешающий вход.

Рассмотрим свойства лишь наиболее распространенных типов триггеров, классификация которых приведена на рис. 2.22.

Рис. 2.22. Классификация триггеров

По *логике работы* различают триггеры *RS*-, *D*-, *JK*- и *T*-триггеры. Ниже эти типы триггеров будут рассмотрены более подробно.

По *способу записи информации* триггеры подразделяются на асинхронные и синхронные.

Запись информации в *асинхронный* триггер происходит в момент изменения управляющих или информационных сигналов.

В *синхронных* триггерах для записи информации используется специальный синхронизирующий вход *C*, сигнал которого разрешает триггеру принять новую информацию.

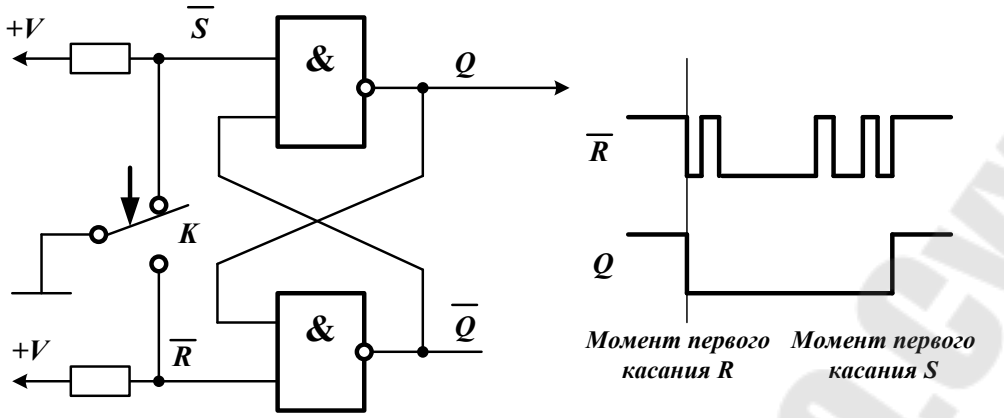
В свою очередь, синхронные триггеры подразделяются на управляемые уровнем синхросигнала и управляемые фронтом. Триггеры, управляемые уровнем синхроимпульса, разрешают запись информации при наличии разрешающего уровня на входе синхронизации. Триггеры, управляемые фронтом синхроимпульса, разрешают запись информации в момент изменения логического уровня на входе синхронизации.

По *числу ступеней* триггеры делятся на одноступенчатые и двухступенчатые. В соответствии с ГОСТ 2.743–82, одноступенчатые триггеры обозначаются буквой *T* в основном поле, а двухступенчатые – двумя (*TT*).

Асинхронный *RS*-триггер. Схема асинхронного *RS*-триггера приведена на рис. 2.21. В качестве самостоятельных устройств эти триггеры находят ограниченное применение, но служат основой всех более сложных триггеров-устройств. Одним из применений асинхронного *RS*-триггера является схема подавления «дребезга контактов» кнопок или клавиш клавиатуры. В процессе однократного нажатия клавиши происходит многократное замыкание и размыкание контактов, что приводит к многократному срабатыванию исполнительного устройства (в случае клавиатуры это многократная печать одного символа). Чтобы устранить этот эффект, применяют простейшую схему на основе асинхронного *RS*-триггера, приведенную на рис. 2.23.

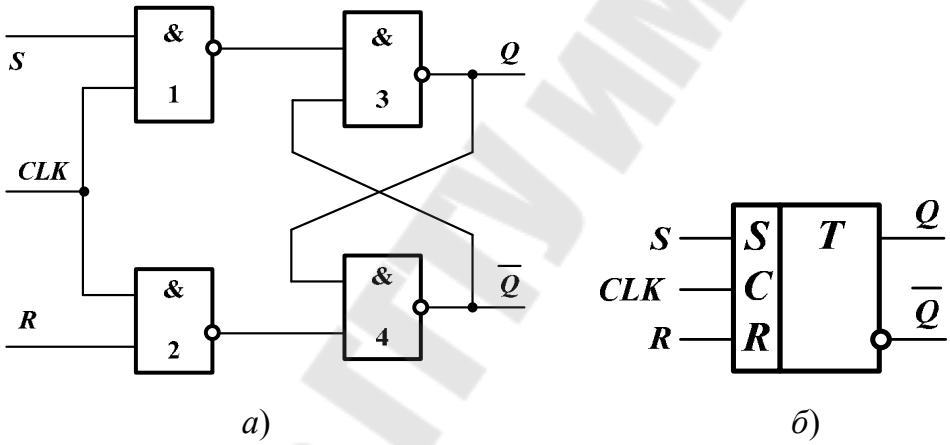
В устройствах цифровой обработки информации в основном применяются синхронизируемые (тактируемые) триггеры.

Синхронный *RS*-триггер. В отличие от асинхронных, синхронные *RS*-триггеры имеют на входе дополнительные логические схемы, которые «привязывают» управляющие сигналы к импульсам синхронизации. Таким образом, информация с входов *R* и *S* поступает на триггер только при наличии активного уровня импульса синхронизации *CLK*. Пример синхронного *RS*-триггера, реализованного на логике И-НЕ, представлен на рис. 2.24.



а) *б*)

Рис. 2.23. Примеры использования *RS*-триггера для подавления «дребезга контактов» (*а*) и временная диаграмма выходного сигнала (*б*)



а) *б*)

Рис. 2.24. Пример синхронного *RS*-триггера на логике И-НЕ (*а*) и его функциональное обозначение (*б*)

При наличии на входе *C* логического нуля на выходах первого и второго элементов будет значение логической единицы и собственно триггер, реализованный на третьем и четвертом элементах И-НЕ, будет сохранять свое значение вне зависимости от состояния входов *R* и *S*. Переключение триггера возможно только при $C = 1$. В этом случае при $R = 0$ и $S = 1$ триггер установится в единичное состояние ($Q = 1$), а при $R = 1$ и $S = 0$ триггер сбросится в ноль ($Q = 0$). То есть при $C = 1$ синхронный триггер работает так же, как и асинхронный.

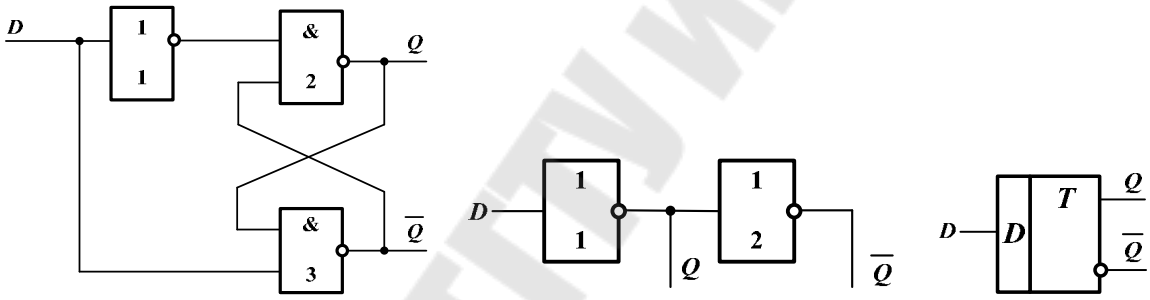
***D*-триггер.** *D*-триггер имеет один информационный вход *D*. Простейшая его реализация – асинхронный *D*-триггер, работа которого определяется таблицей состояний (табл. 2.5).

Таблица 2.5

Таблица состояний *RS*-триггера

D_t	Q_t	Q_{t+1}
0	Q	0
1	Q	1

На основании табл. 2.5 можно записать уравнение, описывающее функционирование *D*-триггера: $Q_{t+1} = D_t$. То есть состояние *D*-триггера в момент времени $t + 1$ совпадает со значением входного сигнала *D* в момент времени *t*. Схема *D*-триггера, спроектированного на основании данного уравнения, представлена на рис. 2.25, *а*. Однако после упрощения получаем схему на рис. 2.25, *б*, которая состоит из двух последовательно включенных инверторов. Поэтому данная схема не имеет практического применения. Исключение составляет КМОП-технология, где данная схема находит применение.

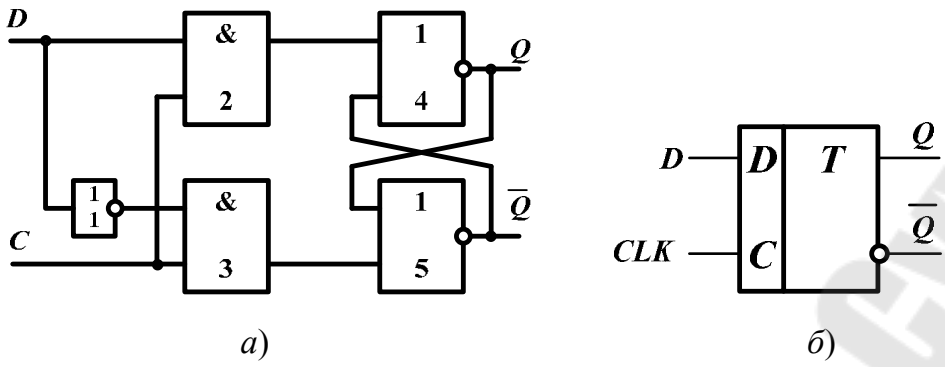


а) *б*) *в*)

Рис. 2.25. Пример *D*-триггера на логике И-НЕ (*а*), эквивалентная схема асинхронного *D*-триггера (*б*) и его функциональное обозначение (*в*)

Больший интерес представляют синхронные *D*-триггеры, которые получили широчайшее распространение при проектировании современных цифровых устройств. *D*-триггер имеет два входа: информационный вход *D* и вход синхронизации *C*, который управляет записью (защелкиванием) информации в триггер. Поэтому в литературе такой триггер часто называют триггер-защелка или просто «защелка» (от англ. *latch* – защелкивание).

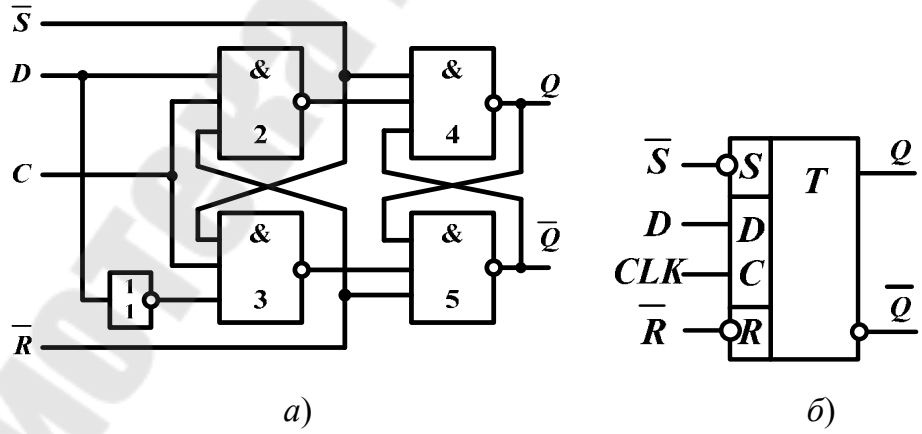
Пример синхронного *D*-триггера и его функциональное обозначение приведены на рис. 2.26. При $C = 1$ выход триггера *Q* принимает значение, равное значению входа *D* ($Q = D$), а при $C = 0$ триггер сохраняет предыдущее значение.



а) *б*)

Рис. 2.26. Пример синхронного *D*-триггера (*а*) и его функциональное обозначение (*б*)

В общем, асинхронных *D*-триггеров не существует, поэтому определение «синхронный» по отношению к *D*-триггеру является избыточным. Поэтому в дальнейшем, говоря о *D*-триггере, будем опускать слово «синхронный». Однако большое распространение получили *D*-триггера, которые имеют асинхронные входы установки *R* и *S*. Пример такого *D*-триггера и его функциональное обозначение приведены на рис. 2.27. Основным недостатком данного триггера является то, что при наличии разрешающего сигнала на входе синхронизации изменение значения входа *D* приводит к изменению значения выхода триггера. Соответственно, триггер запоминает то значение входа *D*, которое было в момент снятия разрешающего сигнала на входе синхронизации.

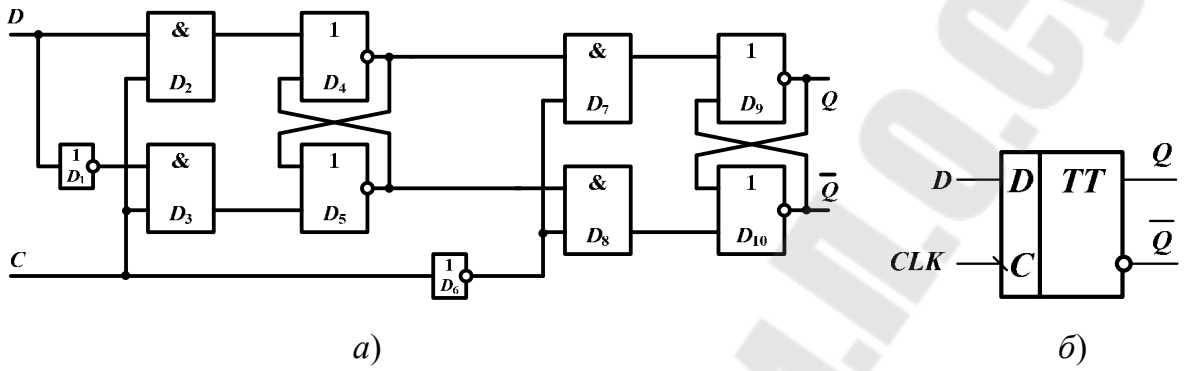


а) *б*)

Рис. 2.27. Пример синхронного *D*-триггера (*а*) и его функциональное обозначение (*б*)

Для устранения этого недостатка используют двухступенчатый триггер, который строится на основе двух одноступенчатых, имею-

щих различные источники синхронизации (сдвинутые во времени друг относительно друга) или один источник синхроимпульсов, но подаваемый на разные ступени в инверсном виде. Пример двухступенчатого *D*-триггера и его функциональное обозначение приведены на рис. 2.28.



а) *б*)

Рис. 2.28. Пример синхронного *D*-триггера (*а*)
и его функциональное обозначение (*б*)

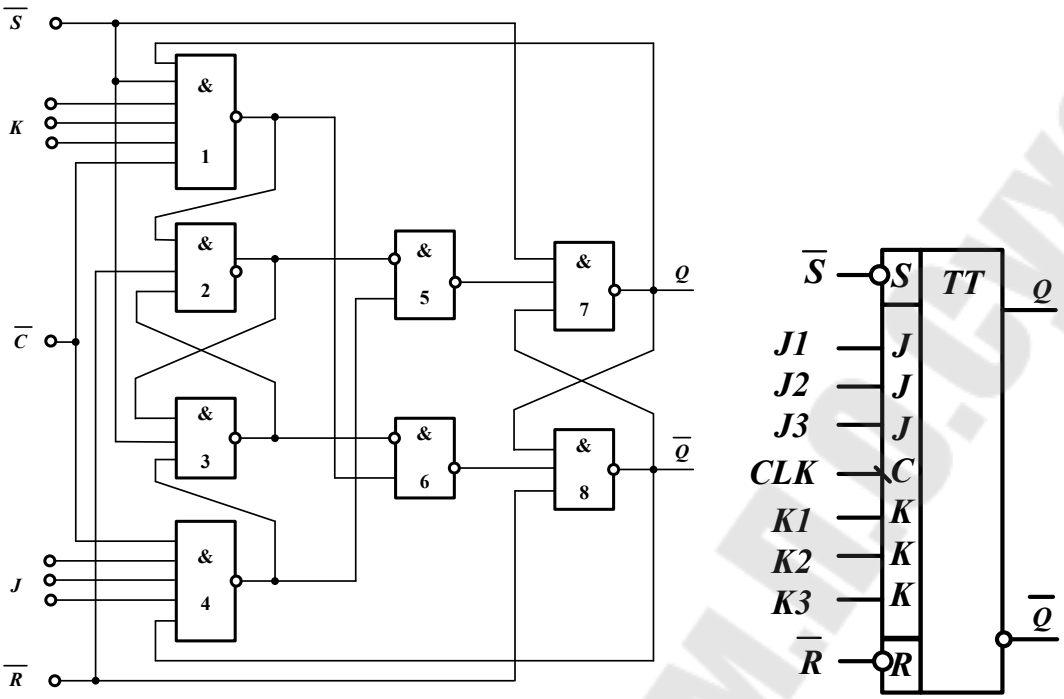
***JK*-триггер.** *JK*-триггер имеет два входа (*J* и *K*) и при выполнении условия $J \cdot K = 1$ осуществляет инверсию предыдущего состояния (т. е. $Q_{t+1} = \overline{Q_t}$), а в остальных случаях функционирует в соответствии с таблицей истинности *RS*-триггера, при этом вход *J* эквивалентен входу *S*, а вход *K* эквивалентен входу *R*.

Таблица 2.6

Таблица состояний *JK*-триггера

t		***t* + 1**
J	***K***	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	$\overline{Q_t}$

Логическое уравнение триггера, составленное на основе таблицы истинности, имеет вид: $Q_{t+1} = \overline{K} \cdot Q_t + J \cdot \overline{Q_t}$. На рис. 2.29 приведена схема и функциональное обозначение одного из первых, массово выпускаемых двухступенчатых триггеров *JK*-типа, который имеет логику 3И по входам *J* и *K* (К155ТВ1).



