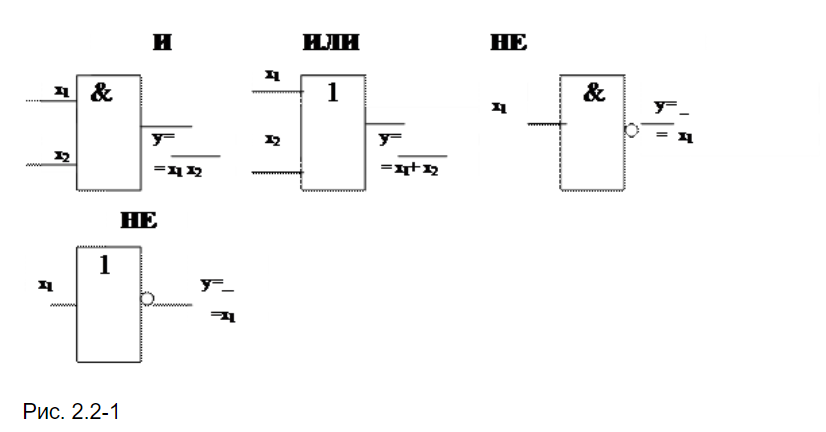
**30. Синтез логических схем по логическим выражениям в булевом базисе. Логический базис И-НЕ. Логический базис ИЛИ-НЕ.**

****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 имеют приведенные на рис. 2.2-1 обозначения. При синтезе схемы по логическому выражению, составляющие логические операции представляются в виде соответствующих логических элементов, связи между которыми определяются последовательностью выполнения логических операций в заданном выражении. Пример: Синтезировать логическую схему в базисе И, ИЛИ, НЕ, реализующую логическое выражение:

*Решение*

Входными сигналами синтезируемой схемы являются x1, x2, x3, а выходным - y1.

Реализацию заданного выражения в виде логической схемы можно начать или с последней операции, или с первой.

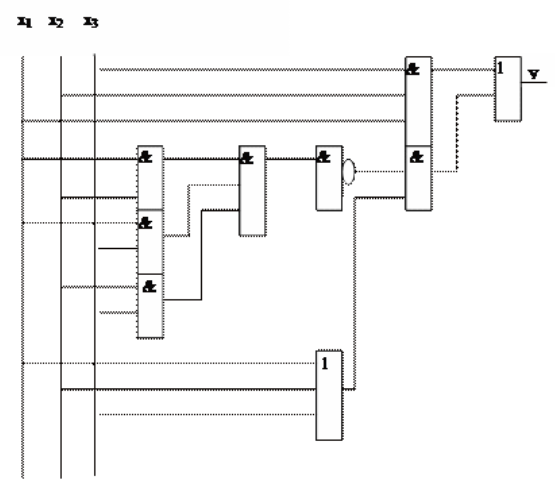
Последней операцией в заданном выражении является операция логического сложения двух операндов

|  |
| --- |
| y= ( x1x2 +x1x3 +x2 x3 )(x1 +x2 +x3 ) и х1 x2 x3, |

поэтому для её реализации требуется элемент ИЛИ(1) с двумя входами, на выходе которого будет сформирован сигнал, соответствующий y1, если на его входы будут поданы эти два слагаемые (например, первое слагаемое на второй вход, а второе слагаемое на первый вход).

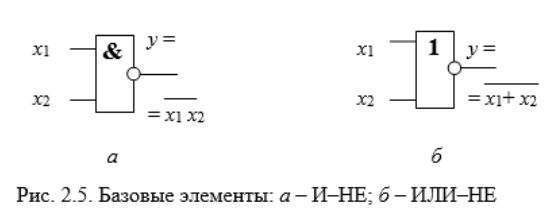
На первый вход ИЛИ(1) подается логическое произведение х1 x2 x3, для реализации которого необходимо использовать логический элемент И с тремя входами, на которые подаются входные переменные х1, x2, x3. Аналогичным образом рассматривается последовательность формирования выражения

|  |
| --- |
| y= ( x1x2 +x1x3 +x2 x3 )(x1 +x2 +x3 ), |

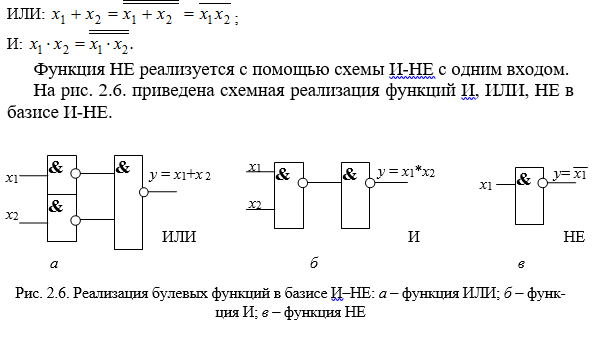
****которое соответствует сигналу, подаваемому на второй вход элемента ИЛИ(1). В результате синтезируется схема для заданного выражения, приведенная на рис.2.2-2 .

Булевый базис не является единственной функционально полной системой логических функций. Среди других наибольшее распространение получили базис И–НЕ и базис ИЛИ–НЕ.

Чтобы доказать логическую полноту любого базиса, достаточно показать, что в этом базисе можно реализовать базовые функции И, ИЛИ, НЕ.

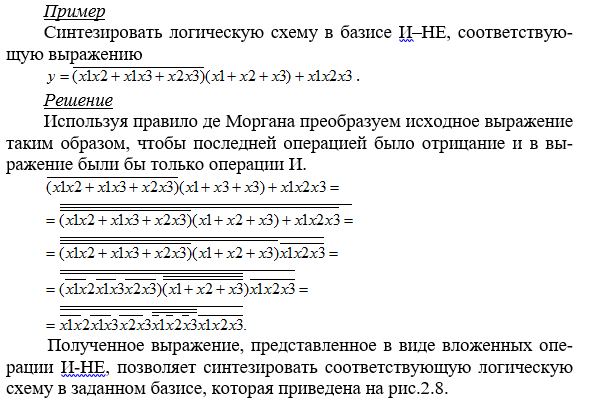
****Для базиса И-НЕ в качестве базового элемента используется элемент приведенный на рисунке рис. 2.5,а.

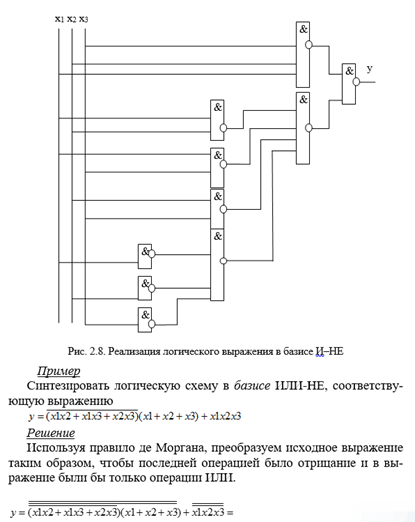
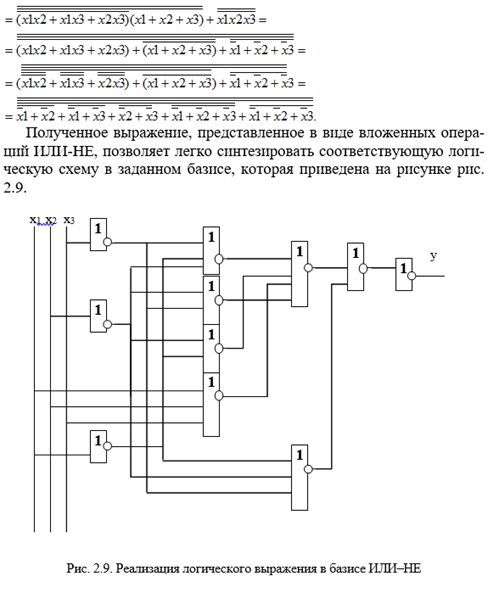
Реализация с помощью функции И-НЕ базовых функций алгебры Буля осуществляется следующим образом.

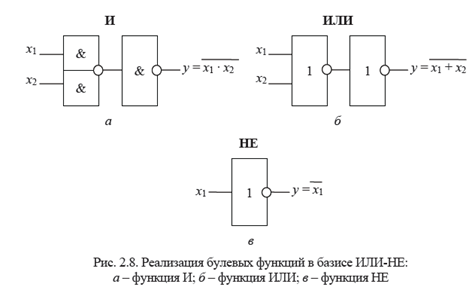


Реализация с помощью логической функции ИЛИ-НЕ базовых функций алгебры Буля осуществляется следующим образом.



При синтезе логических схем в заданном базисе логических элементов (например, в базисах И–НЕ, или ИЛИ–НЕ) целесообразно предварительно исходное выражение привести к форме, в которой в выражении будут использованы только логические операции, соответствующие используемым логическим элементам в заданном базисе.



Реализация с помощью логической функции ИЛИ-НЕ базовых функций алгебры Буля осуществляется следующим образом.



ИЛИ:x1+x2 =

И:

Функция НЕ реализуется с помощью схемы ИЛИ-НЕ с одним входом.

На рис. 2.8. приведена схемная реализация операций И, ИЛИ, НЕ

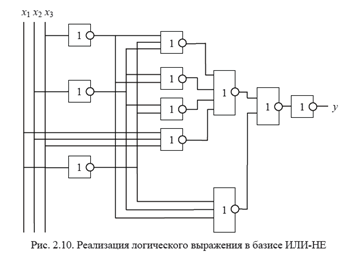
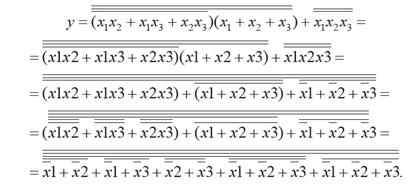
в базисе ИЛИ-НЕ

При синтезе логических схем в заданном базисе логических элементов (например, в базисах И–НЕ, или ИЛИ–НЕ) целесообразно предварительно исходное выражение привести к форме, в которой в выражении будут использованы только логические операции, соответствующие используемым логическим элементам в заданном базисе.

Пример:

Синтезировать логическую схему в базисе ИЛИ-НЕ, соответствующую выражению:

*Решение*

Используя правило де Моргана, преобразуем исходное выражение таким образом, чтобы последней операцией было отрицание и в выражение были бы только операции ИЛИ.

Полученное выражение, представленное в виде вложенных операций ИЛИ-НЕ, позволяет легко синтезировать соответствующую логическую схему в заданном базисе, которая приведена на рисунке рис. 2.10.

**31. Законы и правила алгебры Буля.**

В алгебре Буля логические выражения включают логические операции И, ИЛИ, НЕ, которые могут быть использованы в самых различных сочетаниях. При оценке значения такого выражения необходимо решить его для конкретного набора переменных. В алгебре Буля применяется следующая приоритетность выполнения операций: сначала рассчитываются значения имеющих место отрицаний и скобок, затем выполняется операция И (логическое умножение); самый низший приоритет имеет операция ИЛИ (логическая сумма).

При работе с булевыми логическим выражениями используются следующие законы, правила и операции.

Переместительный (коммутативный) закон.

Закон справедлив как для конъюнкции, так и для дизъюнкции.

– от перемены мест логических слагаемых сумма не меняется

х1 + х2 + х3 + х4 = х4 + х3 + х2 + х1

– от перемены мест логических сомножителей их произведение не меняется

х1х2х3х4 = х4х3х2х1

Этот закон справедлив для любого количества логических операндов.

***Сочетательный*(ассоциативный)*закон.***

Справедлив как для конъюнкции, так и для дизъюнкции.

– при логическом сложении отдельные слагаемые можно заменить их суммой

х1 + х2 + х3 + х4 = (х2 + х3) + х1 + х4 = (х1 + х4 ) + (х2 + х3)

– при логическом умножении отдельные логические сомножители можно заменить их произведением

х1 х2 х3 х4 = (х2 х3)х1х4 = (х1 х4) (х2 х3)

***Распределительный* (дистрибутивный) *закон*.**

(х1 + х2) х3 = х1х3 + х2х3;

(х1 + х2) (х1 + х3) = х1 + х2х3

***Правило де Моргана.***

+– отрицание суммы равно произведению отрицаний

– отрицание произведения равно сумме отрицаний

***Операция склеивания.***

– операция склеивания для конъюнкций, где А – переменная или любое логическое выражение

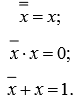
– операция склеивания для дизъюнкций

Если в качестве А используется простая конъюнкция, т. е. конъюнкция, представляющая собой логическое произведение переменных и их отрицаний, то имеет место



Как видно, в результирующем выражении количество переменных на единицу меньше, чем в склеенных конъюнкциях. Количество переменных в простой конъюнкции называется рангом конъюнкции, т. е. операция склеивания, примененная к простым конъюнкциям, дает результат с рангом, на единицу меньшим ранга исходных конъюнкций.

## Операции с отрицаниями.

– двойное отрицание равносильно отсутствию отрицания

## Операции с константами.

## 

## Операции с одинаковыми операндами.

## 

## 

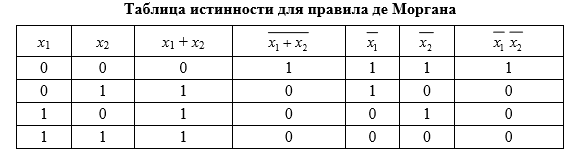
Законы, правила и операции алгебры Буля могут быть доказаны путем логического рассуждения, однако такое доказательство применимо только для простейших случаев. Доказать справедливость того или иного правила можно, если с помощью различных преобразований привести правую часть правила к выражению в левой части (или наоборот). Универсальным приемом доказательства является использование таблицы истинности. Это основано на том утверждении, что два выражения (правая и левая часть правила или закона) эквивалентны, если они принимают одинаковые значения на всех наборах логических переменных. Например, правило двойного отрицания, которое справедливо не только относительно одной переменной, но и любого логического выражения, можно доказать следующим рассуждением: если неверно утверждение, что выражение ложно, то очевидно утверждение, что это выражение истинно. Доказать справедливость распределительного закона в интерпретации выражением (х1 + х2)(х1 + х3) = х1 + х2х3 можно за счет приведения левой части к выражению правой части, раскрыв скобки:

(х1 + х2)(х1 + х3) = х1х2 + х1х1 + х1х3 + х2х3 = = х1х2 + х1 + х1х3 + х2х3 = х1(х2 + 1 + х3) + х2х3.

Помня, что логическая сумма с одним слагаемым, равным константе «1», равна «1», можно записать х1 + х2х3.

Используем таблицу истинности для доказательства правила де Моргана в варианте отрицание суммы равно произведению отрицаний.

Составим таблицу истинности для правой и левой частей и составляющих их функций



Из таблицы истинности видно, что правая и левая части доказываемого правила принимают одинаковые значения на всех наборах, следовательно они эквивалентны.

***Функционально полной системой булевых функций***(ФПСБФ) называется совокупность таких булевых функций (f1, f2, ..., fk), посредством которых можно записать произвольную булеву функцию f. Как уже было сказано, ФПСБФ являются «Стрелка Пирса» и «Штрих Шеффера».

**32. Параллелизм**

*Параллелизм* – основа высокопроизводительной работы всех подсистем вычислительных машин. *Организация памяти* любого уровня иерархии, организация системного ввода/вывода, организация мультиплексирования шин и т.д. базируются на принципах параллельной обработки запросов. Современные операционные системы являются многозадачными и многопользовательскими, имитируя параллельное *исполнение* программ посредством механизма прерываний.

Развитие процессоростроения также ориентировано на *распараллеливание* операций, т.е. на выполнение процессором большего числа операций за такт. Ключевыми ступенями развития архитектуры процессоров стали гиперконвейеризация, *суперскалярность*, неупорядоченная модель обработки, векторное процессирование (технология *SIMD*), *архитектура* *VLIW*. Все ступени были ориентированы на повышение степени параллелизма исполнения.

#### Уровни параллелизма

Распараллеливание операций – перспективный путь повышения производительности вычислений. Согласно закону Мура число транзисторов экспоненциально растет, что позволяет в настоящее время включать в состав CPU большое количество исполнительных устройств самого разного назначения. Прошли времена, когда функционирование ЭВМ подчинялось принципам фон Неймана.

В 70-е годы стал активно применяться принцип конвейеризации вычислений. Сейчас конвейер Intel Pentium 4 состоит из 20 ступеней. Такое распараллеливание на микроуровне – первый шаг на пути эволюции процессоров. На принципах конвейеризации базируются и внешние устройства. Например, динамическая память (организация чередования банков) или внешняя память (организация RAID).

Но число транзисторов на чипе росло. Использование микроуровневого параллелизма позволяло лишь уменьшать CPI (Cycles Per Instruction), так как миллионы транзисторов при выполнении одиночной инструкции простаивали. На следующем этапе эволюции в 80-е годы стали использовать параллелизм уровня команд посредством размещения в CPU сразу нескольких конвейеров. Такие суперскалярные CPU позволяли достигать CPI<1. Параллелизм уровня инструкций (ILP) породил неупорядоченную модель обработки, динамическое планирование, станции резервации и т.д. От CPI перешли к IPC (Instructions Per Clock). Но ILP ограничен алгоритмом исполняемой программы. Кроме того, при увеличении количества ALU сложность оборудования экспоненциально растет, увеличивается количество горизонтальных и вертикальных потерь в слотах выдачи. Параллелизм уровня инструкций исчерпал свои резервы, а тенденции Мура позволили процессоростроителям осваивать более высокие уровни параллелизма. Современные методики повышения ILP основаны на использовании процессоров класса SIMD. Это векторное процессирование, матричные процессоры, архитектура VLIW.

Параллелизм уровня потоков и уровня заданий применяется в процессорах класса MIMD. Многопоточные процессоры позволяют снижать вертикальные потери в слотах выдачи, а Simultaneous Multithreading (SMT) процессоры – как вертикальные, так и горизонтальные потери. Закон Мура обусловил также выпуск многоядерных процессоров (CMP). Лучшие современные вычислители – это мультикомпьютерные мультипроцессорные системы.



Иногда классифицируют параллелизм по степени гранулярности как отношение объема вычислений к объему коммуникаций. Различают мелкозернистый, среднезернистый и крупнозернистый параллелизм. Мелкозернистый параллелизм обеспечивает сам CPU, но компилятор может и должен ему помочь для обеспечения большего IPC. Среднезернистый параллелизм – прерогатива программиста, которому необходимо разрабатывать многопоточные алгоритмы. Здесь роль компилятора заключается в выборе оптимальной последовательности инструкций (с большим IPC) посредством различных методик (например, символическое разворачивание циклов). Крупнозернистый параллелизм обеспечивает ОС.

#### **Метрики параллелизма**

Допустим, имеется в наличии вычислительная система с неограниченными вычислительными ресурсами (в которой количество процессоров неограниченно). Как поведут себя программы в этой системе? Ускорятся ли они в неограниченное количество раз? Конечно, нет! Скорость работы простейшей однопоточной программы не изменится, так как она не использует параллелизм высокого уровня. Ее вычислительная нагрузка ляжет исключительно на один из процессоров. В отличие от микроуровневого параллелизма, который обеспечивается самим процессором, или в отличие от параллелизма уровня инструкций, где участия программиста не требуется (за исключением SIMD), привлечение к вычислениям нескольких процессоров требует существенной модификации алгоритма. Фактически требование дальнейшего повышения производительности вычислений вынуждает применять новые методики программирования, такие как векторное процессирование (например, SSE), многопоточное программирование (например, с использованием технологии OpenMP) и т.д. Отметим, что классические языки программирования высокого уровня (такие, как C++) ориентированы исключительно на класс SISD.

**33. Устройства ЭВМ. Состав АЛУ.**

**Устройства ЭВМ**

Классическая ЭВМ состоит из трех основных устройств: арифметико-логического устройства, устройства управления и запоминающего устройства. Рассмотрим особенности организации этих устройств. Прежде всего, рассмотрим структуру арифметико-логического устройства:

**Состав АЛУ**

АЛУ состоит из следующих типовых узлов:

– Регистры (R), служащие для хранения операндов и результатов;

– Сумматор (SM), служащий для выполнения операции суммирования многоразрядных кодов;

– Операционные узлы (ОУ), служащие для выполнения логических операций;

– Мультиплексор (MS);

– Счетчик (Сч), обеспечивающий подсчет тактов длинных операций;

– Регистр флажков (RF), служащий для фиксации особой информации, характеризующей полученный результат.

Для передачи информации между отдельными узлами используются шины Ш1 – Ш3. Шина Ш3 обеспечивает также связь с запоминающими устройствам ЗУ (ЭВМ).

Управляющий блок осуществляет выработку множества управляющих сигналов Y, обеспечивающих выполнение элементарных операций (микроопераций) типовыми узлами операционного блока.

При работе управляющая часть АЛУ использует код заданной операции (например сложение, умножение, вычитание и т. п.), а также информацию о состоянии операционного блока, представленную в виде множества Х признаков, формируемых типовыми узлами. К признакам, вырабатываемым регистром и посылаемым в управляющую часть, относятся:

– «ноль регистра» (R{0...n} = 0) характеризует состояние, при котором во всех разрядах регистра имеет место нулевое значение;

– «ноль знака» (R{зн} = 0) – в знаковом разряде регистра находится значение 0;

– «единица старшего разряда» (R{1} = 1) – в старшем разряде регистра находится значение единица;

– «единица младшего разряда» (R{n} = 1) – в младшем разряде регистра находится значение единица.

К микрооперациям, которые может выполнять регистр при поступлении соответствующего управляющего сигнала уi , относятся:

– прием кода;

– выдача прямого кода;

– выдача инверсного кода;

– установка единицы в некотором разряде регистра;

– обнуление знакового разряда;

– сдвиг кода влево;

– сдвиг кода вправо;

– обнуление регистра (во все разряды регистра устанавливается нулевое значение).

Счетчик может выполнять следующие операции, инициируемые по управляющим сигналам, поступающим из управляющего блока:

– Установка нуля в счетчике;

– Установка в счетчике некоторого начального значения;

– Установка режима счета (обратный или прямой счет);

– Изменение находящегося в счетчике текущего значения на единицу

К признакам, вырабатываемым счетчиком и посылаемым в управляющую часть, относятся:

– «ноль счетчика» («0» Сч) – характеризует состояние, при котором во всех разрядах регистра имеет место нулевое значение;

– «переполнение счетчика» – при поступлении очередного счетного сигнала счетчик переходит от максимального значения к значению «0».

Счетчик может выполнять следующие операции, инициируемые по управляющим сигналам, поступающим из управляющего блока:

– установка нуля в счетчике;

– установка в счетчике некоторого начального значения;

– установка режима счета (обратный или прямой счет);

– изменение находящегося в счетчике текущего значения на единицу.

К признакам, вырабатываемым сумматором и посылаемым в управляющую часть, относятся:

– признак нулевого результата;

– единичных значений во всех разрядах результата;

– единицы в первом знаковом разряде результата;

– единицы во втором знаковом разряде результата;

– переноса из старшего разряда сумматора;

– наличия в тетраде значения, большего 9;

– межтетрадного переноса.

Каждому из перечисленных состояний может соответствовать отдельный разряд (флажок) в регистре флажков.

Сумматор может выполнять следующие микрооперации, инициируемые по управляющим сигналам, поступающим из управляющего блока:

– прием кода двух операндов на свои входы;

– формирование поразрядной суммы операндов, поступающих на его входы;

– генерирование поразрядного переноса;

– распространение переносов через разряды поразрядной суммы, пропускающие перенос;

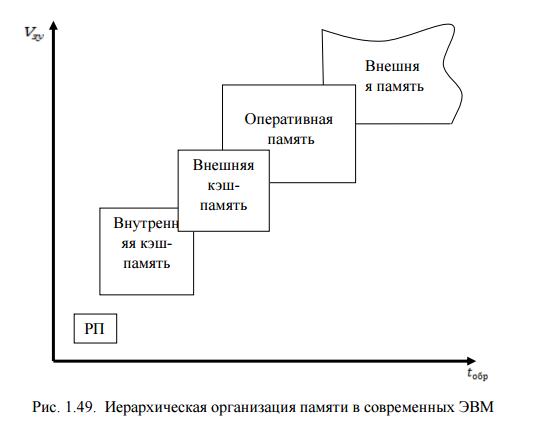
– прибавление единицы в младший разряд;

– прибавление корректирующих кодов в тетрады при сложении двоично-десятичных кодов.

Выполнение любой арифметической операции в АЛУ реализуется за счет выполнения определенной последовательности микроопераций в узлах операционной части АЛУ. Такие последовательности образуют алгоритм выполнения операций на уровне микроопераций. **Регистр флажков**

Регистр флажков (RF), служит для фиксации особой информации, характеризующей полученный результат.

**34. Типы памяти. ???**

 **Памятью** компьютера называется совокупность устройств, предназначенная для *хранения программ*, *вводимой информации*, *промежуточных результатов* и *выходных данных*.

|  |
| --- |
|  |

|  |
| --- |
| Память  Внутренняя память |

|  |
| --- |
| Внешняя память |

|  |
| --- |
| Энергонезависимая |

|  |
| --- |
| Энергозависимая |

|  |
| --- |
| ПЗУ |

|  |
| --- |
| ОЗУ |

|  |
| --- |
| Кэш-память |

|  |
| --- |
| Видеопамять |

|  |
| --- |
| Память с последовательным доступом |

|  |
| --- |
| Память с произвольным доступом |

|  |
| --- |
| ПЛ |

|  |
| --- |
| ПК |

|  |
| --- |
| НМЛ |

|  |
| --- |
| НГМД |

|  |
| --- |
| НЖМД |

|  |
| --- |
| CD-ROM |

|  |
| --- |
| Регистровая память |

Иерархическая структура памяти позволяет экономически эффективно сочетать хранение больших объемов информации с быстрым доступом к информации в процессе ее обработки.

На нижнем уровне иерархии находится **регистровая память** – набор регистров, входящих непосредственно в состав микропроцессора (центрального процессора – CPU). Регистры CPU программно доступны и хранят информацию, наиболее часто используемую при выполнении программы: промежуточные результаты, составные части адресов, счетчики циклов и т.д. Регистровая память имеет относительно небольшой объем (до нескольких десятков машинных слов). РП работает на частоте процессора, поэтому время доступа к ней минимально. Например, при частоте работы процессора 2 ГГц время обращения к его регистрам составит всего 0,5 нс.

**Оперативная память** – устройство, которое служит для хранения информации (программ, исходных данных, промежуточных и конечных результатов обработки), непосредственно используемой в ходе выполнения программы в процессоре. В настоящее время объем ОП персональных компьютеров составляет несколько гигабайт. Оперативная память работает на частоте системной шины и требует 6– 8 циклов синхронизации шины для обращения к ней. Так, при частоте работы системной шины 100 МГц (при этом период равен 10 нс) время обращения к оперативной памяти составит несколько десятков наносекунд.

Для заполнения пробела между РП и ОП по объему и времени обращения в настоящее время используется **кэш-память**, которая организована как более быстродействующая (и, следовательно, более дорогая) статическая оперативная память со специальным механизмом записи и считывания информации и предназначена для хранения информации, наиболее часто используемой при работе программы. Как правило, часть кэш-памяти располагается непосредственно на кристалле микропроцессора (внутренний кэш), а часть – вне его (внешняя кэш-память). Кэш-память программно недоступна. Для обращения к ней используются аппаратные средства процессора и компьютера.

**Внешняя память** организуется, как правило, на магнитных и оптических дисках, магнитных лентах. Емкость дисковой памяти достигает тысяч гигабайт при времени обращения менее 1 мкс. Магнитные ленты вследствие своего малого быстродействия и большой емкости используются в настоящее время в основном только как устройства резервного копирования данных, обращение к которым происходит редко, а может быть и никогда. Время обращения для них может достигать нескольких десятков секунд.

**35. Код Грея.**

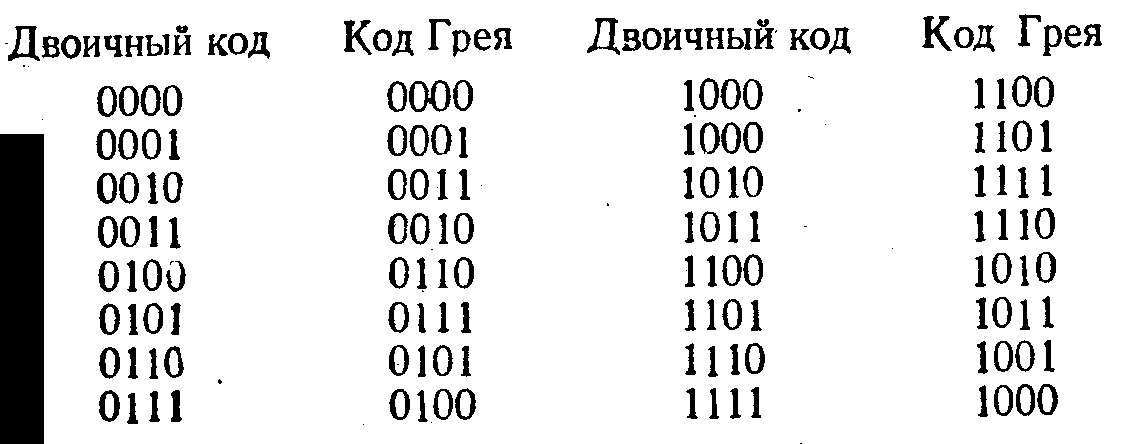
**Код Гре́я** — [двоичный код](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), в котором две «соседние» ([в упорядоченном, то есть лексикографическом, наборе](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BA%D1%81%D0%B8%D0%BA%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA)) кодовые комбинации различаются только цифрой в одном двоичном разряде. Иными словами, [расстояние Хэмминга](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5_%D0%A5%D1%8D%D0%BC%D0%BC%D0%B8%D0%BD%D0%B3%D0%B0) между соседними кодовыми комбинациями равно 1.

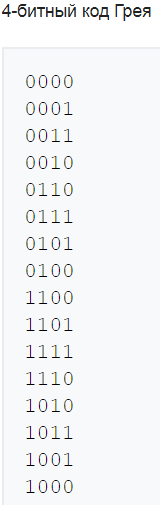
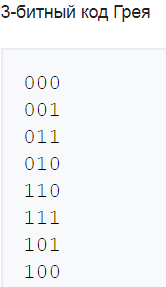
**Двоичным кодом Грея** порядка **n** называется последовательность всех image **n**-битных кодов, в которой любые два соседних кода различаются ровно в одном разряде.  
  
Пример кодов Грея порядка 2:

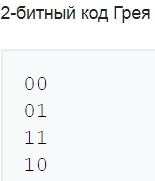
* 00
* 01
* 11
* 10

Нетрудно заметить, что такая последовательность не единственная(её как минимум можно обратить). Поэтому давайте разберемся в существовании двоичных кодов Грея других порядков и заодно выделим какой-то один вид таких последовательностей для дальнейшей работы.