а) *б*)

Рис. 2.29. Пример синхронного *JK*-триггера (*а*) и его функциональное обозначение (*б*)

***T*-триггер.** *T*-триггер, или счетный триггер, имеет один вход *T*. Состояние триггера инвертируется всякий раз, когда на вход *T* поступает активный сигнал (0 или 1). Работа *T*-триггера может быть описана выражением $Q_{t+1} = Q_t \oplus T$ или таблицей состояний (табл. 2.7).

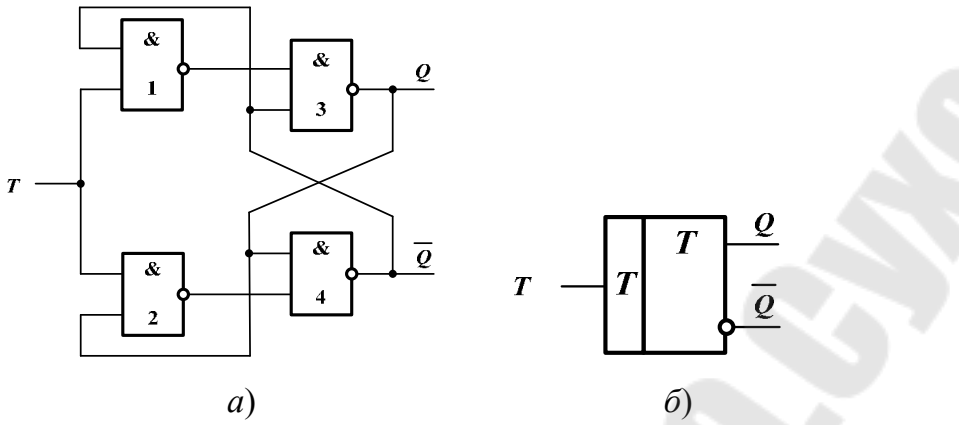
Таблица 2.7

Таблица состояний *T*-триггера

t	***t* + 1**
T	Q_{t+1}
0	Q_t
1	$\overline{Q_t}$

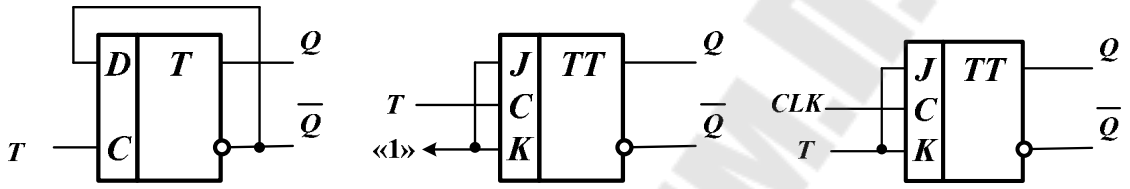
T-триггер реализуют, как правило, на основе ранее рассмотренных триггеров. На рис. 2.30 представлена его реализация на основе синхронного *RS*-триггера.

На рис. 2.31 представлена реализация асинхронного *T*-триггера на основе *D*-триггера (рис. 2.31, *а*), на основе *JK*-триггера (рис. 2.31, *б*) и реализация синхронного *T*-триггера на основе *JK*-триггера (рис. 2.31, *в*).



а) *б*)

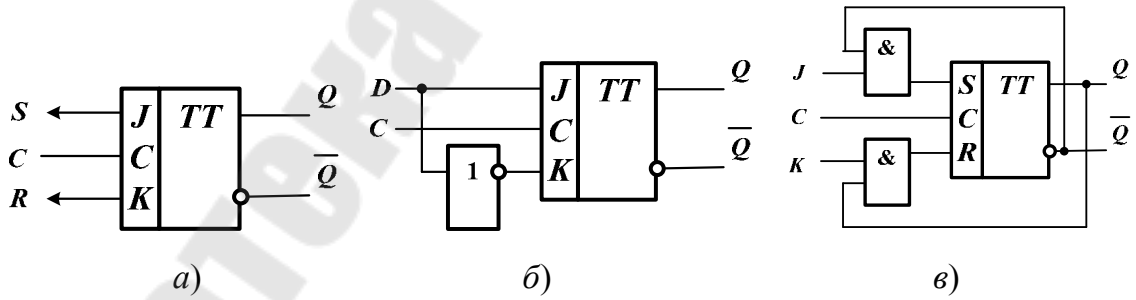
Рис. 2.30. Пример *T*-триггера (*а*) и его функциональное обозначение (*б*)



а) *б*) *в*)

Рис. 2.31. Реализация асинхронного *T*-триггера на основе *D*-триггера (*а*), асинхронного *T*-триггера на основе *JK*-триггера (*б*) и синхронного *T*-триггера на основе *JK*-триггера (*в*)

По аналогии, триггеры одного типа могут быть достаточно просто реализованы на триггерах другого типа. Примеры преобразования триггеров приведены на рис. 2.32.



а) *б*) *в*)

Рис. 2.32. Варианты реализации различных триггеров:
а – синхронного *RS*-триггера на основе *JK*-триггера;
б – синхронного *D*-триггера на основе *JK*-триггера;
в – синхронного *JK*-триггера на основе *RS*-триггера

Промышленностью выпускается достаточно широкая номенклатура триггеров. Приведем примеры наиболее распространенных интегральных схем.

JK-триггер: К155ТВ1 (многофункциональный триггер, технология ТТЛ), К555ТВ9 (два *JK*-триггера, технология ТТЛШ), К1533ТВ15 (два *JK*-триггера, технология ТТЛШ), К1564ТВ9 (два *JK*-триггера, технология КМОП), К561ТВ1 (два *JK*-триггера, технология КМОП).

D-триггер: К155ТМ2 (два *D*-триггера, технология ТТЛ), К1533ТМ9 (шесть *D*-триггеров с общим управлением, технология ТТЛШ), К555ТМ7 (четыре одноступенчатых *D*-триггера, технология ТТЛШ), К555ТМ8 (четыре двухступенчатых *D*-триггера, технология ТТЛШ), К561ТМ3 (четыре одноступенчатых *D*-триггера, технология КМОП).

RS-триггер: К555ТР2 (четыре одноступенчатых *RS*-триггера, технология ТТЛШ), К561ТР2 (четыре одноступенчатых *RS*-триггера, технология КМОП).

2.3.2. Регистры

Регистром называется последовательностное устройство, предназначенное для записи, хранения и передачи информации, представленной в виде многоразрядного двоичного кода. Соответственно, регистры должны содержать элементы памяти (триггера). Для выполнения конкретной операции в регистрах также используются некоторые вспомогательные элементы в виде комбинационных схем. В ЭВМ регистры используются для выполнения таких операций, как: временное хранение данных; ввод и вывод данных; сдвиг информации вправо либо влево на определенное число разрядов; преобразование кода числа из последовательного кода в параллельный и обратно; некоторые арифметические операции, такие, как умножение числа на 2, 4, 8, …, целочисленное деление на 2, 4, 8, …, вычисление остатка (*mod* 2) и т. п.

Основная функция регистра – хранение многоразрядного числа, которое представлено в двоичной системе счисления. Для реализации одного разряда используется, как правило, один триггер. Чаще всего регистры строят на основе *D*- и *JK*-триггера. Таким образом, для хранения *n*-разрядного двоичного числа регистр должен содержать *n* триггеров.

По способу приема информации регистры подразделяются:

– на параллельные (информация записывается/считывается одновременно во все разряды);

– на последовательные, которые также называются регистрами сдвига (запись и считывание информации происходит только в последовательной форме);

– на последовательно-параллельные, или универсальные, регистры (могут работать и как параллельные, и как последовательные регистры; как правило, используются для преобразования кода из параллельного в последовательный и обратно).

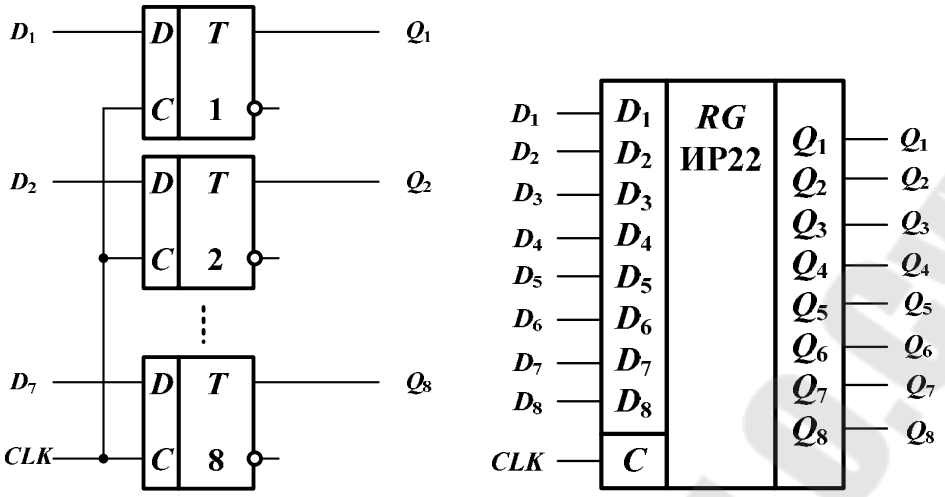
Сдвиговые регистры могут быть однонаправленные, когда записанную информацию сдвигают только в одном направлении, и реверсивные, в которых направление сдвига можно изменять (для таких регистров предусматривается специальный вход направления сдвига).

Различают однофазные и парафазные регистры. В однофазных регистрах информация записывается (считывается) только в одном виде – прямом или инверсном. В парафазных регистрах информация представлена в прямом и инверсном виде.

Кроме триггеров, регистры могут содержать некоторую комбинационную схему, определяющую функцию регистра и режим его работы. Например, схему перевода выходов регистра в третье состояние (для работы на магистраль), выходные буферные каскады (для умощнения сигнала), логику управления направлением сдвига, логику управления загрузкой данных, логику инициализации (установки фиксированного начального состояния), логику выполнения операций (арифметических, логических) и т. п.

Параллельные регистры. В параллельных регистрах запись информации (слова данных) осуществляется одновременно во все разряды регистра. Записанная информация может многократно считываться и храниться в нем сколь угодно длительное время (при наличии питающего напряжения), поэтому такие регистры называют регистрами памяти, или хранения.

Функциональная схема параллельного регистра, реализованного на триггерах-защелках, его функциональное обозначение, представлены на рис. 2.33. По сути он представляет собой восемь одноступенчатых *D*-триггеров, имеющих общую синхронизацию. Промышленностью выпускается ИС КР1533ИР22, которая имеет дополнительно вход перевода выходов регистра в третье состояние (на схеме не показан). Выпускается также ИС КР1533ИР23, отличием которой является использование двухступенчатых *D*-триггеров.

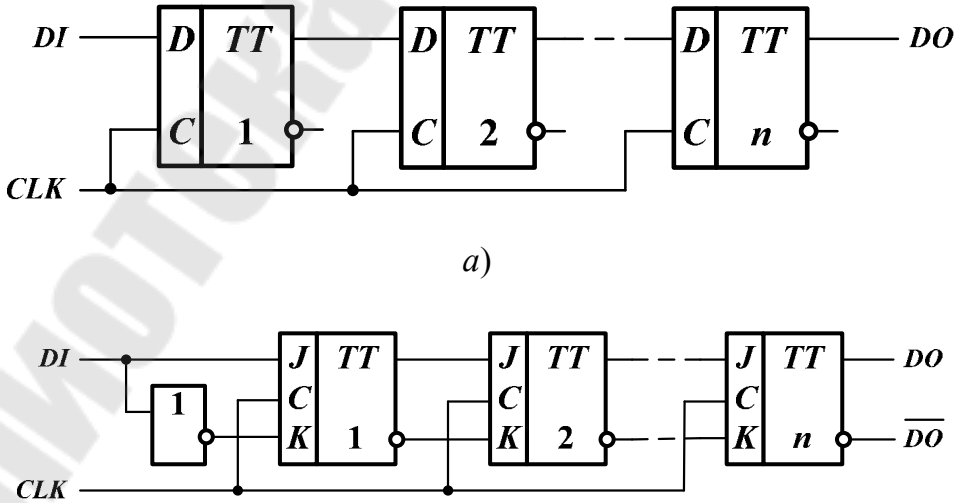


а) *б*)

Рис. 2.33. Параллельный регистр на одноступенчатых *D*-триггерах (*а*) и его функциональное обозначение (*б*)

Использование на выходах триггеров буферных элементов с тремя состояниями позволяет организовать поочередную работу нескольких регистров на общую линию связи (магистраль) либо осуществить мультиплексирование информации от нескольких источников.

Последовательные регистры, или регистры сдвига, осуществляют сдвиг информации вправо или влево (в сторону младших или старших разрядов). Они имеют вход и выход последовательных данных. Примеры реализации регистров сдвига на *D*-триггерах и *JK*-триггерах приведены на рис. 2.34.



а)

б)

Рис. 2.34. Регистр сдвига: *а* – на *D*-триггерах; *б* – на *JK*-триггерах

Данные поступают на вход первого (крайнего левого) триггера. После подачи *n* импульсов синхронизации (*n* – число разрядов регистра сдвига) эти данные появятся на выходе последнего (крайнего правого) триггера. Таким образом, регистры сдвига могут применяться в качестве элементов временной задержки цифровой информации, генераторов циклических кодов и кольцевых счетчиков.

Рассмотрим работу регистра сдвига более подробно. Пусть на вход *DI* поступает четырехразрядное число *A* = 1100 (начиная с младших разрядов). После прихода первого импульса синхронизации *CLK* в первый триггер будет записано значение 0 (рис. 2.35). После прихода второго импульса синхронизации значение первого триггера будет переписано во второй триггер, а в первый триггер будет записано новое значение. По аналогии после третьего синхроимпульса значение второго триггера запишется в третий, значение первого – во второй, а в первый триггер будет записан следующий разряд входного числа. После четвертого импульса синхронизации регистр будет содержать входное число 1100. Таким образом, запись информации в регистр произошла за четыре такта синхронизации. В общем случае, для *n*-разрядного регистра, запись информации осуществляется за *n* тактов.

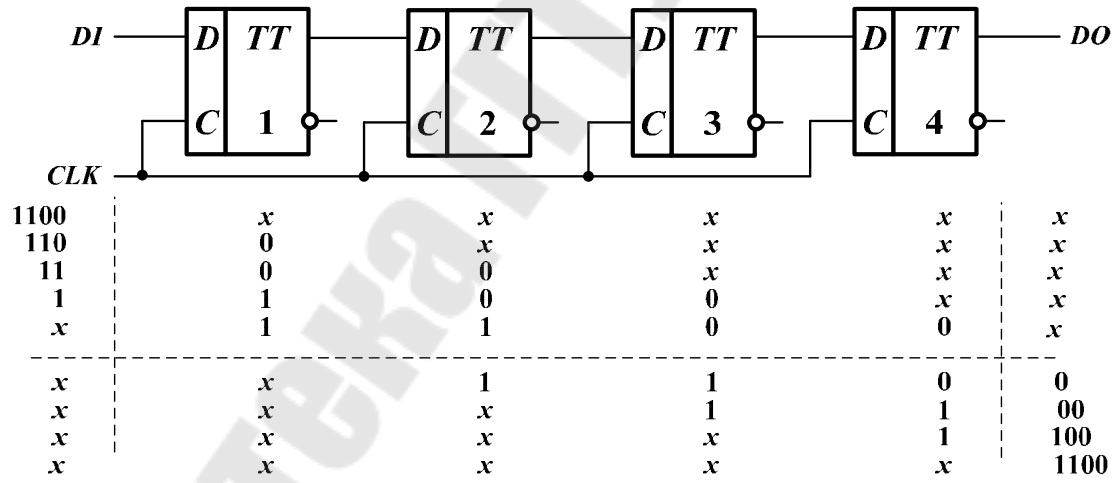


Рис. 2.35. Работа четырехразрядного регистра сдвига

Регистр, представленный на рис. 2.35, позволяет сдвигать информацию только в одну сторону – вправо. Для того чтобы получить возможность сдвига информации в обе стороны, используют реверсивные регистры. Пример одного разряда регистра приведен на рис. 2.36. По информационному входу триггера расположен мультиплексор «2 в 1», который в зависимости от управляющего сигна-

ла $L/\overline{R}$ определяет источник информации для данного разряда регистра сдвига. При $L/\overline{R}=0$ данные поступают с $i+1$-разряда (сдвиг влево). При $L/\overline{R}=1$ данные поступают с $i-1$-разряда (сдвиг вправо).