Наиболее часто на практике применяется **рефлексивный**[**двоичный код**](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4)**Грея**, хотя в общем случае существует бесконечное множество кодов Грея со значениями цифр в разрядах, взятых из различных алфавитов. В большинстве случаев, под термином «код Грея» понимают именно рефлексивный бинарный код Грея.

Изначально предназначался для защиты от ложного срабатывания электромеханических переключателей. Сегодня коды Грея широко используются для упрощения выявления и [исправления ошибок](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BD%D0%B0%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B8_%D0%B8%D1%81%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BE%D1%88%D0%B8%D0%B1%D0%BE%D0%BA) в системах связи, а также в формировании сигналов обратной связи в системах управления.





Код Грея назван «рефлексивным» (отражённым) из-за того, что первая половина значений при изменении порядка эквивалентна второй половине, за исключением старшего бита. Старший бит просто инвертируется. При делении каждой новой половины пополам это свойство сохраняется.

Код назван в честь исследователя [Фрэнка Грея](https://ru.wikipedia.org/w/index.php?title=%D0%93%D1%80%D0%B5%D0%B9,_%D0%A4%D1%80%D1%8D%D0%BD%D0%BA&action=edit&redlink=1)[[en]](https://en.wikipedia.org/wiki/Frank_Gray_(researcher)), работавшего в лаборатории «[Bell labs](https://ru.wikipedia.org/wiki/Bell_Labs)». Грей запатентовал (патент № 2632058) и впервые использовал этот код в своей импульсной системе связи

Код Грея используется в передаче меняющихся цифровых сигналов в отсутствие [тактового сигнала синхронизации](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%BA%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB) (например, во многих видах датчиков). Представим себе, что код (обычный двоичный) перескакивает 3→4, или 0112 → 1002. Если из-за несовершенства считывателя мы прочитаем первый бит от 011, а остальные два — от 100, мы получим 0002=0 — число, далёкое от реальных значений. В коде Грея никаких посторонних значений не будет: перескок будет в одном разряде, 010G → 110G, и мы считаем либо старое 010G=3, либо новое 110G=4.

Если считыватель настолько медленный, что за время считывания показания несколько раз сменились, код Грея гарантирует, что ошибка будет невелика — меньше, чем реальное изменение сигнала. Например, если за время считывания показания сменились 010G=3 → 110G → 111G=5, то помимо этих трёх значений, можно получить 011G=2 — выходит ошибка на единицу.

Если датчик круговой (например, угловой [энкодер](https://ru.wikipedia.org/wiki/%D0%AD%D0%BD%D0%BA%D0%BE%D0%B4%D0%B5%D1%80)), то ему приходится перескакивать и с максимума до нуля. Такой перескок (с 100G=7 до 000G=0) тоже изменяет один разряд.

Коды Грея часто используются в датчиках-[энкодерах](https://ru.wikipedia.org/wiki/%D0%AD%D0%BD%D0%BA%D0%BE%D0%B4%D0%B5%D1%80). Их использование удобно тем, что два соседних значения шкалы сигнала отличаются только в одном разряде. Также они используются для кодирования номера дорожек в [жёстких дисках](https://ru.wikipedia.org/wiki/%D0%96%D1%91%D1%81%D1%82%D0%BA%D0%B8%D0%B9_%D0%B4%D0%B8%D1%81%D0%BA).

Код Грея можно использовать также и для решения задачи о [Ханойских башнях](https://ru.wikipedia.org/wiki/%D0%A5%D0%B0%D0%BD%D0%BE%D0%B9%D1%81%D0%BA%D0%B8%D0%B5_%D0%B1%D0%B0%D1%88%D0%BD%D0%B8).

### **Преобразование двоичного кода в код Грея**

Коды Грея легко получаются из двоичных чисел путём побитовой операции «[Исключающее ИЛИ](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B0%D1%8E%D1%89%D0%B5%D0%B5_%D0%98%D0%9B%D0%98)» с тем же числом, сдвинутым вправо на один бит и в котором старший разряд заполняется нулём. Следовательно, *i*-й бит кода Грея *Gi* выражается через биты двоичного кода *Bi* следующим образом:



где  — операция «исключающее ИЛИ»; биты нумеруются справа налево, начиная с младшего.

### **Преобразование кода Грея в двоичный код**

Обратный алгоритм — преобразование кода Грея в двоичный код — можно выразить рекуррентной формулой



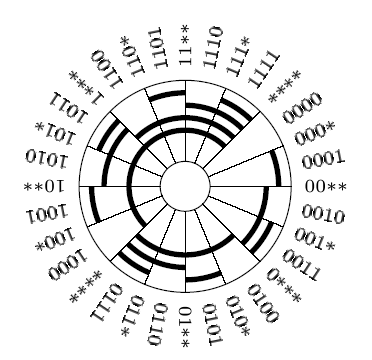
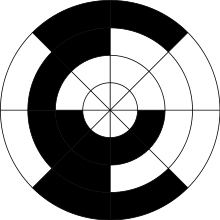
причём преобразование осуществляется побитно, начиная со старших разрядов, и значение Bi+1, используемое в формуле, вычисляется на предыдущем шаге алгоритма. Действительно, если подставить в эту формулу вышеприведённое выражение для *i*-го бита кода Грея, получим



Однако приведённый алгоритм, связанный с манипуляцией отдельными битами, неудобен для программной реализации, поэтому на практике используют видоизменённый алгоритм.

В некоторых [компьютерных играх](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5_%D0%B8%D0%B3%D1%80%D1%8B) (например, [Duke Nukem 3D](https://ru.wikipedia.org/wiki/Duke_Nukem_3D)) для успешного прохождения уровня требуется подобрать нужную комбинацию положений нескольких переключателей. Никаких подсказок нет, надо просто перебрать все комбинации. Для минимизации числа переключений при переборе вариантов следует использовать код Грея. Например, если переключателей три, пробуем их в порядке 000, 001, 011, 010, 110…

Сложные датчики, требующие синхросигнала, отходят от кода Грея и работают в обычном двоичном



**36. Обратная польская запись.**

**Обра́тная** **по́льская** **за́пись** (англ. Reverse Polish notation, RPN) — форма **записи** математических и логических выражений, в которой операнды расположены перед знаками операций. Также именуется как **обратная** бесскобочная **запись**, постфиксная нотация, бесскобочная символика Лукасевича, **польская** инверсная **запись**, ПОЛИЗ. Стековой машиной называется алгоритм, проводящий вычисления по **обратной** **польской** **записи.**

Обратная польская нотация (ОПН) была разработана [австралийским](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%81%D1%82%D1%80%D0%B0%D0%BB%D0%B8%D1%8F) [философом](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%BB%D0%BE%D1%81%D0%BE%D1%84%D0%B8%D1%8F) и специалистом в области [теории вычислительных машин](https://ru.wikipedia.org/w/index.php?title=%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D1%85_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD&action=edit&redlink=1) [Чарльзом Хэмблином](https://ru.wikipedia.org/w/index.php?title=%D0%A5%D1%8D%D0%BC%D0%B1%D0%BB%D0%B8%D0%BD,_%D0%A7%D0%B0%D1%80%D0%BB%D1%8C%D0%B7&action=edit&redlink=1) в середине [1950-х](https://ru.wikipedia.org/wiki/1950-%D0%B5) на основе [польской нотации](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%BD%D0%BE%D1%82%D0%B0%D1%86%D0%B8%D1%8F), которая была предложена в [1920 году](https://ru.wikipedia.org/wiki/1920_%D0%B3%D0%BE%D0%B4) польским математиком [Яном Лукасевичем](https://ru.wikipedia.org/wiki/%D0%9B%D1%83%D0%BA%D0%B0%D1%81%D0%B5%D0%B2%D0%B8%D1%87,_%D0%AF%D0%BD). Работа Хэмблина была представлена на конференции в июне [1957](https://ru.wikipedia.org/wiki/1957), и издана в 1957 и [1962](https://ru.wikipedia.org/wiki/1962).

Отличительной особенностью обратной польской нотации является то, что все аргументы (или [операнды](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D0%BD%D0%B4)) расположены перед знаком операции. В общем виде запись выглядит следующим образом:

* Запись набора операций состоит из последовательности операндов и знаков операций. Операнды в выражении при письменной записи разделяются пробелами.
* Выражение читается слева направо. Когда в выражении встречается знак операции, выполняется соответствующая операция над двумя последними встретившимися перед ним операндами в порядке их записи. Результат операции заменяет в выражении последовательность её операндов и её знак, после чего выражение вычисляется дальше по тому же правилу.
* Результатом вычисления выражения становится результат последней вычисленной операции.

Например, рассмотрим вычисление выражения 7 2 3 \* − (эквивалентное выражение в инфиксной нотации: 7 − 2 \* 3).

1. Первый по порядку знак операции — «\*», поэтому первой выполняется операция умножения над операндами 2 и 3 (они стоят последними перед знаком). Выражение при этом преобразуется к виду 7 6 − (результат умножения — 6, — заменяет тройку «2 3 \*»).
2. Второй знак операции — «−». Выполняется операция вычитания над операндами 7 и 6.
3. Вычисление закончено. Результат последней операции равен 1, это и есть результат вычисления выражения.

Очевидное расширение обратной польской записи на унарные, тернарные и операции с любым другим количеством операндов: при использовании знаков таких операций в вычислении выражения операция применяется к соответствующему числу последних встретившихся операндов.

Особенности обратной польской записи следующие:

* Порядок выполнения операций однозначно задаётся порядком следования знаков операций в выражении, поэтому отпадает необходимость использования скобок и введения приоритетов и ассоциативности операций.
* В отличие от инфиксной записи, невозможно использовать одни и те же знаки для записи унарных и бинарных операций. Так, в инфиксной записи выражение 5 \* (−3 + 8) использует знак «минус» как символ унарной операции (изменение знака числа), а выражение (10 − 15) \* 3 применяет этот же знак для обозначения бинарной операции (вычитание). Конкретная операция определяется тем, в какой позиции находится знак. Обратная польская запись не позволяет этого: запись 5 3 − 8 + \* (условный аналог первого выражения) будет интерпретирована как ошибочная, поскольку невозможно определить, что «минус» после 5 и 3 обозначает не вычитание; в результате будет сделана попытка вычислить сначала 5 − 3, затем 2 + 8, после чего выяснится, что для операции умножения не хватает операндов. Чтобы всё же записать это выражение, придётся либо переформулировать его (например, записав вместо выражения − 3 выражение 0 − 3), либо ввести для операции изменения знака отдельное обозначение, например, «±»: 5 3 ± 8 + \*.
* Так же, как и в инфиксной нотации, в ОПН одно и то же вычисление может быть записано в нескольких разных вариантах. Например, выражение (10 − 15) \* 3 в ОПН можно записать как 10 15 − 3 \*, а можно — как 3 10 15 − \*
* Из-за отсутствия скобок обратная польская запись короче инфиксной. За этот счёт при вычислениях на калькуляторах повышается скорость работы оператора (уменьшается количество нажимаемых клавиш), а в программируемых устройствах сокращается объём тех частей программы, которые описывают вычисления. Последнее может быть немаловажно для портативных и встроенных вычислительных устройств, имеющих жёсткие ограничения на объём памяти.

### Общий порядок

Автоматизация вычисления выражений в обратной польской нотации основана на использовании [стека](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA). Алгоритм вычисления для стековой машины элементарен:

1. Обработка входного символа
   * Если на вход подан операнд, он помещается на вершину стека.
   * Если на вход подан знак операции, то соответствующая операция выполняется над требуемым количеством значений, извлечённых из стека, взятых в порядке добавления. Результат выполненной операции кладётся на вершину стека.
2. Если входной набор символов обработан не полностью, перейти к шагу 1.
3. После полной обработки входного набора символов результат вычисления выражения лежит на вершине стека.

Реализация стековой машины, как программная, так и аппаратная, чрезвычайно проста и может быть очень эффективной. Обратная польская запись совершенно унифицирована — она принципиально одинаково записывает унарные, бинарные, тернарные и любые другие операции, а также обращения к функциям, что позволяет не усложнять конструкцию вычислительных устройств при расширении набора поддерживаемых операций. Это и послужило причиной использования обратной польской записи в некоторых научных и программируемых микрокалькуляторах.

### Пример вычисления выражений

Инфиксное выражение {\displaystyle (1+2)\times 4+3}в ОПН может быть записано так: 1 2 + 4 × 3 +

Вычисление производится слева направо, ввод интерпретируется как указано в приведённой ниже таблице (указано состояние стека после выполнения операции, вершина стека выделена красным цветом):

 Результат, 15, в конце вычислений находится на вершине стека.

Клавиша «Ввод» (обозначаемая на калькуляторах как «Enter» или символом «↑») выполняет роль разделителя операндов, когда два операнда непосредственно следуют друг за другом. Если за операндом следует [знак операции](https://ru.wikipedia.org/wiki/%D0%97%D0%BD%D0%B0%D0%BA%D0%B8_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B9), то её нажатие не требуется, это сокращает количество действий, которые нужно выполнить для получения результата.

### **Простой пример**

Вход: 3 + 4

Добавим 3 к выходной строке (если прочитано число, то оно сразу добавляется к выходной строке).

Помещаем + (или его Идентификатор) в стек операций.

Добавим 4 к выходной строке.

Мы прочитали всё выражение, теперь выталкиваем все оставшиеся в стеке операции в выходную строку. В нашем примере в стеке содержится только +.

Выходная строка: 3 4 +

В данном примере проявляются некоторые правила: все числа переносятся в выходную строку сразу после прочтения; когда выражение прочитано полностью, все оставшиеся в стеке операции выталкиваются в выходную строку.

### **Сложный пример**

[Приоритеты](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BE%D1%80%D0%B8%D1%82%D0%B5%D1%82_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8):

* самый высокий: выражения, заключённые в скобки ( )
* высокий: ^
* средний: \* /
* низкий: + −

Вход: 3 + 4 \* 2 / (1 - 5)^2

Читаем «3»

Добавим «3» к выходной строке

Выход: 3

Читаем «+»

Кладём «+» в стек

Выход: 3

Стек: +

Читаем «4»

Добавим «4» к выходной строке

Выход: 3 4

Стек: +

Читаем «\*»

Кладём «\*» в стек

Выход: 3 4

Стек: + \*

Читаем «2»

Добавим «2» к выходной строке

Выход: 3 4 2

Стек: + \*

Читаем «/»

Выталкиваем «\*» из стека в выходную строку, кладём «/» в стек

Выход: 3 4 2 \*

Стек: + /

Читаем «(»

Кладём «(» в стек

Выход: 3 4 2 \*

Стек: + / (

Читаем «1»

Добавим «1» к выходной строке

Выход: 3 4 2 \* 1

Стек: + / (

Читаем «−»

Кладём «−» в стек

Выход: 3 4 2 \* 1

Стек: + / ( −

Читаем «5»

Добавим «5» к выходной строке

Выход: 3 4 2 \* 1 5

Стек: + / ( -

Читаем «)»

Выталкиваем «−» из стека в выходную строку, выталкиваем «(»

Выход: 3 4 2 \* 1 5 −

Стек: + /

Читаем «^»

Кладём «^» в стек

Выход: 3 4 2 \* 1 5 −

Стек: + / ^

Читаем «2»

Добавим «2» к выходной строке

Выход: 3 4 2 \* 1 5 − 2

Стек: + / ^

Конец выражения

Выталкиваем все элементы из стека в строку

Выход: 3 4 2 \* 1 5 − 2 ^ / +

**37. АЦП и ЦАП.**

ЦАП – цифро-аналоговые преобразователи – устройства, предназначенные для преобразования дискретного (цифрового) сигнала в непрерывный (аналоговый) сигнал. Преобразование производится пропорционально двоичному коду сигнала.

АЦП – аналого-цифровые преобразователи – устройства, предназначенные для преобразования непрерывных (аналоговых) сигналов в цифровые.

ЦАП (цифро-аналоговый преобразователь) и АЦП (аналого-цифровой преобразователь) входят в состав кодека и непосредственно выполняют дискретизацию:

— при воспроизведении ЦАП переводит цифровой код в аналоговый сигнал,  
— при записи АЦП выполняет обратное преобразование.

Чем качественнее АЦП, тем более чистый и детальный звук будет литься из колонок.

Чем качественнее ЦАП, тем точнее будет переведен в цифру аналоговый сигнал.

В состав кодеков для поддержки многоканального звука входят по несколько ЦАП и АЦП.

Большинство датчиков и исполнительных устройств автоматиче­ских систем работает с аналоговыми сигналами. Для ввода таких сигна­лов в ЭВМ их необходимо преобразовать в цифровую форму, т.е. дискретизироватъ по уровню и во времени. Эту задачу решают АЦП. Обрат­ную задачу, т.е. превращение квантованного (цифрового) сигнала в не­прерывный, решают ЦАП.

АЦП и ЦАП являются основными устройствами ввода-вывода ин­формации в цифровых системах, предназначенных для обработки анало­говой информации или управления каким-либо технологическим процес­сом.

*Важнейшие характеристики АЦП и ЦАП:*

1) Вид аналоговой величины, являющейся входной для АЦП и выходной для ЦАП (напряжение, ток, временной интервал, фаза, частота, угловое и линейное перемещение, освещенность, давление, темпе­ратура и т.п.). Наибольшее распространение получили преобразо­ватели, в которых входной (выходной) аналоговой величиной явля­ется напряжение, т.к. большинство аналоговых величин сравни­тельно легко преобразуются в напряжение.

2) Разрешающая способность и точность преобразования (разре­шающая способность определяется количеством двоичных разрядов кода или возможным количеством уровней аналогового сигна­ла, точность определяется наибольшим значением отклонения аналогового сигнала от цифрового и наоборот).

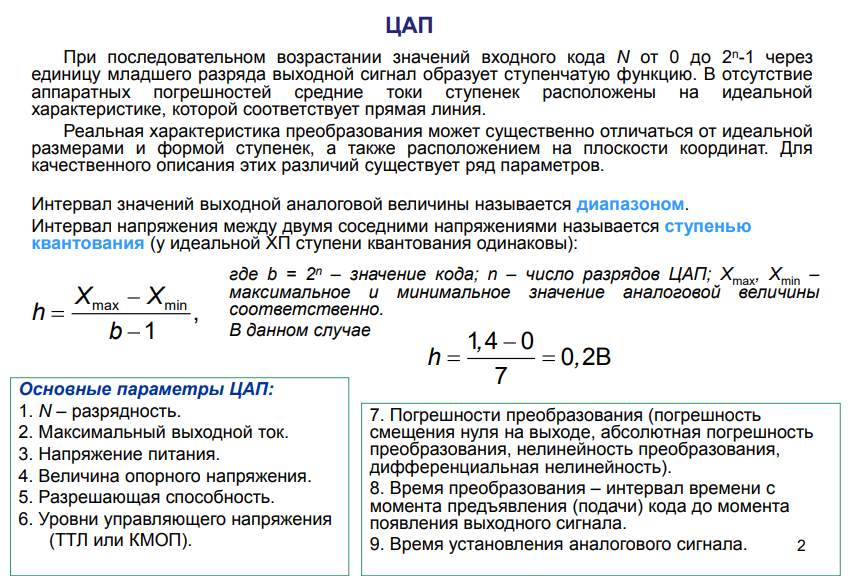
3) Быстродействие, определяемое интервалом времени от момента подачи сигнала опроса (запуска) до момента достижения выход­ным сигналом установившегося значения (ед. микросекунд, десят­ки наносекунд)

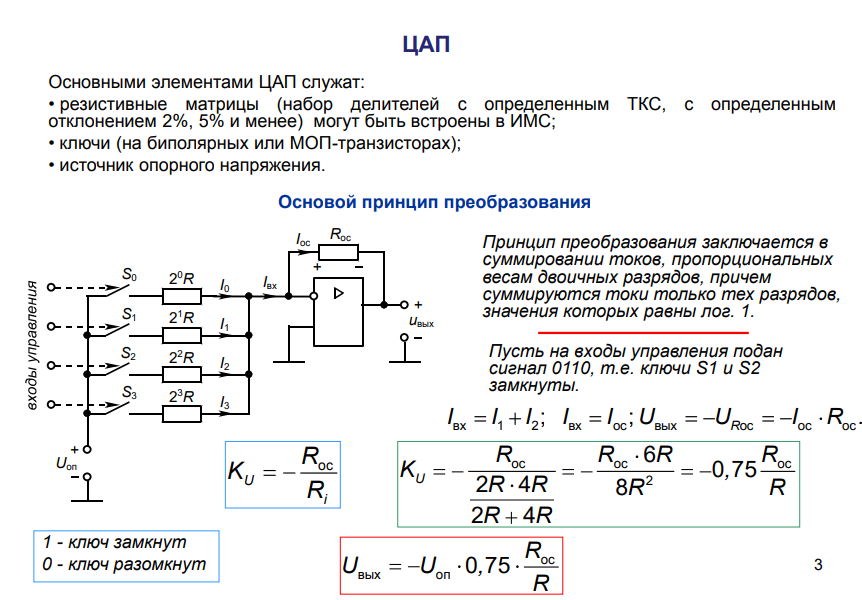
В любом преобразователе выделяют цифровую и аналоговую части. В цифровой производятся кодирование и декодирование цифровых сигна­лов, их запоминание, счет, цифровое компарирование (сравнение), выра­ботка логических сигналов управления. Для этого используют: дешифра­торы, мультиплексоры, регистры, счетчики, цифровые компараторы, логические элементы.

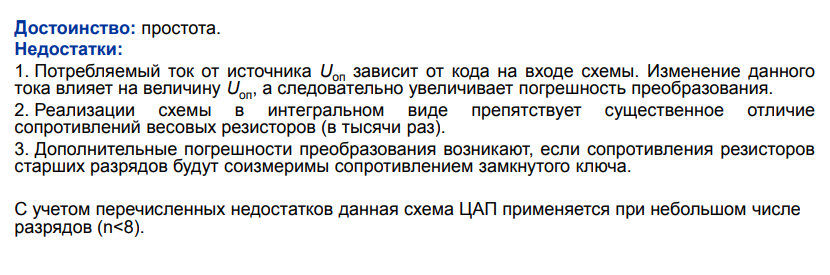
В аналоговой части преобразователя производятся операции: усиле­ния, сравнения, коммутации, сложения и вычитания аналоговых сигна­лов. Для этого используются аналоговые элементы: ОУ, аналоговые ком­параторы, ключи и коммутаторы, резистивные матрицы и т.д.

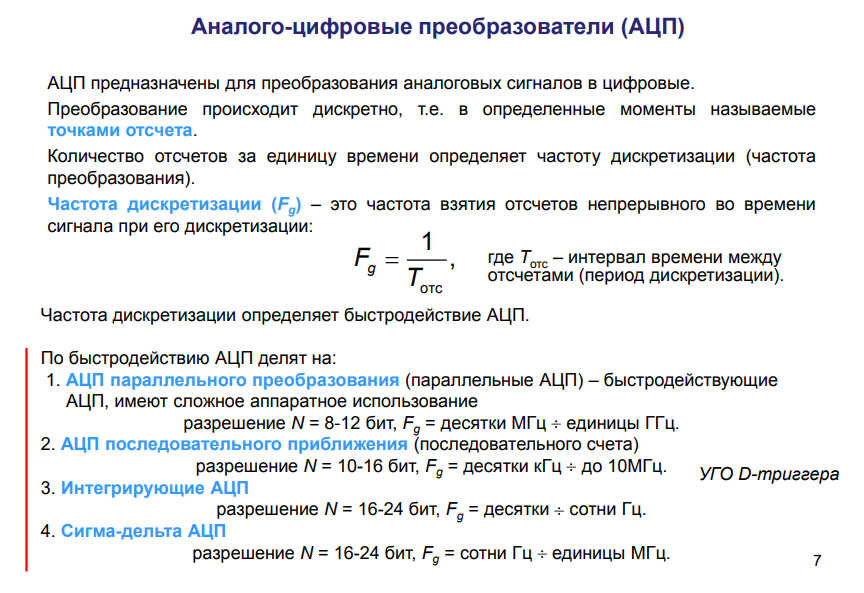
Преобразователи выполняются в виде цифровых и аналоговых ИМС или БИС.

****









**38. Корректирующие коды. Код Хэ́мминга. Область применения.**

**Корректирующий код** (также **помехоустойчивый код**) — [код](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4), предназначенный для [обнаружения и исправления ошибок](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C_%D0%BE%D1%88%D0%B8%D0%B1%D0%BE%D0%BA).

Основная техника — добавление при записи (передаче) в полезные данные специальным образом структурированной избыточной информации (например, [контрольного числа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE)), а при чтении (приёме) использование такой избыточной информации для обнаружения и исправления ошибки. Число ошибок, которое можно исправить, ограничено и зависит от конкретного применяемого кода.

Коды обнаружения ошибок (которые могут только установить факт ошибки) принадлежат к тем же классам кодов, что и коды, исправляющие ошибки. Фактически любой код, исправляющий ошибки, может быть также использован для обнаружения ошибок (при этом он будет способен обнаружить бо́льшее число ошибок, чем был способен исправить). Коды, исправляющие ошибки, применяются в системах [цифровой связи](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C), в том числе: спутниковой, радиорелейной, сотовой, передаче данных по телефонным каналам, а также в системах хранения информации, в том числе магнитных и оптических. Коды, обнаруживающие ошибки, применяются в [сетевых протоколах](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) различных [уровней](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_OSI).

**Коды Хэмминга** — вероятно, наиболее известный из первых самоконтролирующихся и самокорректирующихся кодов. Построены они применительно к двоичной системе счисления.

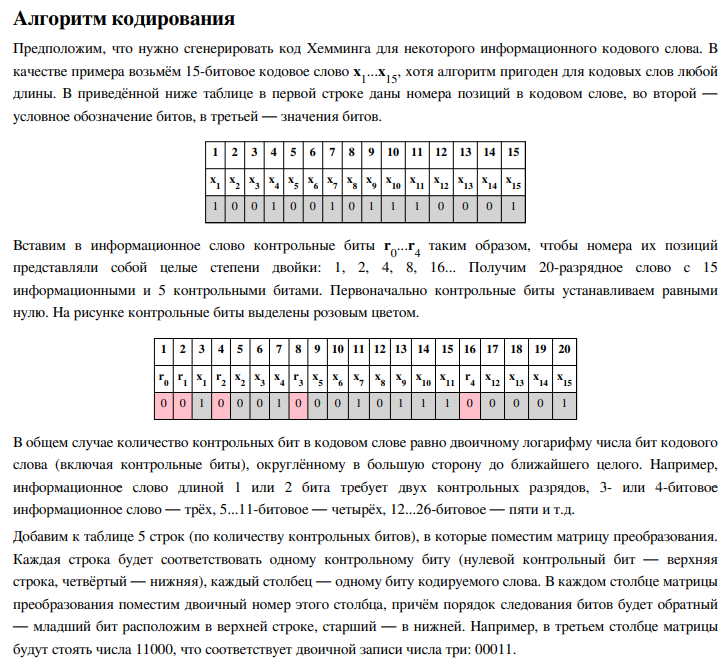
**История**

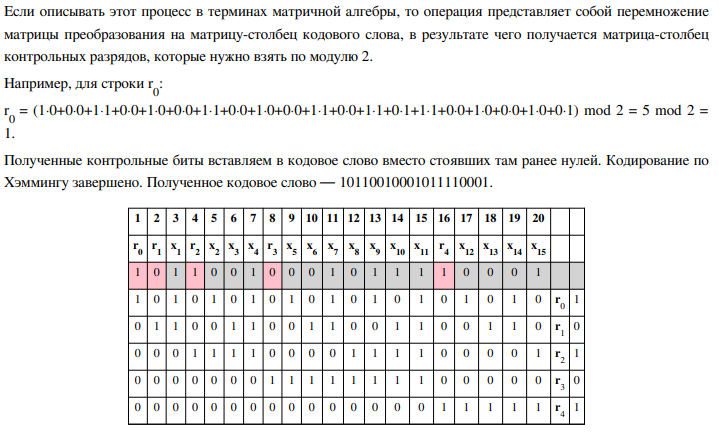
В середине 1940-х годов Ричард Хэмминг работал в знаменитых лабораториях фирмы Белл (Bell Labs) на счётной машине Bell Model V. Это была электромеханическая машина, использующая релейные блоки, скорость которых была очень низка: один оборот за несколько секунд. Данные вводились в машину с помощью перфокарт, поэтому в процессе чтения часто происходили ошибки. В рабочие дни использовались специальные коды, чтобы обнаруживать и исправлять найденные ошибки, при этом оператор узнавал об ошибке по свечению лампочек, исправлял и снова запускал машину. В выходные дни, когда не было операторов, при возникновении ошибки машина автоматически выходила из программы и запускала другую. Хэмминг часто работал в выходные дни, и все больше и больше раздражался, потому что часто был должен перезагружать свою программу из-за ненадежности перфокарт. На протяжении нескольких лет он проводил много времени над построением эффективных алгоритмов исправления ошибок. В 1950 году он опубликовал способ, который известен как код Хэмминга.



**Применение**

Код Хэмминга используется в некоторых прикладных программах в области хранения данных, особенно в RAID 2; кроме того, метод Хэмминга давно применяется в памяти типа ECC и позволяет «на лету» исправлять однократные и обнаруживать двукратные ошибки.





**39. Языки описания аппаратуры. ПЛИС (FPGA) модули.**

**Язык описания аппаратуры**(*HDL* от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *hardware description language*) — специализированный [компьютерный язык](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA), используемый для описания структуры и поведения [электронных схем](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%85%D0%B5%D0%BC%D0%B0), чаще всего [цифровых логических](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B2%D0%B5%D0%BD%D1%82%D0%B8%D0%BB%D1%8C) схем.

Языки описания аппаратуры внешне похожи на такие [языки программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), как [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) или [Паскаль](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%81%D0%BA%D0%B0%D0%BB%D1%8C_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), написанные на них программы также состоят из выражений, операторов, управляющих структур. Важнейшим отличием между обычными языками программирования и языками HDL является явное включение концепции времени в языки описания аппаратуры.

Языки описания аппаратуры являются неотъемлемой частью [САПР](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%B8), особенно для таких сложных схем, как [специализированные интегральные схемы](https://ru.wikipedia.org/wiki/ASIC), [микропроцессоры](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) и [программируемые логические устройства](https://ru.wikipedia.org/wiki/%D0%9F%D0%9B%D0%98%D0%A1).

Основные практически используемые языки описания аппаратуры — [Verilog](https://ru.wikipedia.org/wiki/Verilog) и [VHDL](https://ru.wikipedia.org/wiki/VHDL); также существует несколько десятков альтернативных языков.