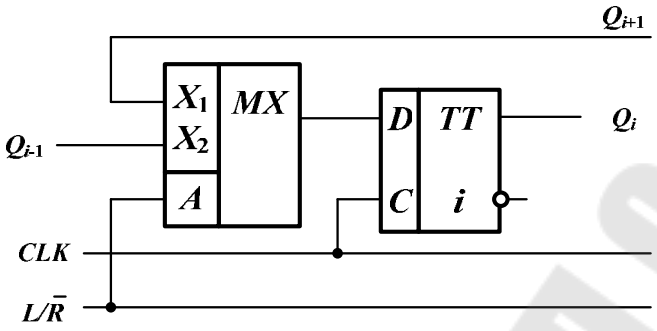
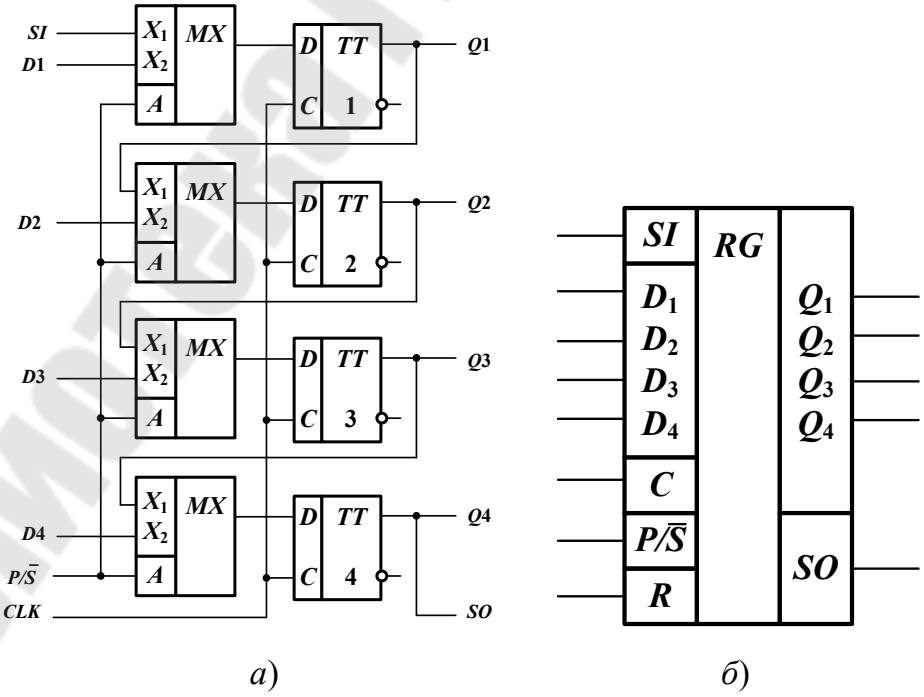


Рис. 2.36. Разряд реверсивного регистра сдвига

Последовательно-параллельные и параллельно-последовательные регистры представляют собой сочетания схем параллельного и последовательного регистров и служат для преобразования информации из последовательной формы в параллельную и обратно. Пример четырехразрядного последовательно-параллельного регистра, выполненного на *D*-триггерах, и его функциональное обозначение приведен на рис. 2.37. (На рис. 2.37, *б* добавлен вход установки триггеров в нулевое состояние, не показанный на рис. 2.37, *а*.)



а) *б*)

Рис. 2.37. Параллельно-последовательный регистр сдвига

В зависимости от управляющего сигнала $P/\overline{S}$ регистр может работать как параллельный ($P/\overline{S}=0$) или как последовательный ($P/\overline{S}=1$). На основе данного регистра легко реализовать преобразователь из параллельного кода в последовательный. Для этого необходимо записать в регистр параллельную информацию ($P/\overline{S}=0$, подать один тактовый импульс), а затем установить последовательный режим ($P/\overline{S}=1$) и подать четыре тактовых импульса. На выходе *SO* данная информация будет представлена в последовательном коде.

Рассмотрим типовые применения данных регистров (в дальнейшем будем называть их просто регистрами сдвига). На основе регистра сдвига легко реализовать делители частоты. Примеры делителей частоты с коэффициентом деления два, три и четыре приведены на рис. 2.38.

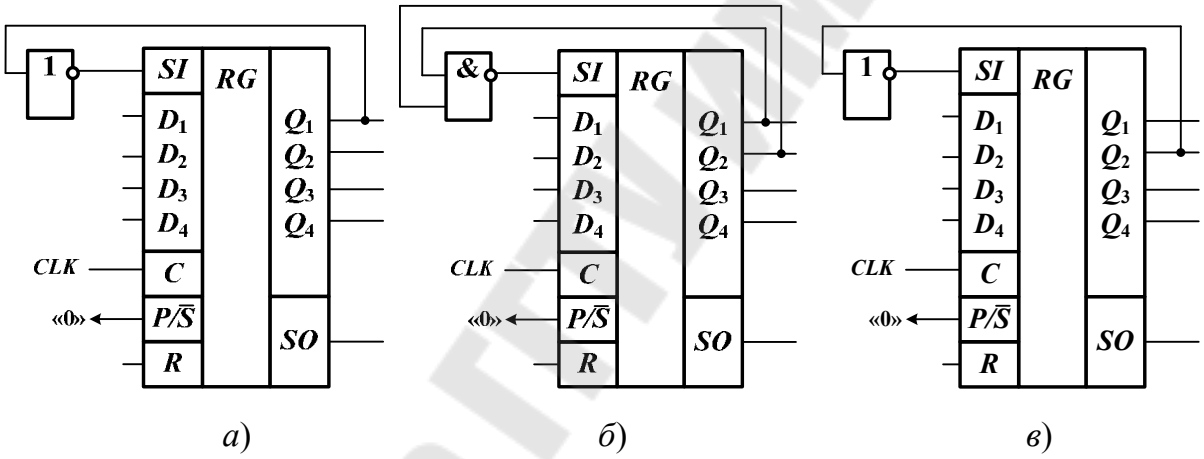


Рис. 2.38. Делители частоты на основе регистра сдвига:
а – делитель на два; *б* – делитель на три; *в* – делитель на четыре

Кольцевой счетчик. На основе регистра сдвига легко реализовать кольцевой счетчик. Основная идея кольцевого счетчика заключается в использовании обратной связи с выхода последнего триггера регистра сдвига на вход первого. При этом необходимо в начальный момент времени записать только одну единицу в какой-либо из разрядов кольцевого счетчика. Тогда при поступлении импульсов синхронизации *CLK* единственная 1 будет перемещается от разряда к разряду. Пример трехразрядного кольцевого счетчика приведен на рис. 2.39, а состояния триггеров – в табл. 2.8. В общем случае модуль пересчета кольцевого счетчика (число различных состояний) равен числу его разрядов.

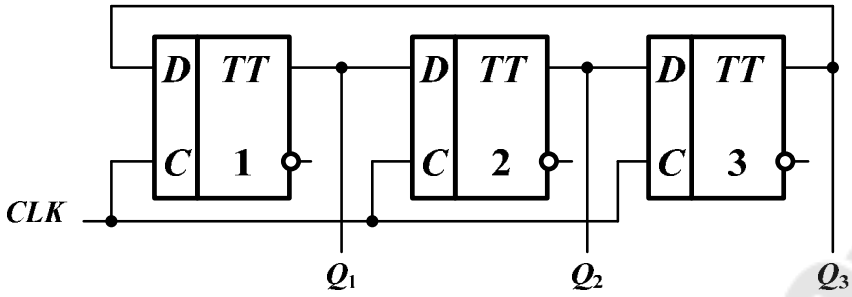


Рис. 2.39. Кольцевой счетчик на основе регистра сдвига

Таблица 2.8

Таблица состояния разрядов кольцевого счетчика

Номер	*Q*1	*Q*1	*Q*1
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0

Счетчик Джонсона – отличается от рассмотренного кольцевого счетчика тем, что в нем сигнал обратной связи на входы триггера первого (младшего) разряда сдвигового регистра подается не с прямого, а с инверсного выхода последнего триггера (старшего разряда). Поэтому если в исходном состоянии все триггеры регистра установлены в 0, то при поступлении импульсов синхронизации начнется их постепенное переключение в единицы. После того как все триггеры окажутся в единичном состоянии, начнется их поочередное (начиная с младшего разряда) переключение в 0 и т. д. Следовательно, число рабочих состояний такого счетчика равно удвоенному числу его разрядов. Пример трехразрядного кольцевого счетчика приведен на рис. 2.40, а состояния триггеров – в табл. 2.9. Недостатком рассмотренных счетчиков на основе регистра сдвига является малое число уникальных (различных) комбинаций.

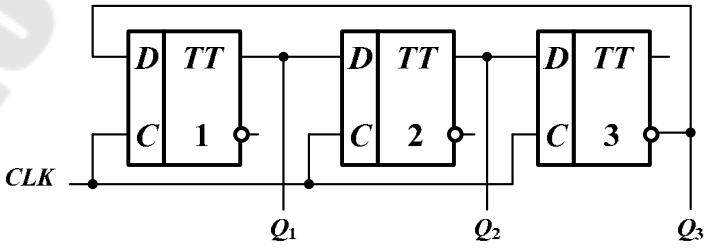


Рис. 2.40. Счетчик Джонсона

Таблица 2.9

Таблица состояния разрядов счетчика Джонсона

Номер	*Q*1	*Q*1	*Q*1
1	0	0	0
2	1	0	0
3	1	1	0
4	1	1	1
5	0	1	1
6	0	0	1
7	0	0	0

Сдвиговый регистр с линейной обратной связью, так называемый *LFSR* (*Linear Feedback Shift Register*), в отличие от кольцевого счетчика и счетчика Джонсона, позволяет получить гораздо больше уникальных комбинаций. Работа *LFSR* определяется характеристическим полиномом $\varphi(x)$, который определяет конфигурацию обратной связи. Пример *LFSR* с характеристическим полиномом $\varphi(x) = 1 \oplus x^3 \oplus x^4$ представлен на рис. 2.41, а состояния выходов триггеров представлено в табл. 2.10. Период формируемой последовательности (число различных комбинаций) зависит от свойств характеристического полинома. В рассмотренном примере период равен максимально возможному значению – 15, а формируемая последовательность является псевдослучайной последовательностью максимальной длины, или М-последовательностью. В общем случае, число уникальных комбинаций равно $2^n - 1$, где n – число разрядов регистра сдвига (более корректно говорить о старшей степени порождающего полинома, который должен быть примитивным и неприводимым).

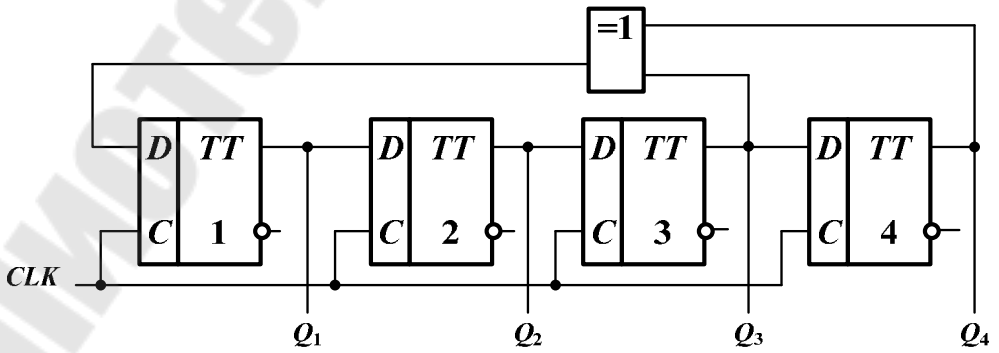


Рис. 2.41. Сдвиговый регистр с линейной обратной связью – *LFSR*

Таблица 2.10

Таблица состояний *LFSR*

Номер такта	Состояние				Номер такта	Состояние			
	***Q*1**	***Q*2**	***Q*3**	***Q*4**		***Q*1**	***Q*2**	***Q*3**	***Q*4**
1	1	0	0	0	9	1	0	1	0
2	0	1	0	0	10	1	1	0	1
3	0	0	1	0	11	1	1	1	0
4	1	0	0	1	12	1	1	1	1
5	1	1	0	0	13	0	1	1	1
6	0	1	1	0	14	0	0	1	1
7	1	0	1	1	15	0	0	0	1
8	0	1	0	1	16	1	0	0	0

В табл. 2.11 приведены все примитивные полиномы седьмой степени.

Таблица 2.11

Примитивные полиномы седьмой степени

Номер	Полином	Номер	Полином
1	$X^7 \oplus X \oplus 1$	10	$X^7 \oplus X^6 \oplus 1$
2	$X^7 \oplus X^5 \oplus X^3 \oplus X \oplus 1$	11	$X^7 \oplus X^6 \oplus X^4 \oplus X^2 \oplus 1$
3	$X^7 \oplus X^5 \oplus X^4 \oplus X^3 \oplus 1$	12	$X^7 \oplus X^4 \oplus X^3 \oplus X^2 \oplus 1$
4	$X^7 \oplus X^6 \oplus X^5 \oplus X^4 \oplus X^2 \oplus X \oplus 1$	13	$X^7 \oplus X^6 \oplus X^5 \oplus X^3 \oplus X^2 \oplus X \oplus 1$
5	$X^7 \oplus X^6 \oplus X^5 \oplus X^2 \oplus 1$	14	$X^7 \oplus X^5 \oplus X^2 \oplus X \oplus 1$
6	$X^7 \oplus X^4 \oplus 1$	15	$X^7 \oplus X^3 \oplus 1$
7	$X^7 \oplus X^6 \oplus X^5 \oplus X^4 \oplus 1$	16	$X^7 \oplus X^3 \oplus X^2 \oplus X \oplus 1$
8	$X^7 \oplus X^6 \oplus X^5 \oplus X^4 \oplus X^3 \oplus X^2 \oplus 1$	17	$X^7 \oplus X^5 \oplus X^4 \oplus X^3 \oplus X^2 \oplus X \oplus 1$
9	$X^7 \oplus X^6 \oplus X^3 \oplus X \oplus 1$	18	$X^7 \oplus X^6 \oplus X^4 \oplus X \oplus 1$

Для *LFSR* запрещенным состоянием является состояние «все нули». Поэтому число различных комбинаций, формируемых на его выходах, на единицу меньше, чем максимально возможное число – 2^n, где n – число разрядов регистра сдвига. Чтобы устранить этот недостаток, применяют генератор де Брейна (рис. 2.42), который искусственно вводит состояние «все нули». В примере состояние 0000 вводится после состояния 0001 при помощи дополнительных логических элементов – трехвходового элемента ИЛИ-НЕ и дополнительного элемента Исключающее ИЛИ. Таким образом, после состояния 0001 генератор переходит в состояние 0000, а затем в состояние 1000.

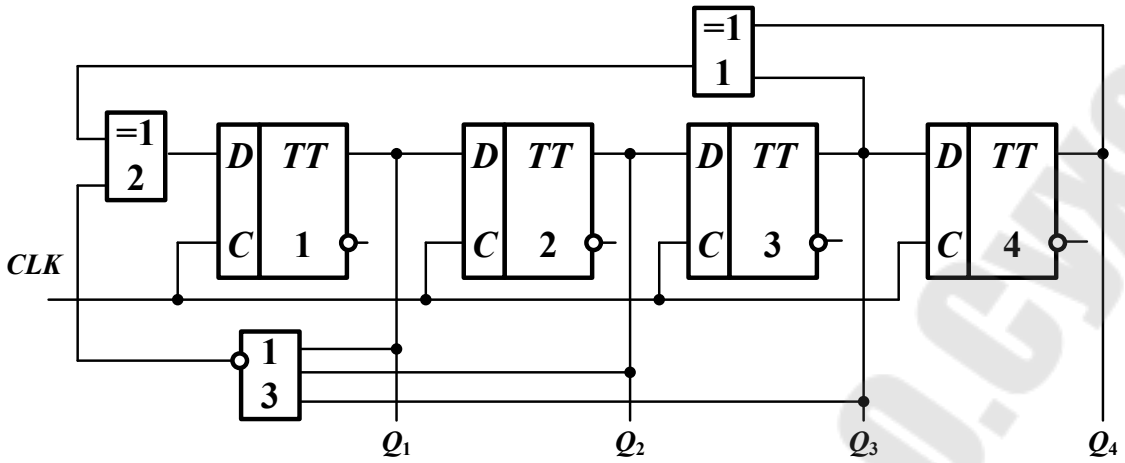


Рис. 2.42. Генератор де Брейна

2.3.3. Счетчики

Счетчик импульсов – это узел ЭВМ, обеспечивающий хранение слова информации и выполнение над ним микрооперации счета, заключающейся в изменении значения числа в счетчике на единицу. По существу счетчик представляет собой совокупность соединенных определенным образом триггеров. Основными характеристиками счетчика являются быстродействие и информационная емкость. Быстродействие счетчика определяется количеством изменений входного сигнала, фиксируемых счетчиком в единицу времени. Информационная емкость – максимальное число импульсов, которые могут быть подсчитаны счетчиком. Емкость счетчика определяется модулем счета *M*. Кроме подсчета импульсов, счетчики также используются для деления частоты. Как правило, счетчики реализуются на *T*- или *JK*-триггерах. Число триггеров определяет максимальное количество импульсов, которое может быть подсчитано счетчиком.

Счетчики могут быть классифицированы по следующим признакам:

1. По модулю счета:
   – двоичные;
   – двоично-десятичные;
   – с произвольным постоянным модулем счета;
   – с переменным модулем счета.
2. По направлению счета:
   – суммирующие;
   – вычитающие;
   – реверсивные.

3. По способу формирования внутренних связей:
   – с последовательным переносом;
   – с параллельным переносом;
   – с комбинированным переносом;
   – кольцевые.
4. По способу кодирования внутренних состояний:
   – двоичные счетчики;
   – счетчики Джонсона;
   – счетчики с кодом «1 из *N*»;
   – счетчики в коде Грея.