ПЛИС (ПЛИС, англ. programmable logic device, PLD) — электронный компонент, используемый для создания цифровых интегральных схем. В отличие от обычных цифровых микросхем, логика работы ПЛИС не определяется при изготовлении, а задаётся посредством программирования.

Здесь под программируемостью понимается возможность изменения внутренней структуры ИС таким образом, чтобы она обеспечивала реализацию заданных функций алгебры логики (ФАЛ) на аппаратном уровне.

Структура ПЛИС основана на ФАЛ (булевой алгебры) и содержит следующие логические элементы: — логическое «И»(коньюнкторы); — логическое «ИЛИ» (дизъюнкторы); — логическое «НЕ» (инверторы); — буферные элементы с прямыми, инверсными и тристабильными выходами; — «Исключающее ИЛИ»; — триггеры D- и T-типа; — мультиплексоры конфигурации.

**Проще говоря,**

С чем ассоциируется цифровая электроника? В первую очередь с логическими элементами И, ИЛИ, НЕ. Далее в память приходят сдвиговые регистры, дешифраторы, мультиплексоры и прочее. Однако, по мере роста сложности электронных устройств и тенденции к миниатюризации, создание приборов на базе интегральных микросхем (ИМС) из вышеперечисленных компонентов затруднилось, заказные ИМС нужной топологии и схемотехники оправдывали себя только при крупном тиражировании устройства, в остальных случаях - это было неоправданно дорого.

Выходом из сложившейся ситуации стало развитие программируемых интегральных логических схем (сокращенно  - ПЛИС, зарубежная аббревиатура - programmable logic device, PLD).

**Отличия от микроконтроллеров**

При слове "программируемый" у большинства новичков происходит ассоциация [с микроконтроллерами](http://electrik.info/main/automation/549-chto-takoe-mikrokontrollery-naznachenie-ustroystvo-princip-raboty-soft.html). Не смотря на то что их также программируют, ПЛИС - это совершенно другое устройство.

Для разработчиков в микроконтроллерах доступен фиксированный набор решений и средств присущих конкретному чипу, отступить от архитектуры никаким образом не получится. Вам предоставлен набор команд, с помощью которых вы производите операции взаимодействия с окружающей средой, посредством считывания данных с цифровых и аналоговых входов и отправка сигналов исполнительным устройствам с помощью выходов.

Кроме этого вы можете производить вычисления, сохранять данные в регистрах или ПЗУ, а также оперировать данными прошитыми в память микроконтроллера. На этом в сущности и заключается назначение и особенности работы с микроконтроллерами.

Программируемые логические интегральные схемы (ПЛИС) отличаются тем, что, программируя устройство вы сами создаете архитектуру из базовых логических элементов. Таким образом вы получаете высокое быстродействию и гибкость микросхемы. Это даёт возможность, не изменяя одного чипа сделать целый ряд проектов.

Обобщённо внутреннее устройство ПЛИС можно разделить на три основных группы:

1. Массив из логических элементов (макроячеек, логических блоков).

2. Блоки входа-выхода (IO).

3. Линии связи между ними и устройство, которое управляет этими связями.

Однако такое структурирование очень обобщено, немного подробнее мы рассмотрим этот вопрос ниже.

Программируя вы соединяете элементы подобно тому как бы вы это делали, собирая из отдельных элементов устройство и соединяя их входы и выходы проводниками.

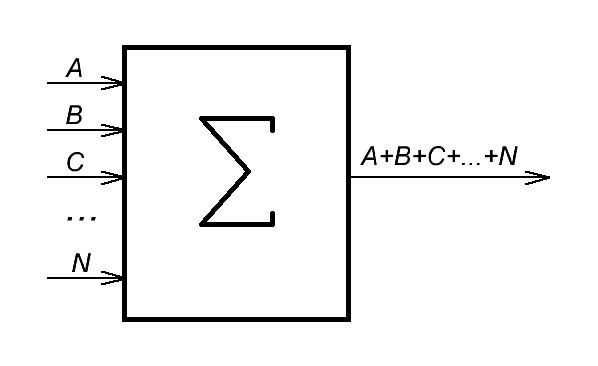
Примечание:

Главным отличием ПЛИС от микроконтроллеров является то, что в микроконтроллере вы не можете изменять внутренних связей между простейшими элементами, а в ПЛИС на основе прописывания связей основывается программирование и работа с ними.

\*\*\*Разработчикам, которые работают с ПЛИС доступно графическое программирование. То есть вы можете просто рисовать логические схемы или комбинировать код с графикой. Последнее называют модульным методом разработки, когда конкретные модули прописываются, а верхний модуль, в котором всё объединяется программируется графическим способом.

**40. Сумматор. Многоразрядный сумматор.**

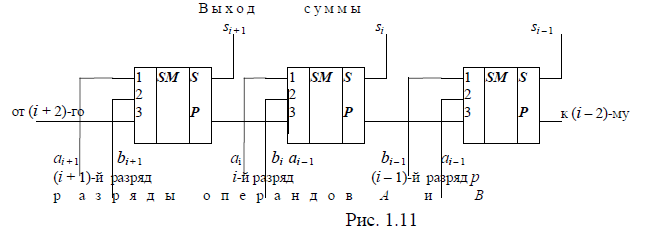
**Сумма́тор** — устройство, преобразующее [информационные сигналы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB) (аналоговые или цифровые) в сигнал, эквивалентный сумме этих сигналов[[1]](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%BC%D0%BC%D0%B0%D1%82%D0%BE%D1%80#cite_note-cyber-1); устройство, производящее операцию [сложения](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5). Т.е. сумматорами называют логические устройства, выполняющие арифметические суммирование кодов двоичного числа.



В электронике сумматоры делятся на две группы:

* *сумматоры аналоговых сигналов*
* *сумматоры цифровых сигналов*

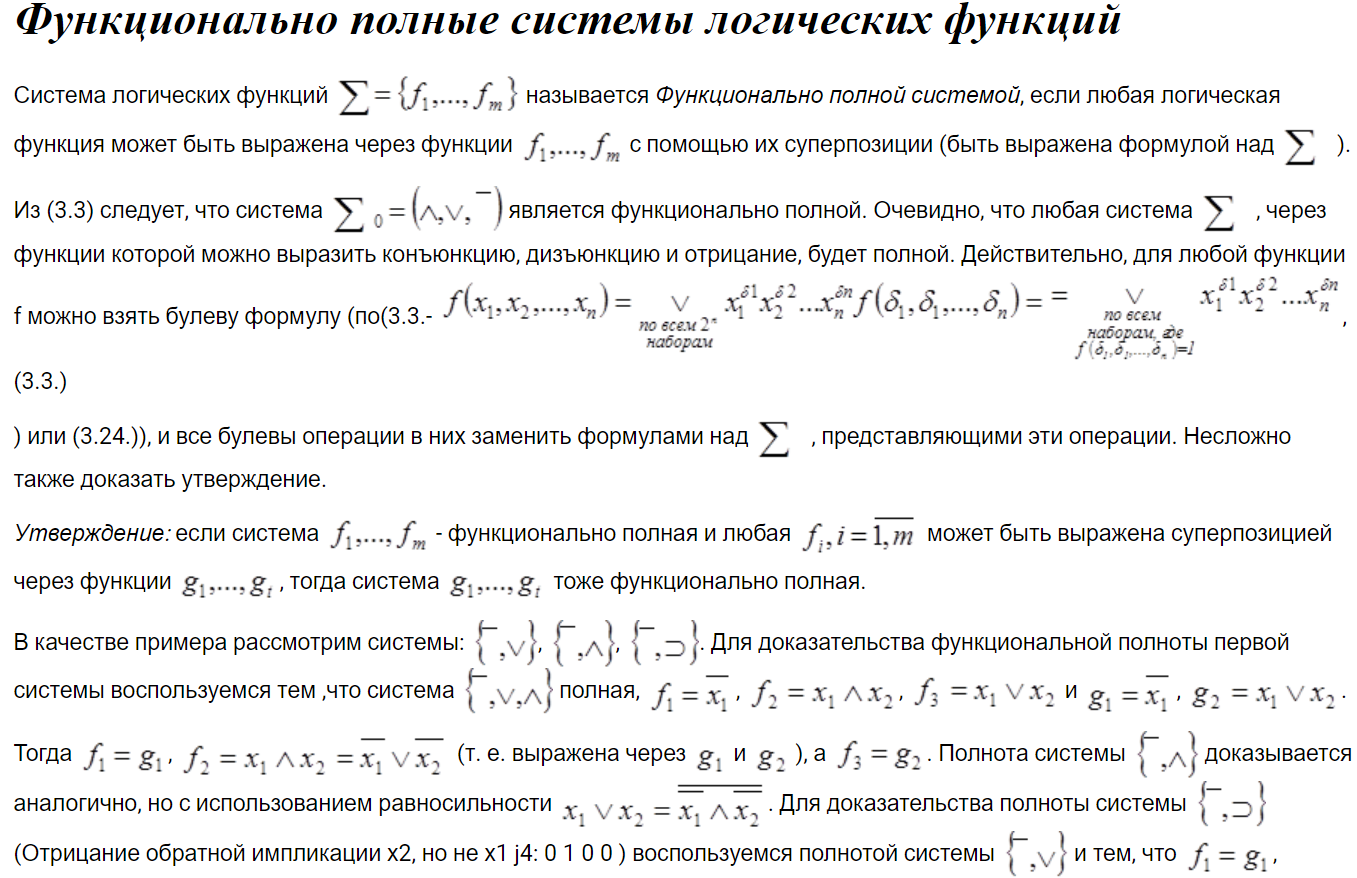
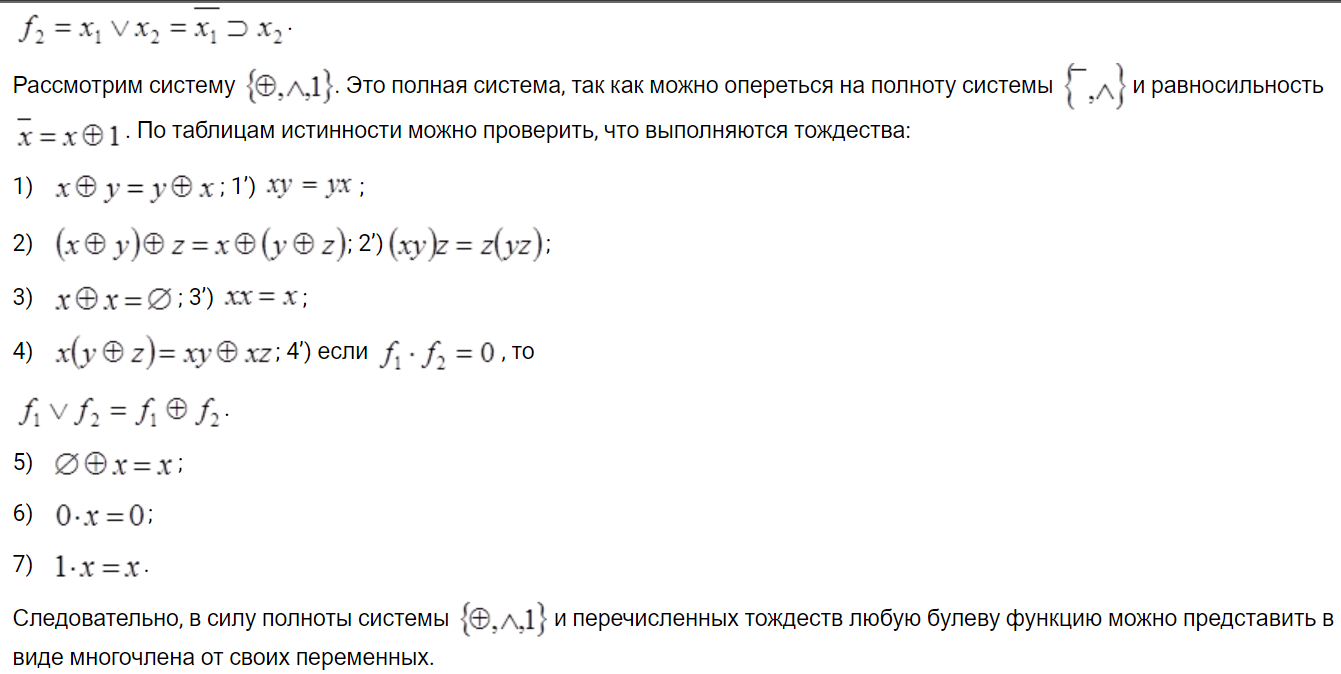
**Многоразрядный двоичный сумматор** строится на основе одноразрядных сумматоров с введением соответствующих связей между разрядами. На рис. 1.11 приведена простейшая схема такого сумматора. На схеме показана только часть сумматора, относящаяся к некоторому i-му разряду и его соседям: (i + 1)-й соседний младший разряд и (i - 1)-й соседний старший разряд.



Приведенная схема многоразрядного сумматора называется **схемой сумматора с последовательным переносом**. Схема очень простая. **Сумматор обладает малым быстродействием** из-за последовательного учета переноса, возникшего в младшем разряде, в непрерывной цепочке старших разрядов, имеющих значение поразрядной суммы, равное единице. Такие разряды называются разрядами, пропускающими перенос. В худшем случае перенос, возникший в младшем разряде, распространяется до самого старшего разряда формируемой суммы.

**41. Полная система логических функций.**

Функционально **полная** **система** **логических** элементов — это такой набор элементов, используя который можно реализовать любую сколь угодно сложную **логическую** **функцию**. Поскольку любая **логическая** **функция** представляет собой комбинацию простейших **функций** — дизъюнкции, конъюнкции и инверсии, то набор из элементов трех типов, реализующих соответственно **функции** И, или и НЕ, естественно, является функционально **полным**.

****

**Система** **функций** называется **полной**, если любую **логическую** **функцию** можно представить в виде формулы, содержащей **функции** этой **системы**. Имеет место теорема Поста: Для того чтобы некоторый набор **функций** был **полным**, необходимо и достаточно, чтобы в него входили **функции**, не принадлежащие каждому из классов Т0, Т1, L, M, S. Т0 – класс **функций** сохраняющих 0. В него входят **функции**, у которых f (0,0,...,0) = 0 . Т1 – класс **функций**, сохраняющих 1, т.е. таких у которых f (1,1,...,1) = 1**.**

**42. Искусство управления сложностью. Цифровая абстракция.**

Одной из характеристик, отличающих профессионального инженера-электронщика или программиста от дилетанта, является систематический подход к управлению сложностью многоуровневой системы. Современные цифровые системы построены из миллионов и миллиардов транзисторов. Человеческий мозг не в состоянии предсказать поведение подобных систем путем составления уравнений, описывающих движение каждого электрона в каждом транзисторе системы, и последующего решения этой системы уравнений. Для того, чтобы разработать удачный микропроцессор и не утонуть при этом в море избыточной информации, необходимо научиться управлять сложностью разрабатываемой системы.