Суммирующий счетчик предназначен для выполнения счета в прямом направлении, т. е. для сложения. С приходом очередного счетного импульса на вход счетчика его показание увеличивается на единицу (инкремент текущего значения).

Вычитающий счетчик предназначен для выполнения счета в обратном направлении, т. е. в режиме вычитания. Каждый счетный импульс, поступивший на вход счетчика, уменьшает его показания на единицу (декремент текущего значения).

Реверсивными называются такие счетчики, которые предназначены для выполнения счета как в прямом, так и в обратном направлении, т. е. могут работать в режиме инкремента и декремента текущего значения.

Обычно счетчик имеет цепь установки в ноль. Однако данное требование не является обязательным. Начальное состояние может устанавливаться, например, параллельной загрузкой в счетчик некоторого числа. Соответственно, после прихода очередного импульса синхронизации происходит инкремент (декремент) текущего значения. Такой режим характерен, например, при реализации программного счетчика в микропроцессорах, которые формируют адрес команды в памяти программ. После загрузки заданного адреса программы идет последовательная выборка команд, начиная с этого адреса.

Наибольшее распространение получили двоичные счетчики, на основании которых можно строить другие типы счетчиков.

Двоичные счетчики. Работу двоичного счетчика рассмотрим на примере четырехразрядного двоичного счетчика на двухступенчатых счетных триггерах, реализованных на основе *D*-триггера. Этот счетчик имеет один счетный вход *CLK* и четыре выхода *Q*1–*Q*4 (рис. 2.43). Состояния разрядов счетчика показано в табл. 2.12.

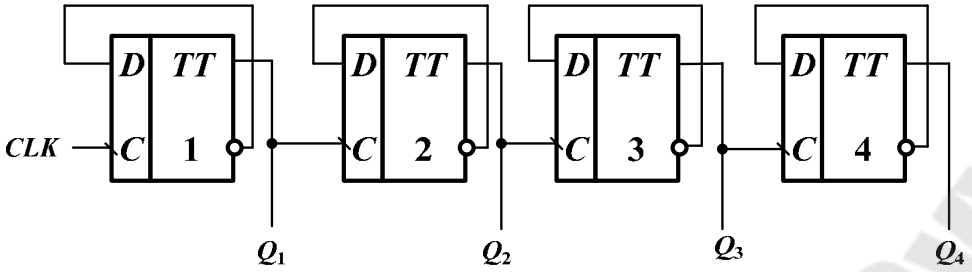


Рис. 2.43. Двоичный счетчик с последовательным переносом

Таблица 2.12

Таблица состояние разрядов двоичного счетчика

Номер	***Q*4**	***Q*3**	***Q*2**	***Q*1**	**Номер**	***Q*4**	***Q*3**	***Q*2**	***Q*1**
0	0	0	0	0	9	1	0	0	1
1	0	0	0	1	10	1	0	1	0
2	0	0	1	0	11	1	0	1	1
3	0	0	1	1	12	1	1	0	0
4	0	1	0	0	13	1	1	0	1
5	0	1	0	1	14	1	1	1	0
6	0	1	1	0	15	1	1	1	1
7	0	1	1	1	16	0	0	0	0
8	1	0	0	0	–	–	–	–	–

Триггеры срабатывают по спаду входного импульса (при переходе из 1 в 0). Четыре последовательно соединенных триггера образуют счетчик по модулю $2^4 = 16$. Максимально возможное число, хранимое в счетчике, равно $2^4 - 1 = 15$. Пусть в исходном состоянии на всех триггерах установлены логические нули. Каждый триггер меняет свое состояние лишь в тот момент, когда на него действует отрицательный перепад напряжения. Поэтому после прихода первого импульса синхронизации в единичное состояние переключится только первый триггер. После прихода второго импульса синхронизации первый триггер опять переключится, уже в состояние 0. На его выходе будет сформирован отрицательный фронт, который вызовет переключение второго триггера. Таким образом, данный счетчик реализует суммирование входных импульсов.

Данный счетчик является асинхронным, так как все триггера переключаются в различное время. Например, третий триггер может переключиться только после того, как переключится второй триггер. В свою очередь, второй триггер может переключиться только после того, как переключится первый триггер. Таким образом, длительность переходного процесса в счетчике зависит от его разрядности. С ростом

разрядности счетчика понижается предельная частота его работы. Это объясняется тем, что с ростом разрядности счетчика n будет возрастать задержка поступления сигнала на вход с некоторого j-го разряда относительно времени поступления входного сигнала f_{cr} на вход с младшего разряда счетчика. Такая задержка может привести к искажению информации в счетчике. Для повышения быстродействия используют синхронные счетчики с параллельным переносом (рис. 2.44).

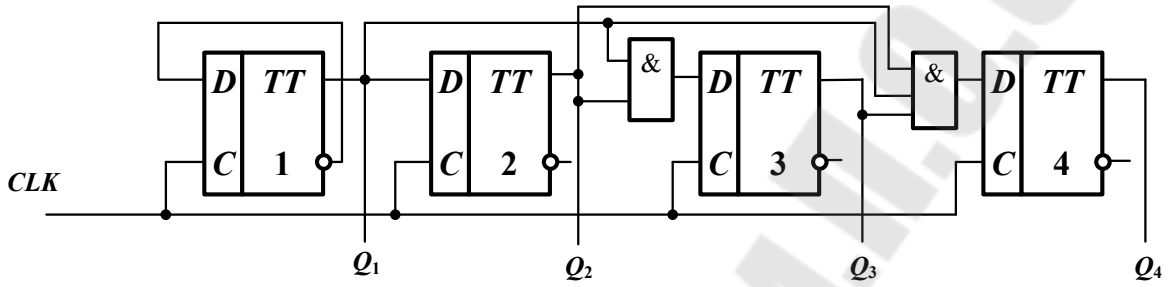


Рис. 2.44. Синхронный двоичный счетчик с параллельным переносом

Отличительной особенностью счетчиков с параллельным переносом является то, что выходы всех предшествующих каскадов (разрядов) подаются на информационные входы D последующих триггеров. Длительность переходного процесса в таком счетчике равна длительности переключения одного разряда. Из схемы видно, что с возрастанием порядкового номера триггера увеличивается число входов в элементах и D-триггеров. А так как нагрузочная способность выходов триггеров ограничена, то и разрядность счетчика с параллельным переносом невелика и обычно равна четырем. Поэтому при большем числе разрядов счетчика его разбивают на группы и внутри каждой группы строят цепи параллельного переноса.

Такой подход удобен и потому, что счетчик часто реализуют в виде ИС в отдельном корпусе. В этом случае при последовательном переносе просто осуществляется увеличение разрядности счетчика.

3. ПАМЯТЬ ЦИФРОВЫХ УСТРОЙСТВ

3.1. Классификация

В настоящее время существует много разнообразных устройств, которые позволяют хранить информацию и выполнять с ней такие операции, как чтение и запись. Это такие устройства, как регистры, оперативная память, кэш-память, дисковые накопители, флэш-память, постоянные запоминающие устройства, магнитные карточки и т. п. Такие устройства называются *RWM* (*Read-Write Memory*) [11]. Мы будем рассматривать только полупроводниковую память, представленную в виде отдельной микросхемы (кристалла) или являющуюся частью СБИС – так называемую встроенную память (*embedded memory*). Доля полупроводниковой памяти в общем объеме выпуска ИС составляет около 40 %. К основным характеристикам памяти относится: *информационная емкость* – максимально возможный объем хранимой информации; *организация* – число хранимых слов и их разрядность, иногда указывают количество банков памяти; *число портов ввода-вывода* данных (одно-, двух- или многопортовая память) и их реализация (двунаправленные или раздельные линии на ввод и вывод данных); *быстродействие* – оценивается временем чтения, записи или длительностью цикла чтения-записи.

По способу хранения информации различают оперативную и постоянную память (рис. 3.1). Оперативная память (ОП), или оперативное запоминающее устройство (ОЗУ), хранит информацию только при наличии напряжения питания. При пропадании напряжения питания информация разрушается (теряется). Постоянная память, или постоянное запоминающее устройство (ПЗУ), хранит информацию независимо от наличия или отсутствия напряжения питания.

Оперативная память (ОП), или оперативное запоминающее устройство (ОЗУ), хранит информацию только при наличии напряжения питания. При пропадании напряжения питания информация разрушается (теряется). Постоянная память, или постоянное запоминающее устройство (ПЗУ), хранит информацию независимо от наличия или отсутствия напряжения питания.

Полупроводниковую память условно можно разделить на память с последовательным доступом, произвольным доступом, ассоциативную и энергонезависимую. Различие между типами памяти представлено на рис. 3.2.

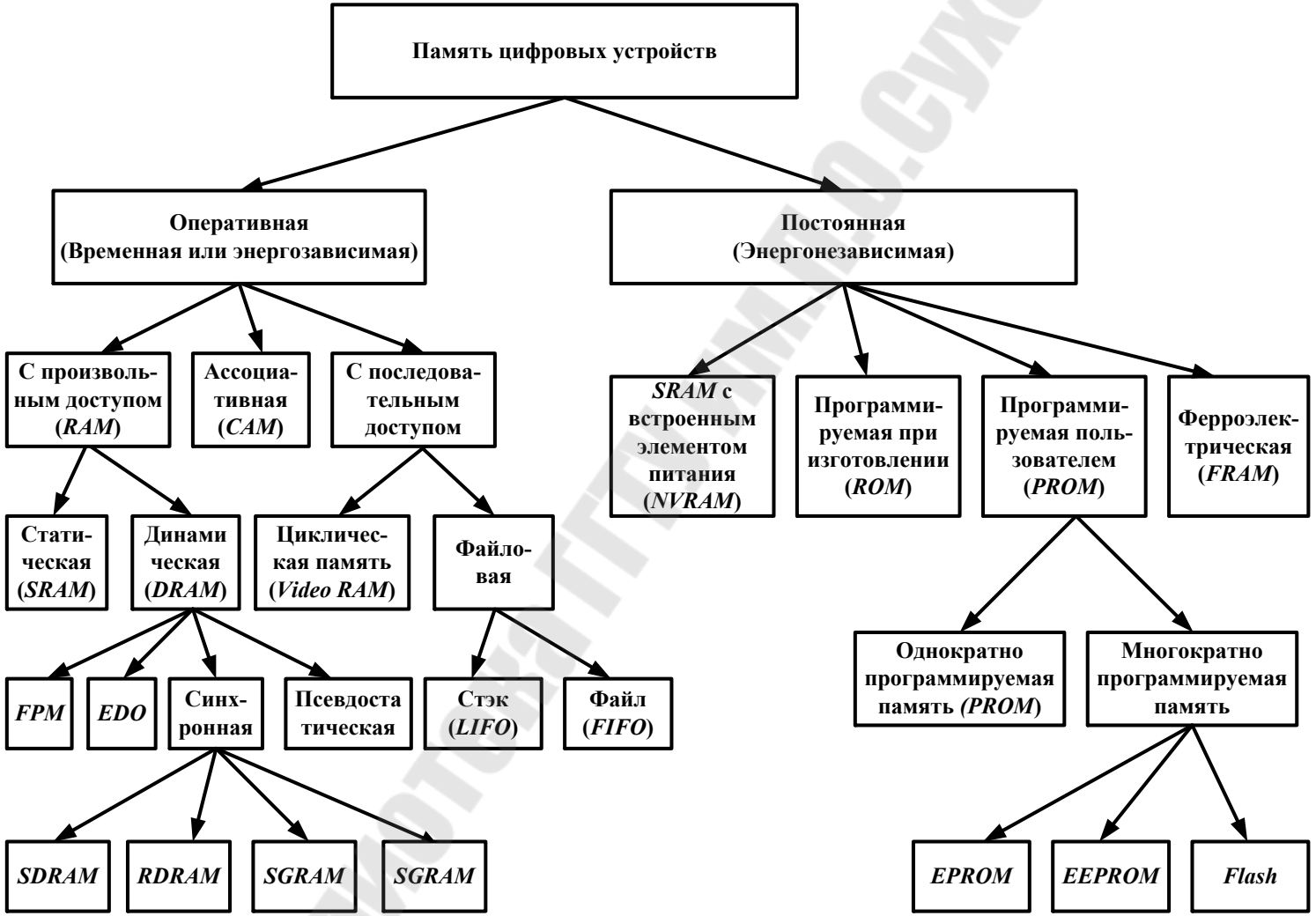


Рис. 3.1. Классификация полупроводниковой памяти

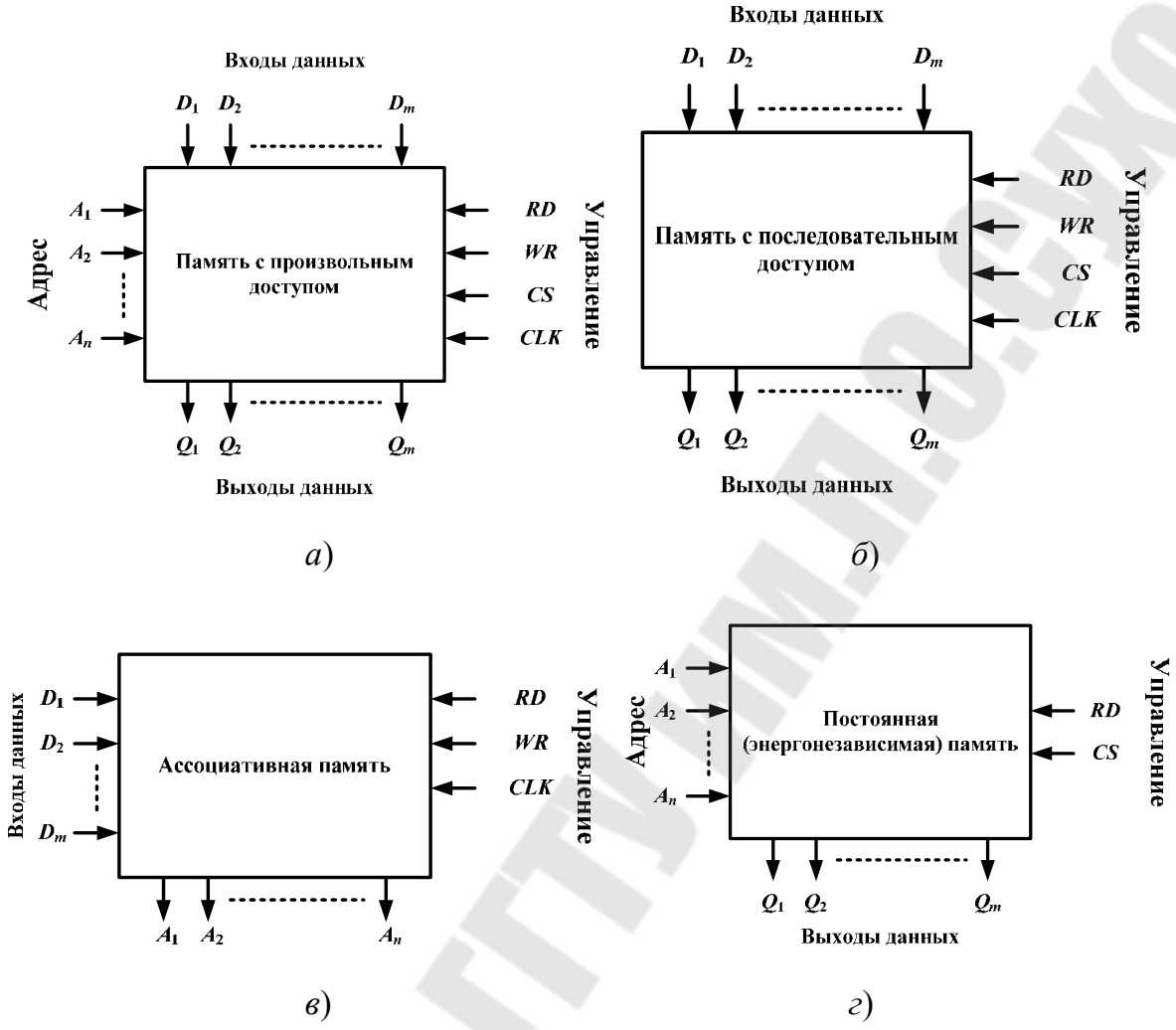


Рис. 3.2. Типы оперативной памяти: *а* – память с произвольным доступом; *б* – память с последовательным доступом; *в* – ассоциативная память; *г* – энергонезависимая память

Память с произвольным доступом *RAM* (*Random Access Memory*) позволяет обращаться (т. е. выполнять операции чтения-записи) с произвольной ячейкой. Рассмотрим работу памяти без учета ее внутренней структуры на примере (рис. 3.2, *а*). Память имеет m входов данных D_1–D_m, m выходов данных Q_1–Q_m, n адресных входов A_1–A_n. Данная память позволяет хранить 2^n слов. Разрядность одного слова составляет m бит. Таким образом, информационная емкость памяти составляет $m2^n$ бит. В настоящее время выпускают память с разрядностью x 1, x 4, x 8, x 16, x 32. Поэтому память одной и той же емкости может быть представлена различными способами. Например, память емкостью 256 Мбит может быть представлена в виде 256M x 1, 64M x 4, 32M x 8, 16M x 16, 8M x 32.