Критически важный принцип управления сложностью системы – абстракция, подразумевающая исключение из рассмотрения тех элементов, которые в данном конкретном случае несущественны для понимания работы этой системы. Любую систему можно рассматривать с различных уровней абстракции. Политику, участвующему в выборах, например, нет нужды учитывать все детали окружающего его мира, ему достаточно абстрактной иерархической модели страны, состоящей из населенных пунктов, областей и федеральных округов. В области может быть несколько населенных пунктов, а федеральный округ включает в себя разные области. Если политик борется за пост президента, то его, скорее всего, интересует то, как проголосует федеральный округ в целом, при этом ему не обязательно знать, какое количество голосов он наберет в каждом конкретном населенном пункте этого округа. Для политика федеральный округ – это его уровень абстракции. С другой стороны, бюро переписи населения обязано знать количество жителей в каждом городе или поселке страны и потому должно оперировать на самом низком уровне абстракции данной системы – на уровне населенных пунктов. На Рис. 1.1 показаны уровни абстракции, типичные для любой электронной компьютерной системы вместе со строительными блоками, характерными для каждого уровня абстракции этой системы. На самом низком уровне абстракции находится физика, изучающая движение электронов. Поведение электронов описывается квантовой механикой и системой уравнений Максвелла.

****Конструкторская дисциплина – это преднамеренное ограничение самим конструктором выбора возможных вариантов разработки, что позволяет работать продуктивнее на более высоком уровне абстракции. Использование взаимозаменяемых частей – это, вероятно, самый хорошо знакомый всем нам пример практического применения конструкторской дисциплины. Одним из первых примеров использования взаимозаменяемых деталей и узлов стала унификация при производстве кремнёвых ружей. До начала 19-го века такие ружья производились вручную и в штучном порядке. Высококвалифицированный оружейный мастер тщательно подтачивал и подгонял комплектующие, произведенные несколькими не связанными друг с другом ремесленниками. Конструкторская дисциплина для обеспечения взаимозаменяемости деталей и узлов произвела революцию в оружейной промышленности. Ограничение ассортимента комплектующих деталей до стандартного набора с жестко установленными допусками для каждой детали позволило собирать и ремонтировать ружья гораздо быстрее и использовать при этом менее квалифицированный персонал. Оружейный мастер перестал тратить свое время на разрешение проблем, связанных с нижними уровнями абстракции, такими как доводка какого-то конкретного ствола или исправление формы отдельного взятого приклада.

В дополнение к абстрагированию от несущественных деталей и конструкторской дисциплине разработчики электронных систем используют еще три базовых принципа для управления сложностью системы: иерархичность, модульность конструкции и регулярность. Эти принципы применимы как к программному обеспечению, так и к аппаратной части компьютерных систем.

► Иерархичность – принцип иерархичности предполагает разделение системы на отдельные модули, а затем последующее разделение каждого такого модуля на фрагменты до уровня, позволяющего легко понять поведение каждого конкретного фрагмента.

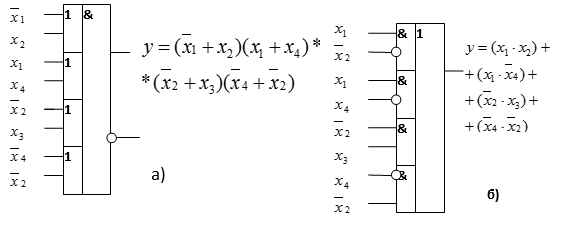
► Модульность – принцип модульности требует, чтобы каждый модуль в системе имел четко определенную функциональность и набор интерфейсов и мог быть легко и без непредвиденных побочных эффектов соединен с другими модулями системы.

► Регулярность – принцип регулярности требует соблюдения единообразия при проектировании отдельных модулей системы. Стандартные модули общего назначения, например, такие как блоки питания, могут использоваться многократно, во много раз снижая количество модулей, необходимых для разработки новой системы**.**

**43. Логические элементы. Таблицы истинности. Обозначения элементов в разных представления.**

Элементы ЭВМ, являющиеся представителями низшего иерархического уровня архитектуры компьютера – цифрового логического уровня (схемотехнического), реализованные на радиотехнических деталях, представляют собой мельчайшие компоненты, на основе которых строятся более крупные составляющие вычислительной машины.

- Можно выделить три основные разновидности элементов – логические элементы, запоминающие, специальные.

Логические элементы, так же как и элементы алгебры логики, реализуют логические функции, но эти функции, оставаясь сравнительно простыми, все же сложней, чем базовые функции в алгебре логики. В одном логическом элементе может быть реализовано несколько простых функций. Кроме того, логические элементы характеризуются дополнительными параметрами, такими, как количество входов, нагрузочная способность (количество входов других элементов, к которым можно подключать выход данного элемента).

На рис. приведены примеры некоторых логических элементов.

Абсолютно все цифровые микросхемы состоят из одних и тех же логических элементов – «кирпичиков» любого цифрового узла. Вот о них мы и поговорим сейчас.

**Логический элемент** – это такая схемка, у которой несколько входов и один выход. Каждому состоянию сигналов на входах, соответствует определенный сигнал на выходе.

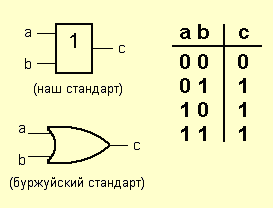
Итак, какие бывают элементы?

**Элемент «И» (AND)**

Иначе его называют «конъюнктор».

Для того, чтобы понять как он работает, нужно нарисовать таблицу, в которой будут перечислены состояния на выходе при любой комбинации входных сигналов. Такая таблица называется «**таблица истинности**». Таблицы истинности широко применяются в цифровой технике для описания работы логических схем.

Вот так выглядит элемент «И» и его таблица истинности:



Поскольку вам придется общаться как с русской, так и с буржуйской тех. документацией, я буду приводить условные графические обозначения (УГО) элементов и по нашим и по не нашим стандартам.

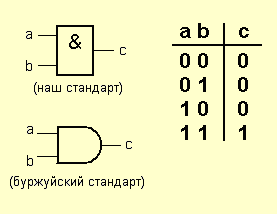
Смотрим таблицу истинности, и проясняем в мозгу принцип. Понять его не сложно: единица на выходе элемента «И» возникает только тогда, когда на оба входа поданы единицы. Это объясняет название элемента: единицы должны быть И на одном, И на другом входе.

Если посмотреть чуток иначе, то можно сказать так: на выходе элемента «И» будет ноль в том случае, если хотя бы на один из его входов подан ноль. Запоминаем. Идем дальше.

**Элемент «ИЛИ» (OR)**

По другому, его зовут «дизъюнктор».

Любуемся:

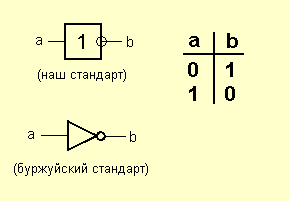


Опять же, название говорит само за себя.

На выходе возникает единица, когда на один ИЛИ на другой ИЛИ на оба сразу входа подана единица. Этот элемент можно назвать также элементом «И» для негативной логики: ноль на его выходе бывает только в том случае, если и на один и на второй вход поданы нули.

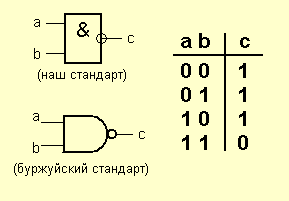
Едем дальше. Дальше у нас очень простенький, но очень необходимый элемент.

**Элемент «НЕ» (NOT)**

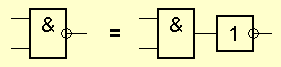
Чаще, его называют «инвертор».

Надо чего-нибудь говорить по поводу его работы?

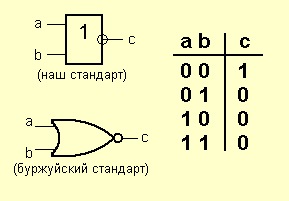
Ну тогда поехали дальше. Следующие два элемента получаются путем установки инвертора на выход элементов «И» и «ИЛИ».

**Элемент «И-НЕ» (NAND)**

Элемент И-НЕ работает точно так же как «И», только выходной сигнал полностью противоположен. Там где у элемента «И» на выходе должен быть «0», у элемента «И-НЕ» - единица. И наоборот. Э то легко понять по эквивалентной схеме элемента:

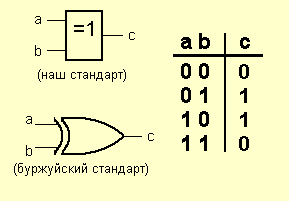


**Элемент «ИЛИ-НЕ» (NOR)**



Та же история – элемент «ИЛИ» с инвертором на выходе.

Следующий товарищ устроен несколько хитрее:  
**Элемент «Исключающее ИЛИ» (XOR)**

Он вот такой:

**Логический элемент** — элемент, осуществляющий определенные логические зависимость между входными и выходными сигналами. Логические элементы обычно используются для построения логических схем вычислительных машин, дискретных схем автоматического контроля и управления. Для всех видов логических элементов, независимо от их физической природы, характерны дискретные значения входных и выходных сигналов.

Логические элементы имеют один или несколько входов и один или два (обычно инверсных друг другу) выхода. Значения «нулей» и «единиц» выходных сигналов логических элементов определяются логической функцией, которую выполняет элемент, и значениями «нулей» и «единиц» входных сигналов, играющих роль независимых переменных. Существуют элементарные логические функции, из которых можно составить любую сложную логическую функцию.



**44. За пределами цифровой абстракции. Напряжение питания. Логические уровни. Допускаемые уровни шумов.**

**1.6 ЗА ПРЕДЕЛАМИ ЦИФРОВОЙ АБСТРАКЦИИ**

Цифровая система оперирует дискретными переменными. Однако для представления этих переменных используются непрерывные физические величины, такие как напряжение в электрической цепи, положение шестеренок в механической передаче или уровень жидкости в гидравлическом цилиндре. Задача разработчика цифровой системы – определить, каким образом непрерывно-меняющаяся величина соотносится с конкретным значением дискретной переменной. Рассмотрим, например, задачу представления двоичного сигнала А напряжением в электрической цепи. Допустим, что напряжение 0 В соответствует значению А = 0, а напряжение 5 В соответствует А = 1. Однако, реальная цифровая система должна быть устойчива к неизбежному в такой ситуации шуму, так что значение 4,97 В, вероятно, также следует толковать как A = 1. А что делать, если напряжение равно 4,3 В? Или 2,8 В? Или 2,500000 В?

**1.6.1 Напряжение питания**

Предположим, что минимальное напряжение в электронной цифровой системе, называемое также напряжением земли (ground voltage, или просто ground, или GND), составляет 0 В. Самое высокое напряжение в системе поступает от блока питания и, как правило, обозначается VDD. Транзисторные технологии семидесятых и восьмидесятых годов прошлого века в основном использовали VDD равное 5 В. С переходом на транзисторы меньшего размера, VDD последовательно снижали до 3,3 В, 2,5 В, 1,8 В, 1,5 В, 1,2 В и даже ниже для экономии электроэнергии и во избежание перегрузки транзисторов.

**1.6.2 Логические уровни.**

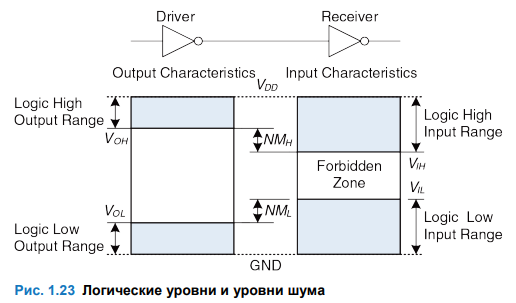
Отображение непрерывно-меняющейся переменной на различные значения дискретной двоичной переменной выполняется путем определения логических уровней, как показано на Рис. 1.23. Первый логический элемент в рассматриваемой схеме называется источник (driver), а второй – приемник (receiver). Выходной сигнал источника подключается ко входу приемника. Источник выдает выходной сигнал низкого напряжения (0) в диапазоне от 0 В до VOL или выходной сигнал высокого напряжения (1) в диапазоне от VOH до VDD. Если приемник получает на вход сигнал в диапазоне от 0 до VIL, он рассматривает такой сигнал как нуль. Если приемник получает на вход сигнал в диапазоне от VIH до VDD, он рассматривает такой сигнал как единицу. Если же по какой-либо причине, например, наличия шумов или неисправности одного из элементов схемы, напряжение сигнала на входе приемника падает настолько, что попадает в запретную зону (forbidden zone) между VIL и VIH, то поведение этого логического элемента становится непредсказуемым. VOH и VOL называются соответственно высоким и низким логическими уровнями выхода (output high and low logic levels), а VIH и VIL называются соответственно высоким и низким логическими уровнями входа (input high and low logic levels).

**1.6.3 Допускаемые Уровни Шумов**

Для того чтобы выходной сигнал источника был правильно интерпретирован на входе приемника, необходимо, чтобы VO < VIL и VOH > VIH. В этом случае, даже если выходной сигнал источника будет загрязнен шумами, приемник по-прежнему сможет правильно определить логический уровень входного сигнала. Допускаемый уровень шумов (noise margin) – это то максимальное количество шума, присутствие которого в выходном сигнале источника не мешает приемнику корректно интерпретировать значение полученного сигнала. Как можно увидеть на Рис. 1.23, значения нижнего допускаемого уровня шумов (low noise margin) и верхнего допускаемого уровня шумов (high noise margin) определяются следующим образом:

NML = VIL – VOL (1.2)

NMH = VO – VIH (1.3)

****

**45. Передаточная характеристика. Статическая дисциплина.**

**1. Передаточная Характеристика**

Для понимания предела цифровой абстракции мы должны рассмотреть поведение логических вентилей с аналоговой точки зрения.

Передаточная характеристика (DС transfer characteristics) какого-либо логического вентиля описывает напряжение на выходе этого элемента как функцию напряжения на его входе, когда входной сигнал изменяется настолько медленно, что выходной сигнал успевает изменяться вслед за ним. Такая характеристика называется передаточной, поскольку описывает взаимосвязь между входным и выходным напряжением.

В случае идеального инвертора переключение будет резким в точке

VDD/2. Для V(A) < VDD/2, V(Y) = VDD. Для

V(A) > VDD/2, V(Y) = 0. В этом случае, VIH = VIL = VDD/2. VOH = VDD и VOL = 0.

DC указывает на состояние, когда напряжение на входе электронной системы поддерживается постоянным или изменяется так медленно, что остальные параметры системы плавно изменяются вместе с ним. Исторически термин DC ведет свое происхождение от понятия постоянный ток (direct current) – метод передачи электрической энергии по схеме на расстояние, когда напряжение в линии поддерживается постоянным. В отличие от DC, переходная характеристика (transient response) схемы – это состояние, когда входное напряжение меняется быстро.Переходные процессы рассматриваются в **разделе 2.9**.

Напряжение при переключении реального инвертора изменяется постепенно между граничными значениями – так, как показано на **Рис. 1.25 (b)**. Если входное напряжение V(A) равно 0, то напряжение на

выходе V(Y) = VDD. Если V(A) = VDD, то V(Y) = 0. Однако, переход между этими конечными точками плавный и может находиться правее или

левее значения VDD/2. В связи с этим, возникает закономерный вопрос, как в этом случае определить логические уровни.

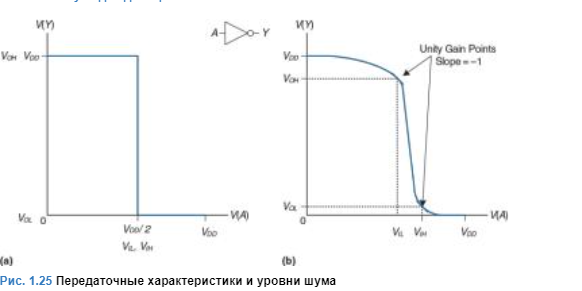
Разумно выбрать в качестве логических уровней те две точки, где наклон передаточной характеристики dV(Y)/dV(A) равен −1. Такие точки называются граничные коэффициенты передачи (unity gain points).