Можно выделить три основных режима работы – запись данных, чтение данных и хранение данных. Режимы работы памяти определяются четырьмя сигналами управления – *RD*, *WR*, *CS* и *CLK* (рис. 3.3). Сигналы *RD* и *WR* определяют режим работы памяти – чтение данных или запись данных соответственно. Сигнал *CS* показывает, что обмен информацией (чтение или запись данных) происходит с данным кристаллом памяти (этот сигнал используется, как правило, при наличии в системе нескольких кристаллов памяти). Сигнал *CLK* используется, как правило, в динамической памяти или для привязки операций обмена данными к системным синхроимпульсам.

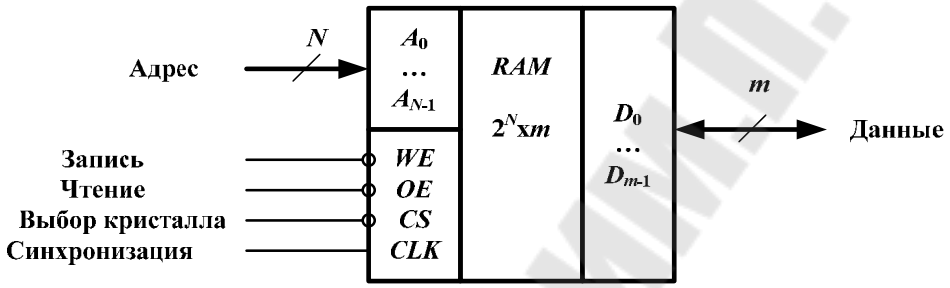


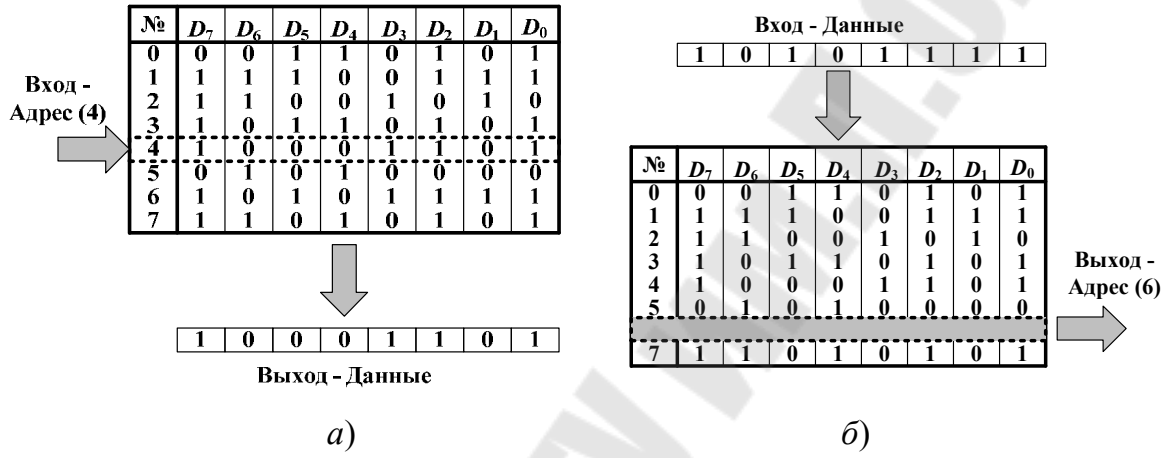
Рис. 3.3. Условное графическое обозначение ОЗУ

Отличительной особенностью памяти с последовательным доступом является то, что она не имеет адресных входов (рис. 3.2, *б*). Поэтому для доступа к информации необходимо, как правило, прочитать все ее содержимое. Примерами такой памяти могут служить регистры сдвига, регистровые файлы с организацией *LIFO* (*Last In First Out*) или *FIFO* (*First In First Out*), линии задержки, видеопамять (*VRAM*) и т. п.

В ассоциативных ЗУ (АЗУ или *CAM – Content Addressable Memory*) запись (чтение) производится не по конкретному адресу, а по заданному сочетанию (ассоциации) признаков, свойственных искомой информации (рис. 3.2, *в*). Такими признаками могут быть: часть слова (числа), приданная ему для обнаружения среди других слов, некоторые особенности самого слова (например, наличие определенных кодов в его разрядах), абсолютная величина слова, нахождение его в заданных пределах и т. д. На рис. 3.4 показано различие между *RAM* и *CAM*.

По способу хранения информации различают статическую и динамическую оперативную память. В статической – *SRAM* (*Static Random Access Memory*) информация после записи хранится без изменений до тех пор, пока не будет выполнена новая запись или не будет выключено напряжение питания. В динамической – *DRAM* (*Dynamic Random Access Memory*) информация сохраняется после записи только

определенное время. Поэтому для надежного хранения требуется периодическое обновление (перезапись) информации, или регенерация. В зависимости от применяемой технологии изготовления и организации памяти временной интервал между циклами регенерации составляет от двух до 256 миллисекунд. Во время регенерации (которое составляет от 1 до 5 % общего времени работы) доступ к данным невозможен или ограничен, поэтому использование *DRAM* более сложно, чем *SRAM*.



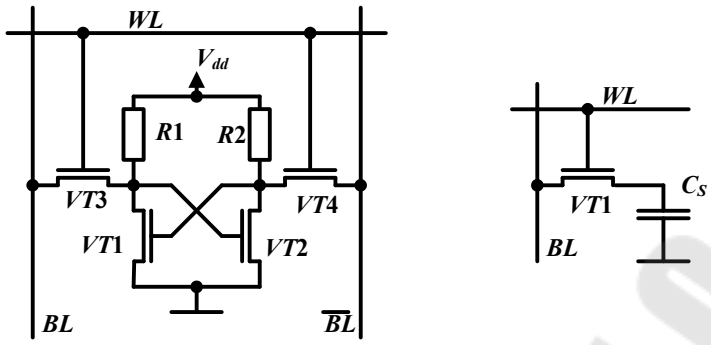
а) *б*)

Рис. 3.4. Принцип работы *RAM* (*а*) и *CAM* (*б*)

Основное различие между *SRAM* и *DRAM* заключается в схемотехнике запоминающего элемента. В *SRAM* запоминающим элементом является статический триггер (рис. 3.5, *а*). В *DRAM* информация хранится в виде заряда конденсатора (рис. 3.5, *б*). Минимальная реализация статического триггера требует 4 транзистора, тогда как для ячейки *DRAM* требуется только один. Поэтому *DRAM* имеет более высокую плотность компоновки ячеек, хотя для их формирования требуется специальный технологический процесс, который позволяет минимизировать токи утечки конденсаторов. Динамическая память характеризуется большей информационной емкостью и невысокой стоимостью, но имеет большее энергопотребление и меньшее быстродействие. Статическая память, выполненная по КМОП технологии, при хранении практически не потребляет энергию, тогда как для динамической требуется регенерация.

Постоянная, или энергонезависимая, память хранит информацию независимо от того, подключена ли она к источнику питания или нет. Она предназначена только для чтения информации, поэтому вхо-

ды данных на рис. 3.2, *г* отсутствуют. Энергонезависимая память в зависимости от принципа работы имеет разновидности.

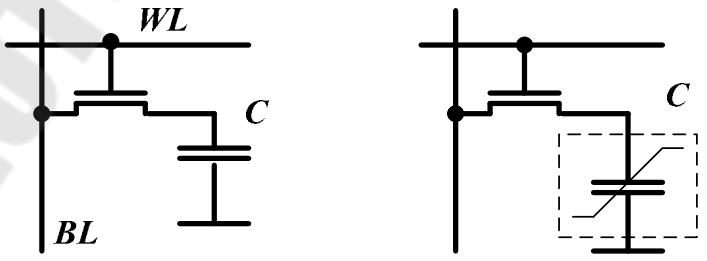


а) *б*)

Рис. 3.5. Схемотехника простейших запоминающих элементов *RAM*: *а* – статической памяти; *б* – динамической памяти

Сегнетоэлектрическая память *FRAM* (*Ferroelectric RAM*) – это запоминающее устройство, которое использует сегнетоэлектрический эффект для запоминания информации. Причем информация сохраняется после отключения питания. Сегнетоэлектрический эффект – это свойство материала сохранять электрическую поляризацию в отсутствие внешнего электрического поля.

Ячейка памяти *FRAM* образовывается из тонкой пленки кристаллического сегнетоэлектрического материала, размещенного между металлическими проводниками. По конструктивному и схемотехническому исполнению ячейка памяти очень похожа на базовую ячейку *DRAM* (рис. 3.6). Однако информация в ней хранится не в виде заряда на конденсаторе, как в *DRAM*, а в виде эффекта поляризации кристалла. Соответственно, отсутствуют токи утечки заряда, следовательно, не требуется регенерация.



а) *б*)

Рис. 3.6. Базовые элементы: *а – DRAM*; *б – FRAM*

Молекула сегнетоэлектрического кристалла содержит подвижный атом, который имеет два стабильных положения внутри кристалла. Под воздействием внешнего электрического поля этот атом перемещается в направлении приложения поля. После снятия электрического поля атом остается в стабильном состоянии. Для того чтобы определить доминирующее положение подвижных атомов в кристалле, необходимо к кристаллу приложить электрическое поле (через обкладки конденсатора). При этом будет происходить перемещение свободных атомов, либо нет. В случае, когда происходит перемещение подвижных атомов, генерируется гораздо больший заряд, чем в случае, когда они остаются на месте. Чувствительный усилитель считывает величину заряда и сравнивает с эталонным значением. По результатам сравнения формируется логическое значение.

По аналогии с *DRAM*, чтение информации является разрушающим, поэтому содержимое памяти после чтения должно быть восстановлено (регенерировано). Следовательно, в общем случае операция чтения включает в себя три фазы – предвыборка, чтение и запись (регенерацию). Операция записи происходит быстрее, так как не требуется чтение содержимого памяти. Например, для микросхемы *FM*3808 (энергонезависимое ПЗУ 256 Кбит с организацией 32К х 8) компании *RAMTRON* время доступа составляет 70 нс, время цикла – 130 нс. При этом число циклов чтения-записи составляет 10^{11} раз. Длительность хранения информации составляет 10 лет. Достоинством памяти на сегнетоэлектрическом эффекте является то, что она не подвержена воздействию магнитных полей и радиации.

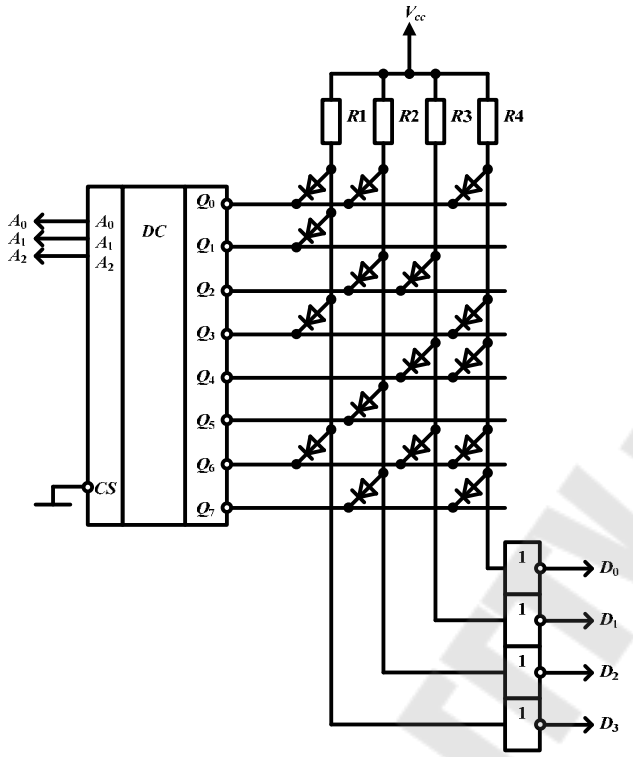
ПЗУ, или *ROM* (*Read Only Memory*), в широком смысле не является памятью. Оно является чисто комбинационной, а не последовательностной схемой. Оно хранит таблицу истинности комбинационной логической схемы с n входами и m выходами (рис. 3.2, *г*). Однако если обозначить входы A_0, A_1, ..., A_n, то получим m-разрядную память с произвольным доступом, которая работает только в режиме чтения данных (рис. 3.7).

ПЗУ может быть реализовано различными способами:

1) ПЗУ, программируемые при изготовлении с помощью одной из масок (ROM). Это масочные ПЗУ. Информация в ПЗУ записывается на стадии изготовления микросхемы и не может быть изменена в процессе ее эксплуатации.

2) ПЗУ, программируемые пользователем (ППЗУ). Среди них можно выделить однократно программируемые ПЗУ, информация в ко-

торые записывается пользователем при помощи специального устройства – программатора, и многократно программируемые ПЗУ, которые позволяют производить изменение содержимого ПЗУ в процессе эксплуатации. К многократно программируемым ПЗУ относятся *EPROM* (*erasable programmable read-only memory*), *EEPROM* (*Electrically Erasable Programmable Read-Only Memory*) и флэш-память (*Flash*).



а)

A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	1	1	0	1
0	0	1	1	0	0	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	0
1	1	0	1	0	1	1
1	1	1	0	1	0	1

б)

Рис. 3.7. Пример простейшего ПЗУ:
а – схема; *б* – хранящаяся информация

В *EPROM* стирание происходит путем облучения кристалла ультрафиолетовыми лучами (РПЗУ-УФ – репрограммируемые ПЗУ с УФ стиранием). В *EEPROM* стирание происходит электрическими сигналами (РПЗУ-ЭС – репрограммируемые ПЗУ с электрическим стиранием). Запись данных для *EPROM* и *EEPROM* производится электрическими сигналами. *Flash* память отличается от *EEPROM* тем, что запись и стирание информации происходят блоками (секторами).

На рис. 3.8 представлена схемотехника ячеек *EPROM* (рис. 3.8, *а*), *Flash* (рис. 3.8, *б*) и *EEPROM* (рис. 3.8, *в*).

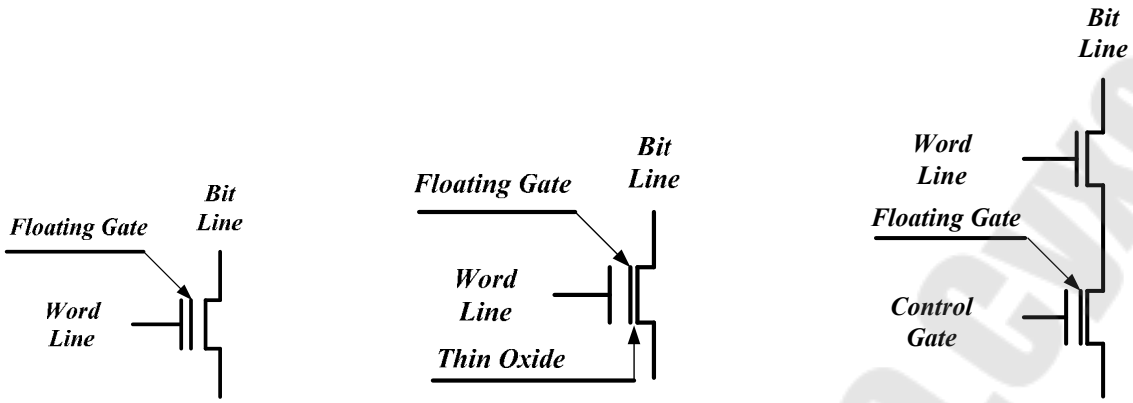


Рис. 3.8. Схемотехника ячеек программируемых постоянных запоминающих устройств: *а – EPROM*; *б – Flash*; *в – EEPROM*

NVRAM (*Non Volatile RAM*) – представляет собой комбинацию ОЗУ и ПЗУ. При выключении питания информация из ОЗУ переписывается в ПЗУ. При включении питания происходит обратная перезапись. Таким образом достигается высокое быстродействие статической памяти при работе и длительное хранение информации при отключении питания. Разновидностью этой памяти является так называемая *BRAM* (*Battery RAM*), которая состоит из микропотребляющей *RAM* с автономным источником питания (рис 3.9). При работе она питается от системного источника питания и функционирует как обыкновенная статическая память. При выключении (пропадании) системного напряжения она переходит на питание от автономного источника питания, в качестве которого используется, как правило, литиевая батарейка. Время хранения информации в такой памяти может достигать десяти лет.

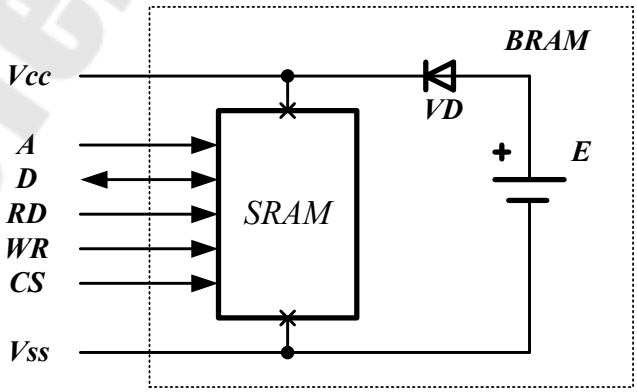


Рис. 3.9. Принцип работы *BRAM*

Рынок памяти составляет почти одну треть от общего объема выпуска интегральных схем. Кроме отдельных интегральных схем и модулей, память широко используется в составе современных процессоров и «Систем на Кристалле» (*SoC – System-on-a-Chip*). Например, процессор *McKinley*, изготавливаемый по 0,18 мкм процессу *Intel*, содержит почти 220 млн транзисторов. Примерно 75 % транзисторов используются для реализации 13 различных блоков памяти (*RAM* и *CAM*). Он имеет 8-уровневый конвейер и 3 уровня кэша на кристалле (*L*3 – 3 Мб, *L*2 – 256 Кб, *L*1 – 16 Кб + 16 Кб).

3.2. Статические оперативные запоминающие устройства

Отличительной особенностью статических ОЗУ (СОЗУ или *SRAM*) является то, что данные могут быть считаны (записаны) практически сразу после изменения адреса, тогда как для других видов ОЗУ требуется некоторая последовательность управляющих импульсов. Основу СОЗУ составляет матрица запоминающих элементов (ЗЭ), в качестве которых используется *RS*-триггер. Организация матрицы определяет способ выборки конкретных запоминающих элементов при операциях с ОЗУ. Количество ЗЭ определяет информационную емкость памяти, которая измеряется в битах (килобитах – Кбит, мегабитах – Мбит, гигабитах – Гбит, причем 1 Кбит = 1024 бит, 1 Мбит = 1024 Кбит = 1 048 576 бит, 1 Гбит = 1024 Мбит = 1 073 741 824 бит). Число ЗЭ, которые могут быть записаны (прочитаны) одновременно, определяет разрядность памяти или разрядность слова памяти (n_d). Число хранимых слов определяет количество линий адреса (n_a).

Рассмотрим простой пример. Пусть имеется память емкостью 1 Мбит. Она может быть представлена как:

– 1 048 576 одноразрядных слов (1М x 1) – $n_a = 20$, $n_d = 1$;

– 262144 четырехразрядных слова (256К x 4) – $n_a = 18$, $n_d = 4$;

– 131072 восьмиразрядных слова (128К x 8) – $n_a = 17$, $n_d = 8$;

– 65536 шестнадцатиразрядных слов (64К x 16) – $n_a = 16$, $n_d = 16$.

Таким образом, в первом случае для обращения к памяти требуется 21 линия, а в последнем – 32 (при этом не учитываются сигналы управления). Поэтому чем больше разрядность слова памяти, тем больше места на кристалле требуется для разводки линий связи и, как следствие, большая емкость линий связи. Это приводит к возрастанию времени обращения к памяти и увеличению потребления энергии. С другой стороны, многоразрядная память позволяет за одну

операцию прочитать (записать) сразу несколько бит информации, что повышает производительность вычислительной системы. Поэтому в настоящее время выпускаются микросхемы памяти с разрядностью слова памяти $n_d = 4$, 8(9), 16(18), 32. В скобках указана разрядность слова памяти при использовании контроля на четность.

На рис. 3.10 представлена упрощенная схема одноразрядной статической оперативной памяти, которая имеет $n_a = 4$ адресных входов и один разряд данных ($n_d = 1$), представленный входом данных *D* и выходом данных *Q*. Основу памяти составляет массив запоминающих элементов (ЗЭ), организованный в виде матрицы из четырех строк и четырех столбцов. Выбор конкретной ячейки памяти осуществляется двумя сигналами – линией выборки слова *WL* (*Word Line*) и разрядной шиной, представленной парафазными сигналами *BL* и $\overline{BL}$ (*Bit Line*). В каждый конкретный момент времени может быть активна только одна строка матрицы (сигналы x_1–x_4) и только один столбец матрицы (сигналы y_1–y_4). Выбор строки и столбца матрицы осуществляют дешифраторы строк (*DCA_ROW*) и столбцов (*DCA_COL*) соответственно.

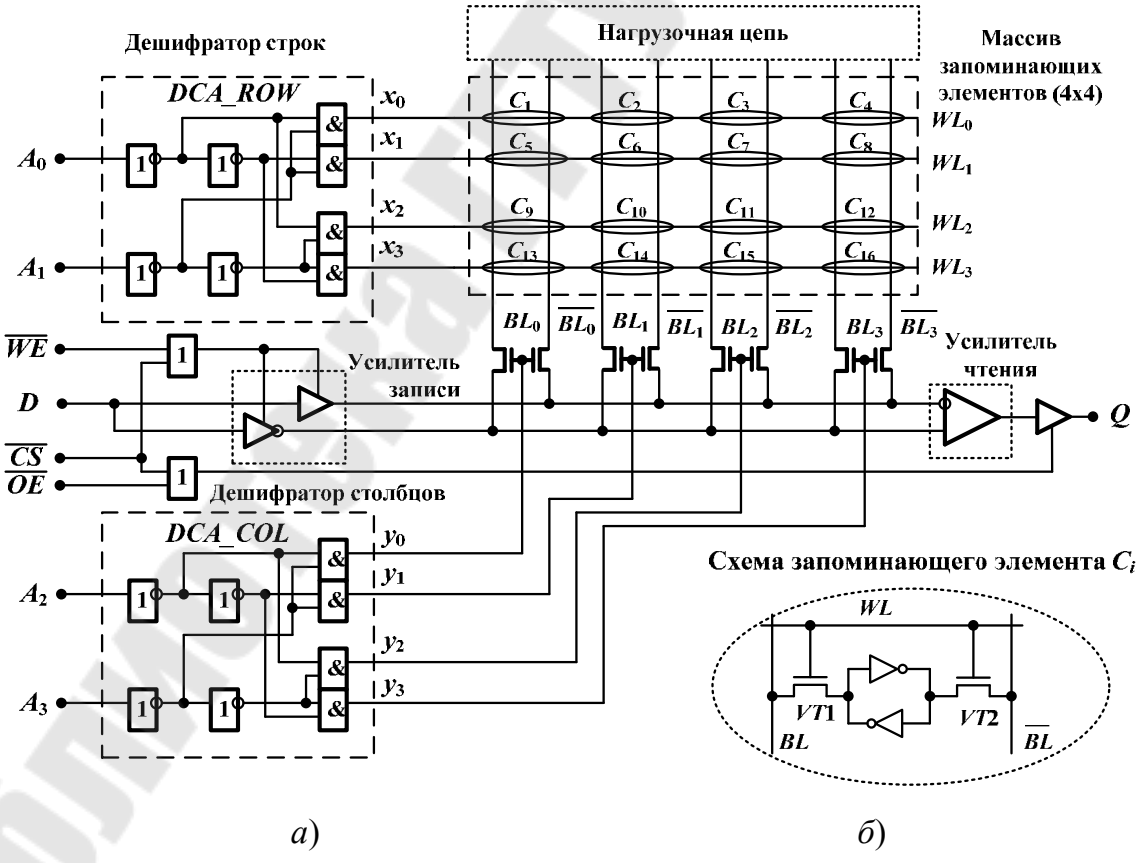


Рис. 3.10. Простейшее статическое ОЗУ:
а – принципиальная схема; *б* – схема запоминающего элемента

Кроме линий адреса и данных, в СОЗУ в общем случае используются три управляющих сигнала. Рассмотрим их назначение для одноразрядной памяти:

– *CS* (*Chip Select*) – сигнал выборки кристалла. Он позволяет выбрать (активизировать) отдельную микросхему памяти в случае использования больших массивов памяти, состоящих из нескольких микросхем (модулей).

– *WE* (*Write Enable*) – сигнал разрешения записи. При активном сигнале *WE* в ячейку, определяемую адресными линиями, записывается информация, находящаяся на линии данных.

– *OE* (*Output Enable*) – сигнал разрешения чтения. При активном сигнале *OE* на линию данных поступает информация из ячейки, определяемой адресными линиями.

Можно выделить три основных режима работы СОЗУ: запись информации, чтение информации и хранение информации.

Запись информации происходит следующим образом. На адресные линии поступает адрес ячейки, в которую будет производиться запись. Пусть A_0–A_3 = 0000. Тогда активизируются линии x_0 и y_0, соответственно будет выбрана ячейка *C*3. На вход данных *D* поступает записываемая информация (логический «0» или «1»). На входы *CS* поступает логический «0» для выбора данной микросхемы памяти, на вход *OE* поступает логическая «1», которая запрещает чтение информации, и на вход *WE* поступает логический «0», который разрешает запись информации. Записываемая информация со входа данных *D* через усилитель записи подается в первый столбец матрицы, в котором выбрана только одна ячейка – *C*3.

Чтение информации происходит аналогично, за исключением того, что на вход *WE* поступает логическая «1», которая переводит выходы усилителя записи в высокоимпедансное состояние, а на вход *OE* поступает логический «0», который разрешает работу усилителя чтения и выходного буфера.

Ячейка памяти (рис. 3.10, *б*) представляет собой *RS*-триггер, который может находиться в одном из двух устойчивых состояний. Запись-чтение состояния *RS*-триггера происходит через транзисторные ключи *VT*1 и *VT*2, которые стоками соединены с разрядными шинами BL и $\overline{BL}$, а затворами – с линией выборки слова *WL*. Когда *WL* = 0, триггер хранит свое состояние. Когда *WL* = 1, триггер подключается к разрядным шинам BL и $\overline{BL}$, на которых при помощи нагрузочной цепи установлен некоторый фиксированный потенциал – в данном

случае *Vdd* (напряжение питания матрицы). Иногда используют нулевой потенциал или *Vdd*/2. Тогда если триггер хранил логический «0», на шине BL наблюдается падение, тогда как на шине $\overline{BL}$ напряжение не изменяется. Аналогично, если триггер хранил логическую «1», падение напряжения наблюдается на шине $\overline{BL}$.

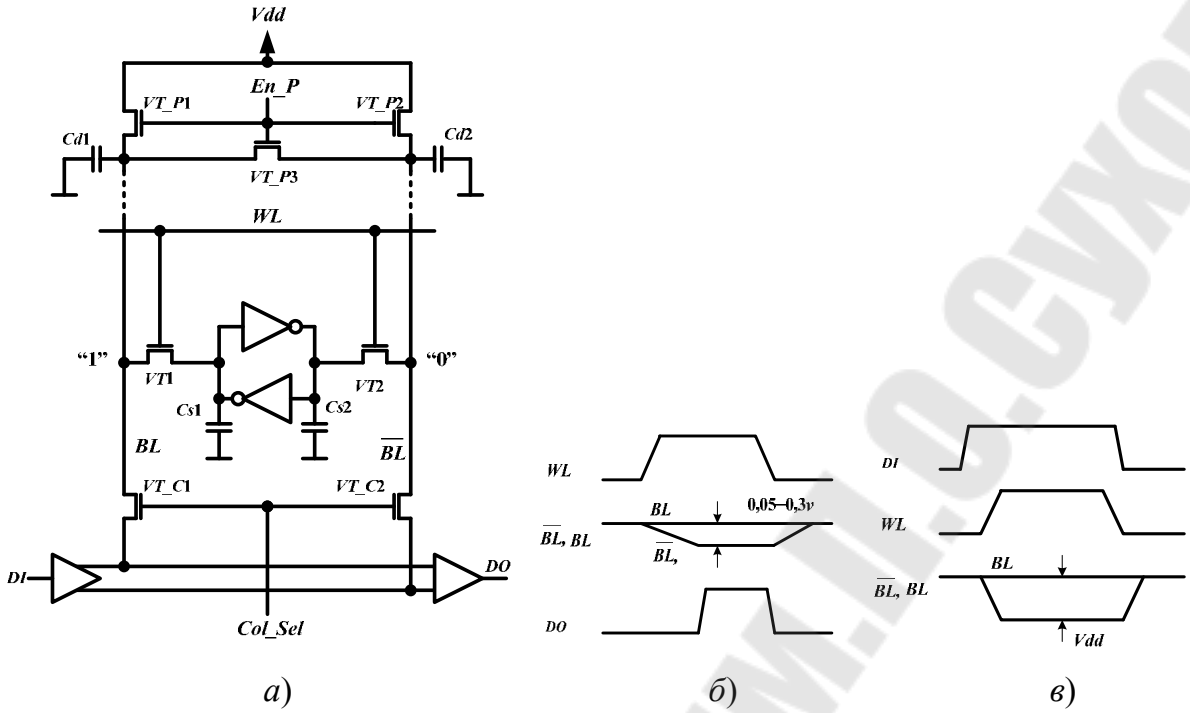
Линии BL и $\overline{BL}$ имеют значительную емкость, поэтому при чтении информации возникает опасность ее разрушения. Поэтому применяют специальную схему, которая позволяет автоматически восстанавливать или регенерировать информацию и усиливать разницу потенциалов разрядной шины. На рис. 3.11 приведены принципиальная схема столбца матрицы ЗЭ и временные диаграммы чтения и записи логической «1». Усилитель чтения должен иметь высокую чувствительность, чтобы преобразовать парафазный сигнал ($\Delta V = 0{,}05–0{,}33v$) в логический уровень, который через выходной буфер поступает на выход *Q*.

При записи информации в ячейку памяти на одну из линий BL или $\overline{BL}$ необходимо подать низкий потенциал. При этом усилитель записи должен иметь достаточно высокую нагрузочную способность, чтобы быстро перезарядить суммарную емкость разрядной шины и *RS*-триггера.

Статическая память может входить в состав СБИС или выпускаться в виде отдельной микросхемы. Память, входящая в состав СБИС, как правило, имеет раздельные линии для входных и выходных данных и разрядность, соответствующую разрядности применяемого процессора. Для быстродействующих систем по входу и выходу данных, а также по входу адреса располагают триггера-защелки.

Память, выпускаемая в виде отдельной микросхемы, имеет некоторые отличия. Для уменьшения числа внешних выводов входы и выходы данных объединяют в двунаправленную шину. Режим работы памяти определяется только двумя сигналами – *CS* и *WE* (а не тремя, как на рис. 3.10). Сигнал *CS* разрешает работу микросхемы (как правило, активный уровень – низкий). Низкий уровень на линии *WE* определяет операцию записи, а высокий – чтения.

Память, представленная на рисунке 3.10, является полностью асинхронной. Время чтения или записи информации определяется только внутренними задержками кристалла и не зависит от тактовой частоты работы системы. В микропроцессорных системах все операции определяются фронтами тактовых импульсов. Поэтому выпускают статическую память, которая имеет вход синхронизации и выполняет операции под управлением импульсов синхронизации.



а) *б*) *в*)

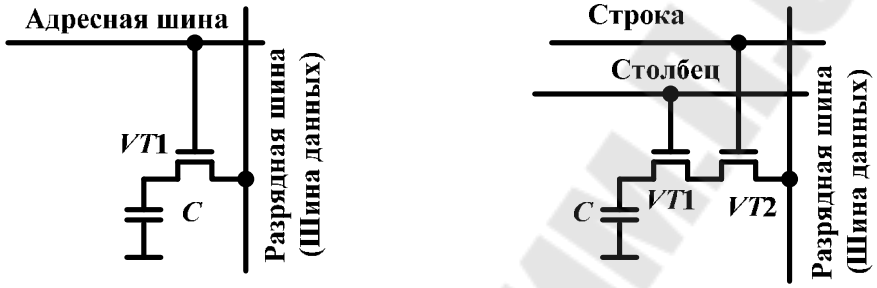
Рис. 3.11. Столбец матрицы статической памяти:
а – принципиальная схема; *б* – временная диаграмма чтения;
в – временная диаграмма записи

В высокопроизводительных системах применяют также конвейерную статическую память, которая имеет дополнительные регистры-защелки по шине данных. Это позволяет читать или писать данные одновременно с передачей следующего адреса. Кроме того, конвейерная память оснащена специальными схемами управления адресом, которые позволяют реализовать пакетный режим работы, при котором передается только адрес первой ячейки, а адреса следующих последовательно расположенных ячеек вычисляются внутри памяти.

3.3. Динамические оперативные запоминающие устройства

Хранение информации в динамическом ОЗУ осуществляется на конденсаторе, а не на триггере, как в статическом. Поэтому такие схемы более компактны и позволяют размещать на кристалле почти в пять раз больше запоминающих элементов при равной площади. Отсутствие заряда на конденсаторе соответствует значению логического «0», а наличие заряда – логической «1». На рис. 3.12 представлены схемы простейших запоминающих элементов динамических ОЗУ – с одно- и двухкоординатной выборкой.

Транзистор *VT*1 на рис. 3.12, *а* или последовательно соединенные транзисторы *VT*1, *VT*2 на рис. 3.12, *б* служат для отключения запоминающего конденсатора от разрядной линии (или шины данных) в режиме хранения. Сток транзистора *VT*1 (рис. 3.12, *а*) образует одну из обкладок конденсатора. Между обкладками расположен тонкий слой оксида кремния. При выборке ЗЭ на затвор подается напряжение, открывающее транзистор. Емкость *C* подключается к разрядной линии, и в зависимости от наличия или отсутствия заряда изменяется потенциал разрядной линии.