Подобный выбор обычно максимизирует допускаемые уровни шумов. При уменьшении VIL VOH увеличивается незначительно. Однако, если VIL растет, VOH падает практически отвесно.

**2. Статическая Дисциплина**

Для того, чтобы избежать попадания входных сигналов в запретные зоны, логические вентили должны разрабатываться в соответствии с

*принципом статической дисциплины (static discipline). Принцип статической дисциплины требует, чтобы при условии наличия логически корректных сигналов на входе каждый элемент системы выдавал логически корректные сигналы на выходе.*



Применение принципа статической дисциплины ограничивает свободу разработчика в выборе аналоговых элементов для построения цифровых систем, однако помогает обеспечить простоту и надежность разрабатываемых цифровых схем. Используя этот принцип, разработчик поднимается с аналогового уровня абстракции на цифровой, что увеличивает производительность проектировщика, избавляя его от рассмотрения излишних деталей. Выбор *VDD*и логических уровней может быть произвольным, однако этот выбор должен обеспечить совместимость всех логических вентилей, обменивающихся данными в пределах одной цифровой системы. Поэтому вентили обычно группируются в *семейства логики (logic families)*таким образом, что любой элемент из одного семейства при соединении с любым другим элементом из этого же семейства автоматически обеспечивает соблюдение принципа статической дисциплины. Логические вентили одного семейства соединяются друг с другом так же легко, как и блоки конструктора Лего, поскольку они полностью совместимы по напряжению источника питания и логическим уровням.

Четыре основные семейства логических вентилей доминировали с 70-х

*по 90-е годы прошлого века – это ТТЛ – транзисторно-транзисторная логика (Transistor-Transistor Logic, или TTL), КМОП – логика,*

*построенная на комплементарной структуре металл-оксид-*

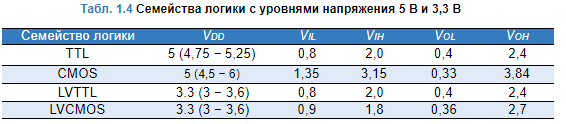
*полупроводник (Complementary Metal-Oxide-Semiconductor Logic, или*

*CMOS), НТТЛ – низковольтная транзисторно-транзисторная логика*

*(Low-Voltage Transistor-Transistor Logic, или LVTTL) и НКМОП низковольтная логика на комплементарной структуре металл-*

*оксид-полупроводник (Low-Voltage Complementary Metal-Oxide- Semiconductor Logic, или LVCMOS). Логические уровни для всех этих семейств представлены в****Табл. 1.4****. Начиная с 90-х годов прошлого*

века, четыре вышеперечисленных семейства распались на большое количество более мелких семейств в связи со все большим распространением устройств, требующих еще более низкого напряжения питания. В **приложении A.6**наиболее распространённые семейства логических вентилей рассматриваются детально.

****

**46. Биполярные и КМОП транзисторы. Полупроводники. Конденсаторы. n-МОП и p-МОП-транзисторы.**

**Биполярный транзистор***– это полупроводниковый прибор с двумя взаимодействующими между собой р-п–переходами и тремя или более выводами.* Полупроводниковый кристалл транзистора состоит из трех областей с чередующимися типами электропроводности, между которыми находятся два *р-п*-перехода. Средняя область обычно выполняется очень тонкой (доли микрона), поэтому *р-п*-переходы близко расположены один от другого.

**Биполя́рный транзи́стор** — трёхэлектродный [полупроводниковый прибор](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%83%D0%BF%D1%80%D0%BE%D0%B2%D0%BE%D0%B4%D0%BD%D0%B8%D0%BA%D0%BE%D0%B2%D1%8B%D0%B5_%D0%BF%D1%80%D0%B8%D0%B1%D0%BE%D1%80%D1%8B), один из типов [транзисторов](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B8%D1%81%D1%82%D0%BE%D1%80). В полупроводниковой структуре сформированы два [p-n-перехода](https://ru.wikipedia.org/wiki/P-n-%D0%BF%D0%B5%D1%80%D0%B5%D1%85%D0%BE%D0%B4), перенос [заряда](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B7%D0%B0%D1%80%D1%8F%D0%B4) через которые осуществляется носителями двух полярностей — [электронами](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD) и [дырками](https://ru.wikipedia.org/wiki/%D0%94%D1%8B%D1%80%D0%BA%D0%B0). Именно поэтому прибор получил название «биполярный» (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *bipolar*), в отличие от [полевого (униполярного) транзистора](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5%D0%B2%D0%BE%D0%B9_%D1%82%D1%80%D0%B0%D0%BD%D0%B7%D0%B8%D1%81%D1%82%D0%BE%D1%80).

**КМОП** (К-МОП; комплементарная логика на [транзисторах](https://science.fandom.com/ru/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B8%D1%81%D1%82%D0%BE%D1%80) металл-оксид-полупроводник; [англ.](https://science.fandom.com/ru/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *CMOS, Complementary-symmetry/metal-oxide semiconductor*) — технология построения электронных схем. В технологии КМОП используются полевые транзисторы с изолированным затвором с каналами разной проводимости. Отличительной особенностью схем КМОП по сравнению с биполярными технологиями (ТТЛ, ЭСЛ и др.) является очень малое энергопотребление в статическом режиме (в большинстве случаев можно считать, что энергия потребляется только во время переключения состояний). Отличительной особенностью структуры КМОП по сравнению с другими МОП-структурами (N-МОП, P-МОП) является наличие как n-, так и p-канальных полевых транзисторов; как следствие, КМОП-схемы обладают более высоким быстродействием и меньшим энергопотреблением, при том характеризуются менее сложным технологическим процессом изготовления и меньшей стоимостью.

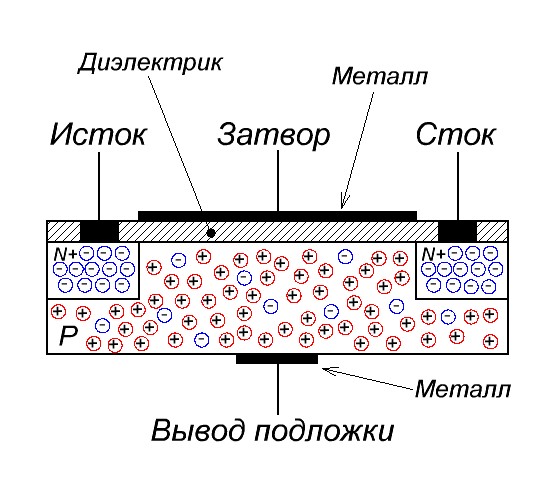
**Полупроводни́к** — [материал](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D1%80%D0%B8%D0%B0%D0%BB), по [удельной проводимости](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B2%D0%BE%D0%B4%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D1%8C) занимающий промежуточное место между [проводниками](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B2%D0%BE%D0%B4%D0%BD%D0%B8%D0%BA_(%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D1%82%D0%B2%D0%BE)) и [диэлектриками](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%B8%D0%BA), и отличающийся от проводников сильной зависимостью [удельной проводимости](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B2%D0%BE%D0%B4%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D1%8C) от концентрации примесей, температуры и воздействия различных видов [излучения](https://ru.wikipedia.org/wiki/%D0%98%D0%B7%D0%BB%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Основным свойством полупроводников является увеличение электрической проводимости с ростом температуры[[1]](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%83%D0%BF%D1%80%D0%BE%D0%B2%D0%BE%D0%B4%D0%BD%D0%B8%D0%BA#cite_note-1).

Полупроводниками являются кристаллические вещества, [ширина запрещённой зоны](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%80%D0%B8%D0%BD%D0%B0_%D0%B7%D0%B0%D0%BF%D1%80%D0%B5%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B9_%D0%B7%D0%BE%D0%BD%D1%8B) которых составляет порядка [электрон-вольта](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD-%D0%B2%D0%BE%D0%BB%D1%8C%D1%82) (эВ). Например, [алмаз](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%BC%D0%B0%D0%B7) можно отнести к *широкозонным полупроводникам* (около 7 эВ), а [арсенид индия](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%81%D0%B5%D0%BD%D0%B8%D0%B4_%D0%B8%D0%BD%D0%B4%D0%B8%D1%8F) — к *узкозонным* (0,35 эВ). К числу полупроводников относятся многие химические элементы (германий, кремний, [селен](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BB%D0%B5%D0%BD), теллур, мышьяк и другие), огромное количество сплавов и химических соединений (арсенид галлия и др.).

**Конденсатор** или как в народе говорят — «кондер», образуются от латинского «condensatus», что означает как «уплотненный, сгущенный». Он представляет из себя пассивный радиоэлемент, который обладает таким свойством, как сохранение электрического заряда на своих обкладках, если, конечно, перед этим его зарядить каким-нибудь источником питания.

Грубо говоря, конденсатор можно рассматривать как батарейку или аккумулятор электрической энергии. Но вся разница в том, что аккумулятор или батарейка имеют в своем составе источник [ЭДС](https://www.ruselectronic.com/eds-elektrodvizhushchaya-sila/), тогда как конденсатор лишен этого внутреннего источника.

Как часто вы слышали название полевой транзистор***МОП, MOSFET, MOS****, полевик, МДП-транзистор, транзистор с изолированным затвором*? Это все слова синонимы и относятся к одному и тому же радиоэлементу: полевому МОП-транзистору.

Полное название такого радиоэлемента на английский манер звучит как **M**etal **O**xide **S**emiconductor **F**ield **E**ffect **T**ransistors (MOSFET), что в дословном переводе **М**еталл **О**ксид **П**олупроводник Поле Влияние Транзистор. Если преобразовать на наш могучий русский язык, то получается как *полевой транзистор со структурой Металл Оксид Полупроводник* или просто *МОП-транзистор*. Почему МОП-транзистор также называют МДП-транзистором и транзистором с изолированным затвором.

Если «разрезать» МОП-транзистор, то можно увидеть вот такую картину:

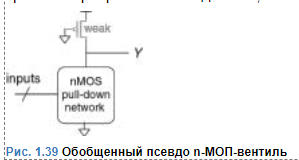
С точки зрения еды на вашем столе, МОП-транзистор будет больше похож на бутерброд. [Полупроводник P-типа](https://www.ruselectronic.com/bipolyarniy-tranzistor/#P_N) — толстый кусок хлеба, диэлектрик — тонкий слой колбасы, слой металла — тонкая пластинку сыра. В результате у нас получается вот такой бутерброд.

**47. Логический вентиль НЕ и другие на КМОП-транзисторах. Псевдо n-МОП-Логика Потребляемая Мощность.**

**Логи́ческий ве́нтиль** — базовый элемент цифровой схемы, выполняющий элементарную логическую операцию, преобразуя таким образом множество входных логических сигналов в выходной логический сигнал. Логика работы вентиля основана на [битовых операциях](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D1%8B%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8) с входными цифровыми сигналами в качестве [операндов](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D0%BD%D0%B4). При создании цифровой схемы вентили соединяют между собой, при этом выход используемого вентиля должен быть подключён к одному или к нескольким входам других вентилей. В настоящее время в созданных человеком цифровых устройствах доминируют электронные логические вентили на базе [полевых транзисторов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5%D0%B2%D0%BE%D0%B9_%D1%82%D1%80%D0%B0%D0%BD%D0%B7%D0%B8%D1%81%D1%82%D0%BE%D1%80), однако в прошлом для создания вентилей использовались и другие устройства, например, электромагнитные [реле](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D0%B5), гидравлические устройства, а также механические устройства. В поисках более совершенных логических вентилей исследуются квантовые устройства[[3]](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B2%D0%B5%D0%BD%D1%82%D0%B8%D0%BB%D1%8C#cite_note-3)[[4]](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B2%D0%B5%D0%BD%D1%82%D0%B8%D0%BB%D1%8C#cite_note-4), [биологические](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BB%D0%B5%D0%BA%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F) [молекулы](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BB%D0%B5%D0%BA%D1%83%D0%BB%D0%B0), [фононные](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D0%BD%D0%BE%D0%BD) тепловые системы.

|  |  |  |
| --- | --- | --- |
| **1. Псевдо n-МОП-Логика** | |  |
| Построенный | по технологии КМОП | логический вентиль ИЛИ-НЕ, |
| у которого число входных контактов равно N, использует N параллельно | | |
| включенных | n-МОП-транзисторов и N | последовательно включенных |

p-МОП-транзисторов. Последовательно включенные транзисторы передают сигнал медленнее, чем транзисторы, включенные параллельно, аналогично тому, как сопротивление резисторов, включенных последовательно, будет больше, чем сопротивление резисторов, включенных параллельно. Кроме того, p-МОП-транзисторы передают сигналы медленнее, чем n-МОП-транзисторы, поскольку дырки не могут перемещаться по кристаллической решетке кремния так же быстро, как электроны. В результате, соединенные параллельно n-МОП-транзисторы работают быстро, а соединенные последовательно p-МОП-транзисторы работают медленно, особенно если их много.

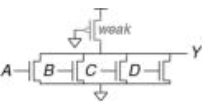


Как показано на **Рис. 1.39**, при использовании *псевдо n-МОП-логики (pseudo-nMOS logic)*, или просто *псевдо-логики*, медленный стек из p-МОП-транзисторов заменяют одним «слабым» p-МОП-транзистором, который всегда находится во включенном состоянии. Такой транзистор часто называют *слабым подтягивающим транзистором (weak pull-up)*.

Физические параметры p-МОП-транзистора подбираются таким образом, что этот транзистор до высокого логического уровня (1) выход Y «подтягивает слабо» – то есть только в том случае, когда все n-МОП-транзисторы выключены. Но если при этом хотя бы один из n-МОП-транзисторов включается, то он, превосходя по мощности слабый подтягивающий транзистор, «перетягивает» выход Y настолько близко к напряжению земли GND, что на выходе получается логический 0.

Преимущество псевдо-логики заключается в том, что такую логику можно использовать для создания быстрых ИЛИ-НЕ вентилей с большим количеством входов. Например, на **Рис. 1.40**показан вентиль ИЛИ-НЕ с четырьмя входами, построенный с использованием псевдо-логики.

Логические вентили, использующие псевдо-логику, могут быть очень полезны для построения некоторых видов памяти и логических массивов, описанных в Главе 5. Недостаток псевдо-логики – наличие короткого замыкания между питанием *VDD*и землей GND, когда сигнал на выходе – это логический нуль (0). Слабые p-МОП- и n-МОП-транзисторы выключены. При этом, через короткое замыкание постоянно протекает ток, и электрическая энергия от источника питания расходуется впустую. Именно по этой причине псевдо-n-МОП-логика используется ограниченно.



Термин «псевдо-n-МОП-логика» родился в 70-тые годы прошлого века. Тогда существовал производственный процесс для изготовления только n-МОП-транзисторов. В то время слабые n-МОП-транзисторы использовались для «подтягивания» выходного сигнала до логической единицы (1), поскольку p-МОП-транзисторов просто не было.

**2.ПОТРЕБЛЯЕМАЯ МОЩНОСТЬ**

*Потребляемая*мощность – это количество энергии, потребляемой системой в единицу времени. Энергопотребление имеет большое значение в цифровых системах. Именно потребляемая мощность определяет время автономной работы без подзарядки батареи любого портативного устройства, такого как сотовый телефон или ноутбук. Не стоит думать, однако, что потребляемая мощность – второстепенный параметр для стационарных устройств. Электричество стоит денег, и к тому же любое устройство может перегреться, если оно потребляет слишком много электроэнергии.