а) *б*)

Рис. 3.12. Простейшая ячейка запоминающего элемента динамического ОЗУ: *а* – однотранзисторная; *б* – двухтранзисторная

Процесс чтения состояния запоминающего элемента ДОЗУ представлен на рис. 3.13. Перед считыванием производится предзаряд разрядной шины *D* до уровня половины напряжения питания – *Vdd*/2. Пусть на конденсаторе C_S в момент считывания отсутствовал заряд (или хранился логический «0»). Разрядная шина *D* имела заряд:

$$Q = C_D \cdot V_{dd} / 2. \tag{3.1}$$

После выборки разрядная шина *D* имеет разряд:

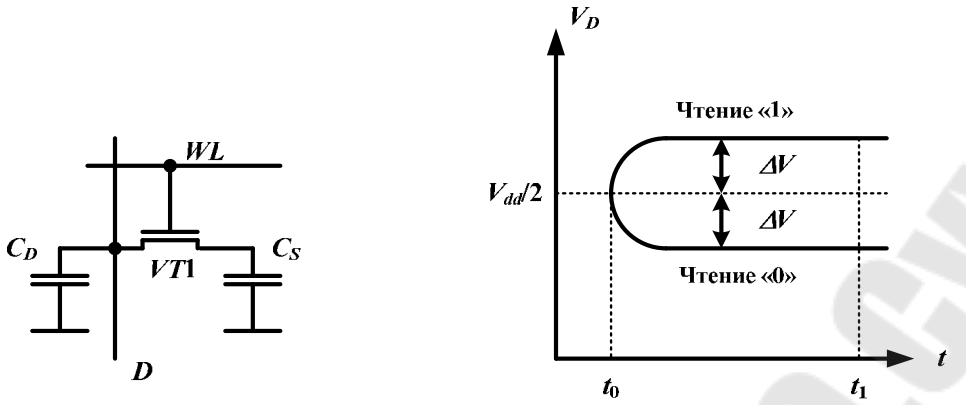
$$Q = (C_D + C_S) \cdot (V_{dd} / 2 - \Delta V). \tag{3.2}$$

Приравнивая правые части, получим:

$$C_D \cdot V_{dd} / 2 = (C_D + C_S) \cdot (V_{dd} / 2 - \Delta V). \tag{3.3}$$

Откуда:

$$\Delta V = \frac{C_D \cdot V_{dd}}{2 \cdot (C_D + C_S)}. \tag{3.4}$$



а) *б*)

Рис. 3.13. Простейший запоминающий элемент динамического ОЗУ: *а* – схема; *б* – диаграмма чтения

В современных ДОЗУ используют следующие соотношения емкостей C_S и C_D:

$$\frac{1}{15} \le \frac{C_D}{C_S} \le \frac{1}{10}. \qquad (3.5)$$

Емкость разрядной шины C_D намного больше, чем емкость конденсатора хранения C_S, поэтому чтение разрушает хранимую информацию. Усилитель чтения должен перезаписать информацию в ячейку памяти. Кроме того, емкость C_S имеет саморазряд, поэтому необходимо периодически восстанавливать на ней хранимое напряжение, или регенерировать. Для снижения частоты регенерации стараются увеличивать C_S, увеличивая геометрические размеры или применяя диэлектрик с более высокой диэлектрической проницаемостью (например, – двуокись титана).

На рис. 3.14 представлена упрощенная структурная схема одноразрядной динамической оперативной памяти, которая имеет *n* адресных входов и один разряд данных, а на рис. 3.15, *а* – ее условное графическое изображение. Для сравнения условное графическое изображение СОЗУ представлено на рис. 3.19, *б*. Временные диаграммы работы динамического и статического ОЗУ представлены на рис. 3.15, *в*, *г* соответственно. Основу памяти составляет массив из запоминающих элементов, организованный в виде матрицы из 2^n строк и 2^n столбцов. Выбор конкретной ячейки памяти осуществляется двумя сигналами – линией выборки слова *WL* и разрядной шиной *D*. В каждый конкретный момент времени может быть активна только одна строка матрицы и только один столбец матрицы. Выбор

строки и столбца матрицы осуществляют дешифраторы строк и столбцов, соответственно. Однако если в статической памяти адрес строки и адрес столбца поступают раздельно, то в динамической эти адреса поступают по одним и тем же линиям. Для того чтобы определить, какая именно часть адреса находится в данный момент на шине адреса, используются два дополнительных вывода – *RAS* (*Row Address Strobe* – строб адреса строки) и *CAS* (*Column Address Strobe* – строб адреса столбца) соответственно. При отсутствии обращения к памяти на этих выводах установлен высокий уровень.

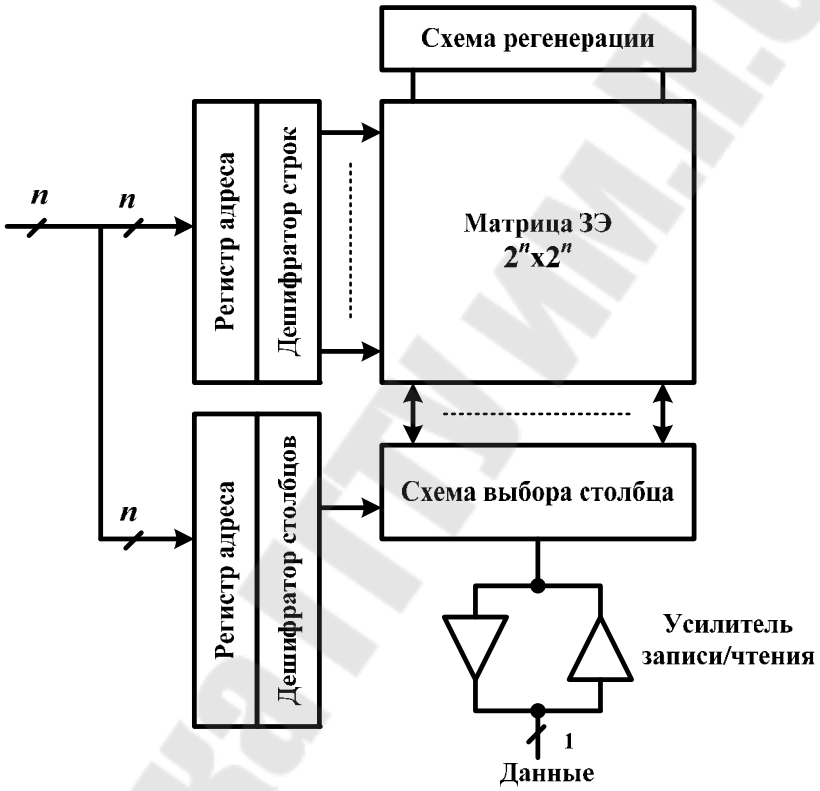
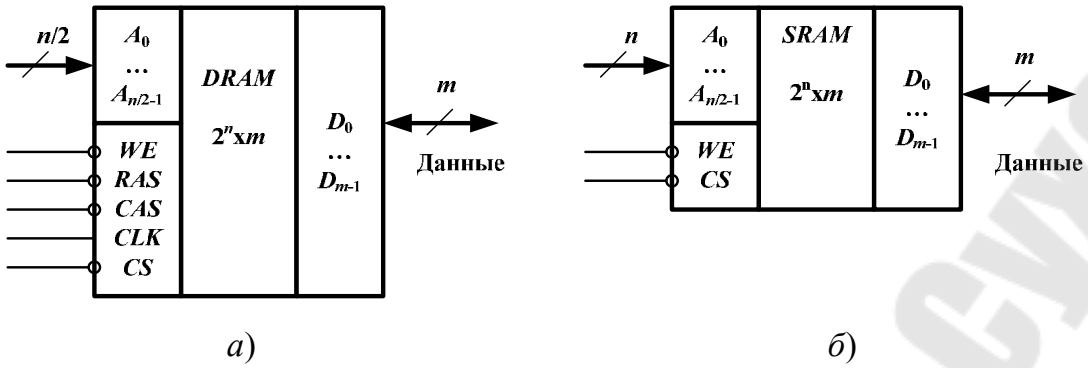
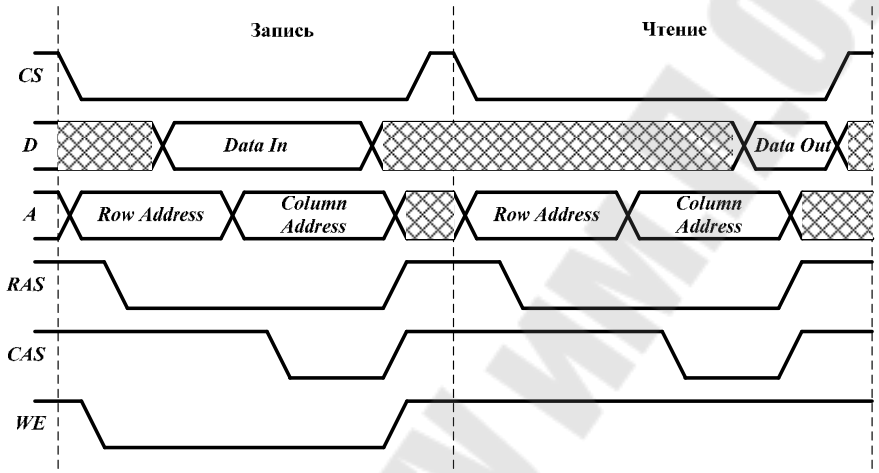


Рис. 3.14. Упрощенная структура динамического ОЗУ

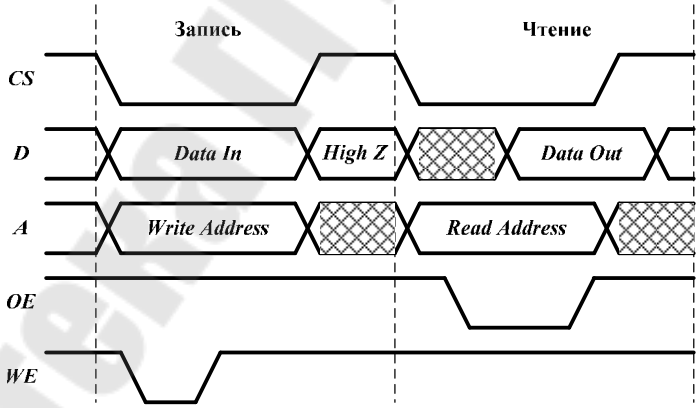
Сигнал выборки кристалла *CS* позволяет выбрать отдельную микросхему памяти или отдельный кристалл в случае использования нескольких микросхем или модулей. Сигнал разрешения записи *WE* определяет режим работы ОЗУ. При активном, как правило, низком уровне сигнале *WE* происходит запись информации, а при пассивном – чтение.



а) *б*)



в)



г)

Рис. 3.15. Сравнение работы ДОЗУ и СОЗУ:
а – условное графическое обозначение ДОЗУ; *б* – условное графическое обозначение СОЗУ; *в* – временные диаграммы работы ДОЗУ; *г* – временные диаграммы работы СОЗУ

Сигнал *CLK* служит для внутренней синхронизации работы ОЗУ. Рассмотрим основные режимы работы ДОЗУ – запись информации, чтение информации и хранение (регенерацию) информации и

сравним их с режимами работы статического ОЗУ с точки зрения времени обращения.

Запись информации в ДОЗУ происходит следующим образом (рис. 3.15, *в*). Активизируется сигнал *CS* и на адресные входы выставляется первая половина адреса – адрес строки. Также активизируется сигнал *WE*, который определяет операцию работы памяти, а именно – запись. Затем активизируется сигнал *RAS*, который запишет этот адрес во внутренний регистр адреса строки. На шину данных поступают записываемые данные. Затем на шину адреса поступает адрес столбца, и через некоторое время активизируется сигнал *CAS*, который сигнализирует о наличии второй половины адреса. Через некоторое время снимаются сигналы *RAS*, *CAS*, *WE* и *CS*, завершая цикл обращения к памяти. Чтение информации происходит аналогично, за исключением того, что не активизируется сигнал *WE*. Данные становятся доступными через некоторое время после активизации сигнала *CAS*.

Запись информации в СОЗУ происходит следующим образом (рис. 3.15, *г*). Активизируется сигнал *CS*, на адресные входы выставляется адрес, на шину данных поступают записываемые данные, а через некоторое время активизируется сигнал *WE*, показывая, что происходит операция записи. При чтении активизируется сигнал *CS*, на адресные входы выставляется адрес, а через некоторое время на шине данных становятся доступными считанные данные.

Таким образом, время доступа статической памяти определяется лишь внутренними задержками, в то время как для динамической это время вычисляется как сумма времен задержек установки адреса строки (t_{AS}), удержания адреса строки и столбца (t_{AH}), задержка между подачей адреса строки и адреса столбца (t_{RCD}), время получения содержимого ячейки после подачи сигнала *CAS* (t_{CAS}), время, требуемое для смены адреса строки (t_{RP}) и период регенерации (t_{REF}).

При хранении информации в статическом ОЗУ энергия практически не потребляется (за исключением токов утечки, которые на несколько порядков меньше рабочих токов). В динамическом ОЗУ для хранения необходима регенерация информации. Для регенерации необходимо обратиться к каждой ячейке памяти. Однако учитывая, что при обращении к любой ячейке строки активизируется вся строка, для регенерации часто используют только сигнал *RAS*. Поэтому для регенерации ДОЗУ емкостью 128 Мбит (например, MT48*LC*16M8A2 с организацией 4M x 8 x 4 банка) достаточно перебрать всего 4096 адресов. Современные микросхемы ДОЗУ, как правило, оснащены встроенными аппаратными средствами, позволяющими автоматически проводить регенерацию.

3.4. Тестирование оперативного запоминающего устройства

3.4.1. Физические дефекты оперативного запоминающего устройства

Для ОЗУ основным технологическим параметром является количество запоминающих ячеек на единицу площади кристалла. Регулярность структуры ЗУ позволяет достигать максимально возможной плотности по сравнению с цифровыми устройствами произвольной структуры. Таким образом, как правило, новые технологические нормы и соответствующие им техпроцессы впервые отрабатываются на ОЗУ. Поэтому как на стадии производства, так и во время их эксплуатации могут появляться различные физические дефекты (далее именуемые неисправностями), которые вызывают неисправное поведение ОЗУ. Причины возникновения физических дефектов в первую очередь связаны с особенностями новой внедряемой технологии. На рис. 3.16 представлены возможные причины отказов памяти.

Природа физических дефектов и форма их проявления весьма разнообразна и динамична. Она в сильной мере зависит от многих параметров, таких, как: тип изготавливаемого ЗУ, технология его производства, качество исходных материалов и препаратов, продолжительность производства данного типа ЗУ, квалификация обслуживающего персонала и др. Определение факта возникновения дефекта и его классификация представляется весьма трудоемкой и зачастую неразрешимой задачей. Это прежде всего объясняется тем, что возникновение дефекта чаще всего можно определить лишь по косвенным признакам (например, – неправильное функционирование).

Рис. 3.16. Причины возникновения дефектов

Наиболее распространены следующие неисправности ОЗУ:

– константное состояние элемента памяти;

– взаимное влияние элемента памяти;

– константное состояние «Блока Управления Записью»;

– константное состояние управляющей линии «Чтение/Запись»;

– константное состояние управляющей линии «Выбор Микросхемы»;

– константное состояние линии шины данных;

– обрыв линии шины данных;

– замыкание линий шины данных;

– взаимное влияние линий шины данных;

– константное состояние линии шины адреса;

– обрыв линии шины адреса;

– замыкание линий шины адреса;

– обрыв линии дешифратора.

Следует отметить, что данный список является далеко не полным. Приведенные неисправности являются доминирующими при тестировании микросхем ОЗУ. Все возможные неисправности можно представить двумя основными группами – неисправности схем обрамления и неисправности массива элементов памяти. Рассмотрим более подробно данные типы неисправностей.

3.4.2. Классические модели неисправностей оперативного запоминающего устройства

К *неисправностям схем обрамления* относят следующие:

– адресные неисправности (неисправности дешифраторов адреса);

– неисправности логики чтения/записи.

Традиционно адресные неисправности (*AF*) включают в себя четыре модели: ни одна из всех ячеек памяти недоступна по заданному адресу A_i (рис. 3.17, *а*), заданная ячейка памяти C_i недоступна (рис. 3.17, *б*), по заданному адресу памяти A_i осуществляется доступ сразу к нескольким ячейкам (рис. 3.17, *в*) и доступ к заданной ячейке памяти C_j осуществляется по нескольким адресам (рис. 3.17, *г*).

Неисправности логики чтения/записи включают в себя неисправности таких функциональных блоков, как «Регистр Данных», «Усилители Считывания», «Блок Управления Записью» и для динамических ЗУ «Блок Регенерации». Все перечисленные блоки обеспечивают транспортировку данных из массива элементов памяти на внешнюю шину данных и обратно. Как было показано в ряде работ,

физические дефекты в электронном обрамлении ЗУ могут быть описаны моделями неисправностей матрицы ЗУ. Так, например, константные неисправности в логике чтения/записи проявляют себя так же, как и константные неисправности массива ЭП. Таким образом, при тестировании ЗУ, как правило, не рассматриваются неисправности логики чтения/записи как отдельные виды неисправностей.

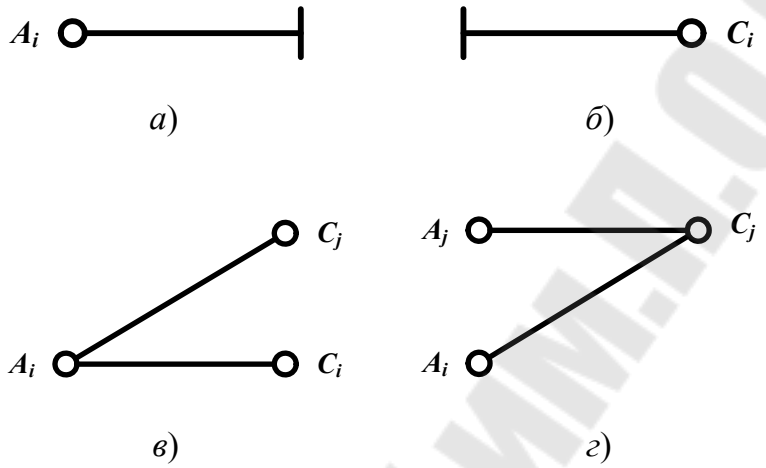


Рис. 3.17. Модели адресных неисправностей

Неисправности массива ячеек ОЗУ. Все множество неисправностей матрицы запоминающих ячеек ЗУ делится на несколько подмножеств в зависимости от количества ячеек, участвующих в описании конкретной неисправности:

– одну ячейку ЗУ;

– две ячейки ЗУ;

– несколько ячеек ЗУ (кодочувствительные неисправности).

Данная классификация характерна как для статических, так и для динамических ОЗУ.

К неисправностям, затрагивающим одну ячейку, относят константные и переходные неисправности:

1. *Константные неисправности* (*stuck-at faults* (*SAF*)).

Данные неисправности характеризуются тем, что состояние конкретной ячейки ЗУ (неисправной) постоянно находится в одном из возможных состояний, в состоянии нуля (0) или состоянии (1) независимо от выполненных ранее операций записи в данную ячейку противоположного значения. Различают неисправность константного нуля (*stuck-at* 0 (*SAF*0)) и неисправность константной единицы (*stuck-at* 1 (*SAF*1)). Также выделяют однократные и многократные константные неисправности.

2. *Переходные неисправности* (*transition faults*(*TF*)).

Подобные неисправности характеризуются невозможностью логического перехода состояния ячейки ЗУ (неисправной) из 0 в 1 (*transition up*) $\langle\uparrow\rangle$ или из 1 в 0 (*transition down*) $\langle\downarrow\rangle$ при выполнении соответствующих операций записи. Данный тип неисправности достаточно близок по своей сути к константным неисправностям. Действительно, если ячейка, имеющая переходную неисправность, оказывается в состоянии, из которого она не может перейти в другое состояние, ее поведение повторяет поведение ячейки, содержащей константную неисправность. Это обстоятельство позволяет использовать приемы, позволяющие обнаруживать одновременно оба типа неисправностей.

К неисправностям, затрагивающим две ячейки, относят неисправности взаимного влияния:

1. *Инверсные неисправности взаимного влияния* (*inversion coupling faults* (*CFin*)).

В данной неисправности участвуют две ячейки ЗУ – a_i и a_j, $i \neq j$, одна из которых a_i называется агрессором (*aggressor*), а вторая a_j жертвой (*victim*). Расположение агрессора и жертвы в адресном пространстве ЗУ произвольно. При наличии данной неисправности логический переход из 1 в 0 или из 0 в 1 в агрессоре a_i приводит к инверсии логического значения в жертве a_j. Таким образом, различают два вида инверсных неисправностей – $\langle\uparrow, a_j\rangle$ и $\langle\downarrow, a_j\rangle$. Для представления соотношения адресов агрессора и жертвы используют символы $\wedge$ и $\vee$, причем символ $\wedge$ означает факт того, что адрес агрессора меньше адреса жертвы ($i < j$), а символ $\vee$ наоборот ($i > j$). Тогда имеем четыре различных инверсных неисправности $\wedge\langle\uparrow, a_j\rangle$, $\wedge\langle\downarrow, a_j\rangle$, $\vee\langle\uparrow, a_j\rangle$ и $\vee\langle\downarrow, a_j\rangle$. Пример инверсной неисправности взаимного влияния $\wedge\langle\uparrow, a_j\rangle$ приведен на рис. 3.18.

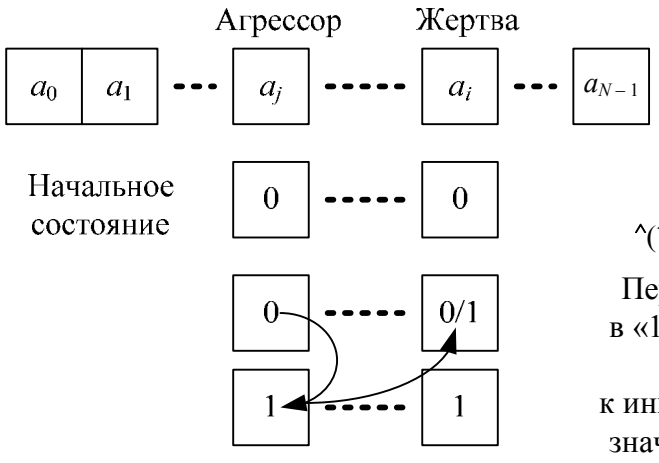


Рис. 3.18. Пример инверсной неисправности взаимного влияния $\wedge\langle\uparrow, a_j\rangle$

2. *Неисправности прямого действия* (*idempotent coupling faults* (*CFid*)).

При данной неисправности во время логического перехода из 1 в 0 или из 0 в 1 во влияющей a_i (агрессоре) ячейке происходит принудительная установка определенного логического значения 0 или 1 в ячейке жертвы a_j, на которую оказывается влияние агрессора. Различают восемь неисправностей прямого действия: $\wedge\langle\uparrow, 0\rangle$, $\wedge\langle\uparrow, 1\rangle$, $\wedge\langle\downarrow, 0\rangle$, $\wedge\langle\downarrow, 1\rangle$, $\vee\langle\uparrow, 0\rangle$, $\vee\langle\uparrow, 1\rangle$, $\vee\langle\downarrow, 0\rangle$ и $\vee\langle\downarrow, 1\rangle$.

При анализе эффективности тестов ЗУ анализируется их покрывающая способность для всех 12 неисправностей, в которых участвуют две ячейки.

3. *Кодочувствительные неисправности* (*pattern sensitive faults* (*PSF*)).

Неисправности, затрагивающие несколько ячеек ЗУ, называются кодочувствительными неисправностями. Для подобных неисправностей логическое состояние одной ячейки (жертвы) может зависеть от содержимого или от логических переходов других ячеек (агрессоров). Различают два вида кодочувствительных неисправностей: неограниченные (жертва и агрессоры расположены в произвольных местах памяти) и ограниченные (жертва и агрессоры расположены рядом). При тестировании массива ячеек ЗУ обычно придерживаются последней более реальной модели. На рис. 3.19 представлено схематическое изображение модели ограниченной кодочувствительной неисправности.

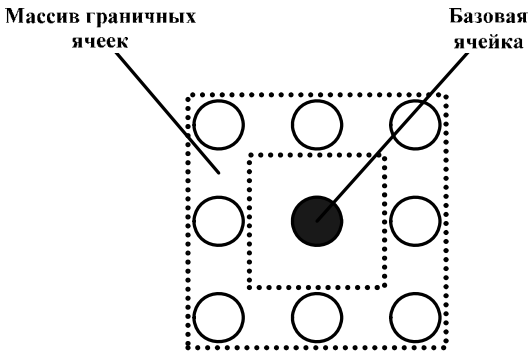


Рис. 3.19. Модель кодочувствительной неисправности

Как видно из приведенного рисунка, выделяют *базовую ячейку* (*base cell*), которая в данном случае является аналогом жертвы, и *соседние ячейки* (*neighborhood cells*), выступающие в роли агрессоров, количество и местоположение которых может быть произвольным.

3.4.3. Тесты оперативного запоминающего устройства

К данному моменту времени разработано большое количество разнообразных функциональных тестов ОЗУ классов N, N^2 и $N^{3/2}$, соответствующие зависимости числа циклов в последовательности тестирования от емкости N запоминающего устройства. Ниже приведены примеры наиболее распространенных тестов.

Memory scan, или *MSCAN*. Это наиболее простой тест, который состоит из следующей последовательности действий. Во все ячейки памяти записываются «0», значение ячеек проверяется на соответствие «0». Потом во все ячейки памяти записывается «1», значение ячеек проверяется на равенство «1». Процедура проверки памяти является очень быстрой, однако имеет низкую покрывающую способность. Данный тест имеет сложность $4N$ и гарантирует 100 % обнаружения только константных неисправностей.

GALPAT и *Walking* 0/1. Эти два теста состоят из похожего набора операций. Сначала тест записывает «0» или «1» во все ячейки, исключая базовую ячейку, которая содержит «1» или «0» соответственно. В течение теста базовая ячейка последовательно перемещается через всю память. Различие между тестами состоит в способе чтения значения базовой ячейки. *Walking* 0/1 после каждого шага выполнения читает значения всех ячеек памяти и только после этого читает базовую ячейку. *GALPAT* также читает значения всех ячеек, однако чтение базовой ячейки производится после чтения каждой ячейки памяти. Данные тесты имеют сложность $O(n^2)$. Например, *GALPAT* име-

ет сложность $2N^2 + 6N$, при этом гарантирует 100 % обнаружения *SAF*, *AF*, *TF*. Для более сложных неисправностей (*CF* и *PSF*) он не дает гарантии 100 % обнаружения.

В настоящее время объем оперативной памяти в персональных компьютерах составляет от 256 Мб до 1 Гб. Поэтому применение тестов со сложностью N^2 и $N^{3/2}$ неэффективно вследствие большого времени тестирования. Рассмотрим пример проверки памяти. Проверяется память объемом 256 Мб, которая работает на частоте 100 Мгц (10 нс цикл). Запись в каждую ячейку одного фиксированного значения (0 или 1) и чтение этого значения займет 5368709120 нс ≈ 5,37 с ($2^{28} = 268435456$). Тест сложности $N^{3/2}$ будет выполняться $2^{42} \cdot 10$ нс = = 43980 с ≈ 733 мин или 12,2 ч. Соответственно, нереально дождаться результатов.

Поэтому широкое распространение нашли только тесты сложности $O(n)$, получившие название маршевых тестов (табл. 3.1). Эти тесты обладают достаточной для большинства приложений покрывающей способностью и простотой реализации.

Любой маршевый тест состоит из конечного числа маршевых элементов. Маршевый элемент представляет собой конечную последовательность операций, применяемых к каждой ячейке памяти перед переходом к тестированию следующей. Направление перехода может осуществляться в одном из двух направлений: в направлении увеличения адреса (обозначается символом ⇑), т. е. адреса ячеек изменяются от 0 до $N - 1$, либо в направлении уменьшения адреса (обозначается символом ⇓), т. е. адреса ячеек изменяются в обратном порядке. Когда направление изменения адреса не имеет значения, это указывается символом ⇕ (при реализации конкретного маршевого теста необходимо выбрать какое-то определенное направление). Увеличение адресов не обязательно означает изменение от 0 до $N - 1$. В общем случае это может быть произвольная последовательность адресов, например: 5, 7, 2, 0, 3, 4, 1, 6. Необходимым условием является то, чтобы направления, обозначаемые символами ⇑ и ⇓, были бы противоположны друг другу. Подобные последовательности адресов (в которых не соблюдается принцип постоянного увеличения или уменьшения адреса) используются для обнаружения неисправностей в адресной логике систем памяти. Набор операций, применяемых в рамках маршевого элемента к каждой ячейке памяти, содержит следующие операции:

– запись логического нуля в ячейку памяти (обозначается *w*0);

– запись логической единицы в ячейку памяти (обозначается *w*1);

– считывание хранимого значения из ячейки памяти и сравнение его с логическим нулем (обозначается символом $r0$);

– считывание хранимого значения из ячейки памяти и сравнение его с логической единицей (обозначается символом $r1$).

Таблица 3.1

Маршевые тесты

Номер	Тест	Алгоритм	*N*
1	*MATS*	⇳($w0$); ⇑($r0$,$w1$); ⇳($r1$)	4*N*
2	*MATS+*	⇳($w0$); ⇑($r0$,$w1$); ⇓($r1$,$w0$)	5*N*
3	*MATS++*	⇳($w0$); ⇑($r0$,$w1$); ⇓($r1$,$w0$,$r0$)	6*N*
4	*Marching* 1/0	⇳($w0$); ⇑($r0$,$w1$,$r1$); ⇓($r1$,$w0$,$r0$); ⇑($w1$); ⇑($r1$,$w0$,$r0$); ⇓($r0$,$w1$,$r1$)	14*N*
5	*March X*	⇳($w0$); ⇑($r0$,$w1$); ⇓($r1$,$w0$); ⇳($r0$)	6*N*
6	*March Y*	⇳($w0$); ⇑($r0$,$w1$,$r1$); ⇓($r1$,$w0$,$r0$); ⇳($r0$)	8*N*
7	*March C*	⇳($w0$); ⇑($r0$,$w1$); ⇑($r1$,$w0$); ⇳($r0$); ⇓ ($r0$,$w1$); ⇓($r1$,$w0$); ⇳($r0$)	11*N*
8	*March C–*	⇳($w0$); ⇑($r0$,$w1$); ⇑($r1$,$w0$); ⇓($r0$,$w1$); ⇓ ($r1$,$w0$); ⇳($r0$)	10*N*
9	*March A*	⇳($w0$); ⇑($r0$,$w1$,$w0$,$w1$); ⇑($r1$,$w0$,$w1$); ⇓ ($r1$,$w0$,$w1$,$w0$); ⇓($r0$,$w1$,$w0$)	15*N*
10	*March B*	⇳($w0$);⇑($r0$,$w1$,$r1$,$w0$,$r0$,$w1$);⇑ ($r1$,$w0$,$w1$);⇓($r1$,$w0$,$w1$,$w0$);⇓($r0$,$w1$,$w0$)	17*N*
11	*Algorithm B*	⇳($w0$); ⇑($r0$,$w1$,$w0$,$w1$); ⇑ ($r1$,$w0$,$r0$,$w1$); ⇓($r1$,$w0$,$w1$,$w0$); ⇓ ($r0$,$w1$,$r1$,$w0$)	17*N*

Например, маршевый тест *March C* выглядит следующим образом: ⇳($w0$); ⇑($r0$, $w1$); ⇑($r1$, $w0$); ⇳($r0$); ⇓($r0$, $w1$); ⇓($r1$, $w0$); ⇳($r0$).

Если в результате операции сравнения оказывается, что считанное из памяти значение не совпадает с ожидаемым, считается, что данная ячейка содержит ошибочное значение, вызванное наличием неисправности.

Литература

1. Угрюмов, Е. П. Цифровая схемотехника / Е. П. Угрюмов. – Санкт-Петербург : БХВ, 2000. – 528 с.

2. Преснухин, Л. Н. Расчет элементов цифровых устройств : учеб. пособие для вузов / Л. Н. Преснухин. – Москва : Высш. шк., 1982. – 383 с.

3. Браммер, Ю. А. Импульсные и цифровые устройства : учебник / Ю. А. Браммер. – Москва : Высш. шк., 1999. – 351 с.

4. Нарышкин, А. К. Цифровые устройства и микропроцессоры : учеб. пособие для вузов / А. К. Нарышкин. – Москва : Академия, 2008. – 318 с.

5. Гуров, В. В. Синтез комбинационных схем в примерах / В. В. Гуров. – Москва : МИФИ, 2001.

6. Гук, М. Аппаратные средства *IBM PC* : энциклопедия / М. Гук. – Санкт-Петербург : Питер, 2002. – 816 с.

7. Несвижский, В. Программирование аппаратных средств в *Windows* / В. Несвижский. – Санкт-Петербург : БХВ-Петербург, 2008. – 528 с.

8. Новиков, Ю. В. Основы цифровой схемотехники. Базовые элементы и схемы. Методы проектирования / Ю. В. Новиков. – Москва : Мир, 2001. – 379 с.

9. Василевский, А. В. Устройство и функционирование ЭВМ / А. В. Василевский. – Минск : ЕГУ, 2002.

10. Цилькер, Б. Я. Организация ЭВМ и систем / Б. Я. Цилькер, С. А. Орлов. – Санкт-Петербург : Питер, 2006. – 668 с.

11. Неразрушающее тестирование запоминающих устройств / В. Н. Ярмолик [и др.]. – Минск : Бестпринт, 2005. – 230 с.

Содержание

Учебное электронное издание комбинированного распространения

Учебное издание

Мурашко Игорь Александрович

ЭВМ И ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА

Курс лекций
по одноименной дисциплине
для студентов специальности 1-40 01 02
«Информационные системы и технологии»
дневной формы обучения

Электронный аналог печатного издания

Редактор *А. В. Власов*
Компьютерная верстка *М. В. Аникеенко*

Подписано в печать 13.04.11.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 6,04. Уч.-изд. л. 5,89.
Изд. № 64.
E-mail: ic@gstu.by
http://www.gstu.by

Издатель и полиграфическое исполнение:
Издательский центр учреждения образования
«Гомельский государственный технический университет
имени П. О. Сухого».
ЛИ № 02330/0549424 от 08.04.2009 г.
246746, г. Гомель, пр. Октября, 48.