Цифровая система потребляет энергию как в динамическом режиме, когда выполняет какие-либо операции, так и в статическом, когда система находится в состоянии покоя *(idle)*. В динамическом режиме энергия расходуется на зарядку емкостей элементов системы, когда эти элементы переключаются между 0 и 1. И хотя в статическом режиме никаких переключений не происходит, система все равно расходует электрическую энергию.

И сами логические вентили, и проводники, соединяющие эти вентили друг с другом, являются конденсаторами и обладают определенной емкостью. Энергия, получаемая от блока питания, которую необходимо затратить на зарядку емкости C до напряжения *VDD*, равна *C*×*VDD*2. Если напряжение на конденсаторе переключается с частотой *f*(т.е. *f*раз в секунду), то конденсатор заряжается *f*/2 раза и разряжается *f*/2 раза в секунду. И, поскольку в процессе разрядки конденсатор не потребляет энергию от источника питания, то получается, что потребление энергии в динамическом режиме можно рассчитать как:



Утечка тока в системе происходит, даже если система находится в состоянии покоя. У некоторых типов электронных схем, таких как псевдо n-МОП-логика, существует путь, соединяющий напряжение питания *VDD*с землей GND, через который ток протекает постоянно. Суммарная величина тока, протекающего в системе в ее статическом состоянии *IDD*, называется *током утечки*

*(leakage current) или током покоя (quiescent supply current). Мощность,*

потребляемая системой в статическом состоянии, пропорциональна величине тока утечки и может быть рассчитана как:



(пример 1.23)

Сотовый телефон некоторой модели имеет аккумулятор емкостью 6 Ватт-часов и работает от напряжения 1,2 вольта. Предположим, что во время использования телефон работает на частоте 300 МГц и средняя емкость цифровой схемы телефона в любой конкретный момент составляет 10 нФ (10-8 Фарады). При работе телефон также выдает сигнал мощностью 3 Вт на антенну. Когда телефон не используется, динамическая потребляемая мощность падает практически до нуля, так как обработка сигналов отключена. Но телефон также потребляет 40 мА тока покоя в любом случае, работает он или нет. Рассчитайте время, на которое хватит аккумулятора телефона, для случаев:

(а) Если телефон не используется (б) Если телефон используется непрерывно

**Решение:**Статическая мощность *P*static равна (0,040 А)\*(1,2 В) = 48 мВт. Если телефон не используется, это единственное потребление мощности, поэтому время жизни аккумулятора равно (6 Ватт-часов)/(0,048 Вт) = 125 часов (примерно 5 дней). В случае, если телефон используется, динамическая

мощность *P*dynamic равна (0,5)\*(10–8 Ф)\*(1,2 В)2 \*(3 \* 108 Гц) = 2,16 Вт. Общая мощность, являющаяся суммой *P*dynamic, *P*static и мощности вещания, составит 2,16 Вт + 0,048 Вт + 3 Вт = 5,2 Вт, поэтому время жизни аккумулятора будет

равно 6 Ватт-часов/5,2 Вт = 1,15 часа. В этом примере фактическая работа телефона представлена в несколько упрощенном виде, но тем не менее, он иллюстрирует ключевые идеи, касающиеся мощности потребления.

**48. Проектирование комбинационной логики. От логики к логическим элементам. Что такое Х и Z: способы сопряжения микросхем в ЭВМ.**

Принципиальная схема – это изображение цифровой схемы, показывающее элементы и соединяющие их проводники. Например, схема на Рис. 2.23 показывает возможную аппаратную реализацию нашей любимой логической функции.

****

Изображая принципиальные схемы в унифицированном виде, нам становится легче читать их и отлаживать. В большинстве случаев мы будем придерживаться следующих правил:

-Входы изображаются на левой (или верхней) части схемы;

-Выходы изображаются на правой (или нижней) части схемы;

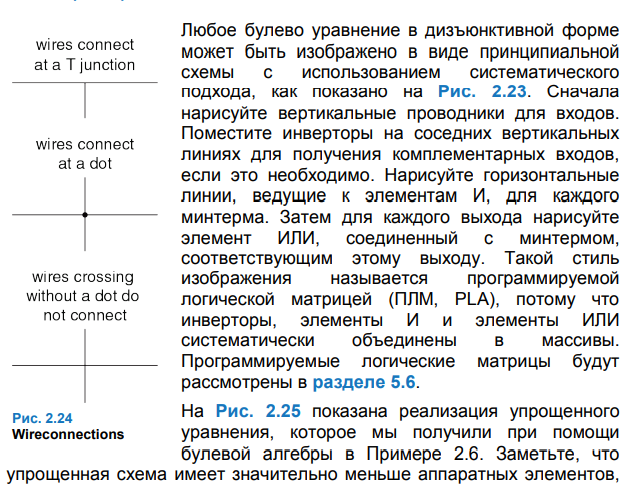
-Всегда, когда это возможно, элементы необходимо изображать слева направо;

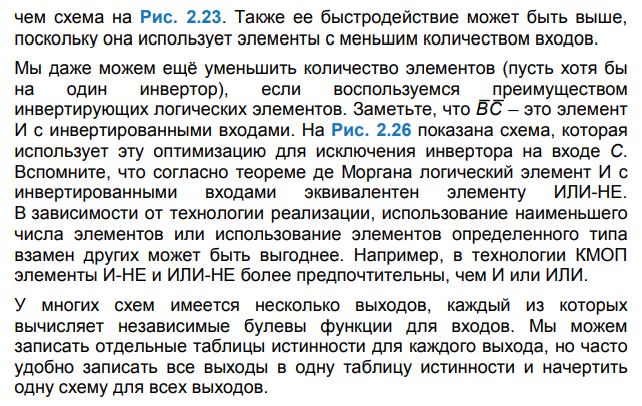
-Проводники лучше изображать прямыми линиями, чем линиями с множеством углов (неровные рваные линии отвлекают внимание: приходится следить за тем, куда ведут провода, а не думать о том, что делает схема);

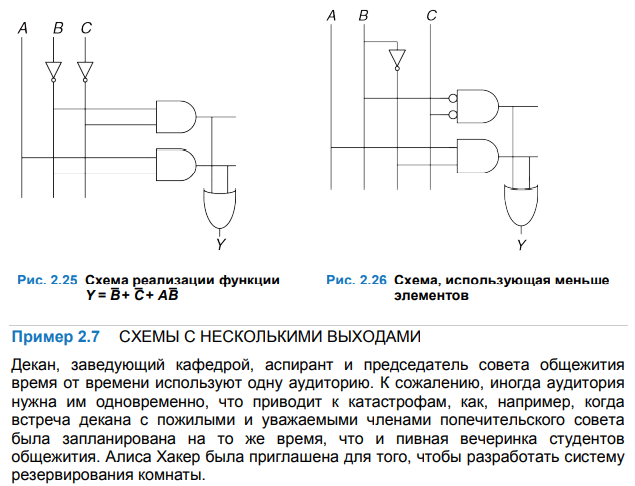
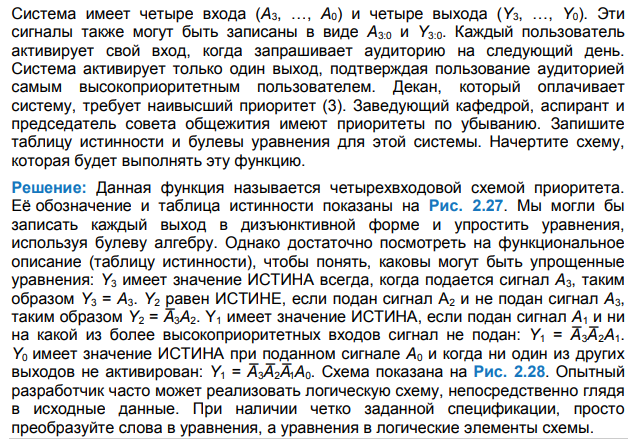
-Проводники всегда должны соединяться в виде буквы «Т»;

-Точка в месте пересечения проводников обозначает их соединение;

- Проводники, пересекающиеся без точки, не имеют соединения друг с другом. Три последних правила показаны на Рис. 2.24

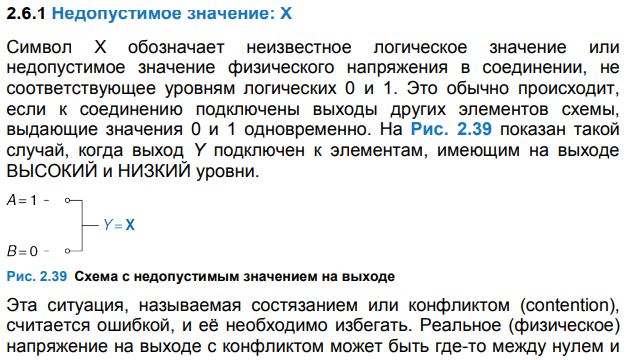
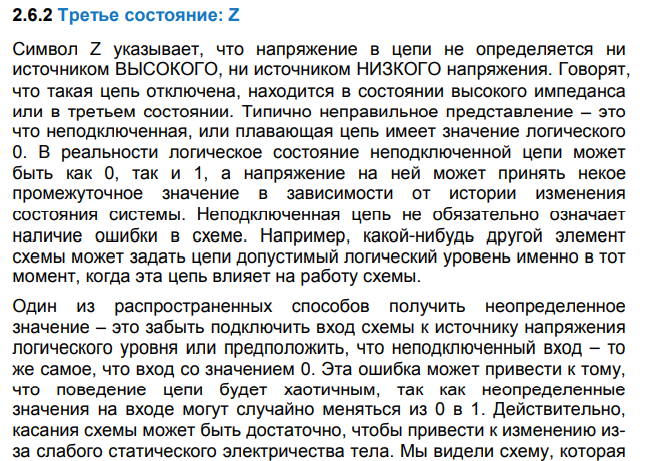
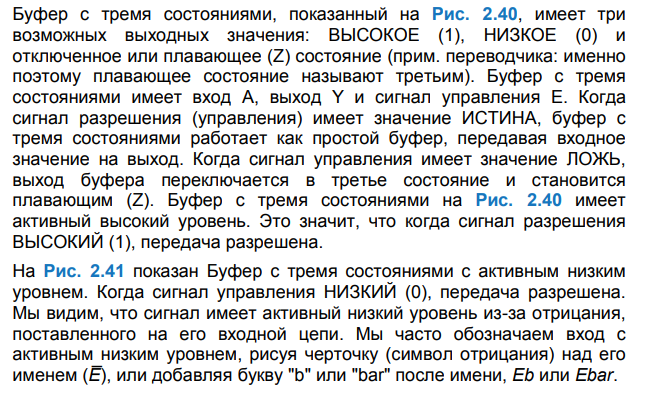
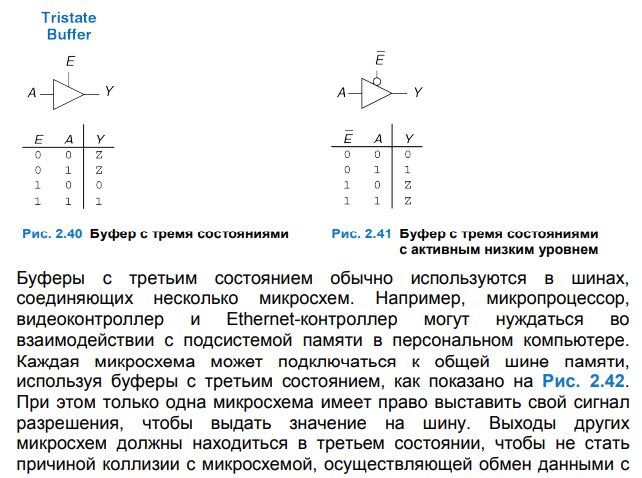
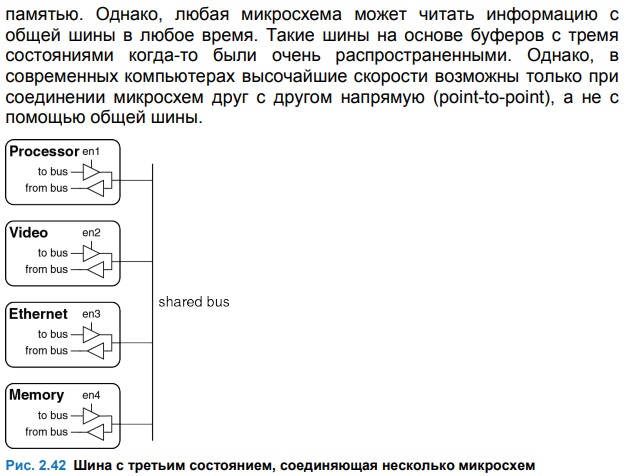
****

****

****

ЧТО ЗА **X И Z**?

Булева алгебра ограничена значениями 0 и 1. Однако реальные схемы могут также иметь недопустимое и плавающее состояния, представляемые символами X и Z соответственно.

****

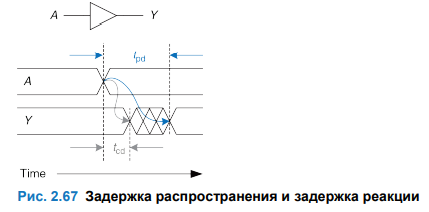
**49. Временные характеристики цифровых микросхем. Задержка распространения и задержка реакции. Импульсные помехи.**

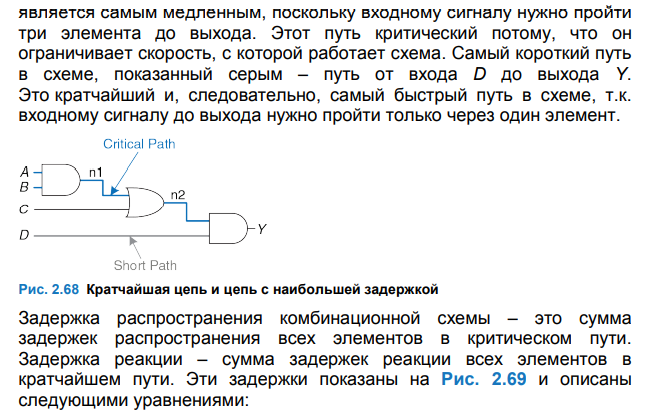
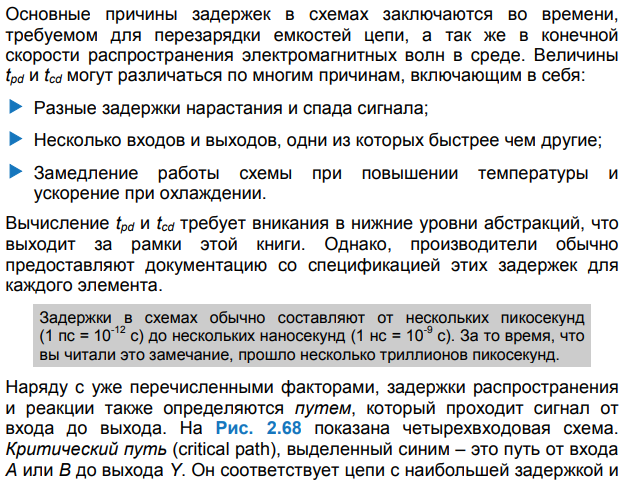
Как подтвердит любой опытный разработчик, одна из самых сложных задач в разработке схем – это учет всех ограничений, накладываемых на временные характеристики работы схемы, ведь хорошая схема должна работать предельно быстро и при этом без сбоев. Изменение выходного значения в ответ на изменение входа занимает время. На Рис. 2.66 показана задержка между изменением входа буфера и последующим изменением его выхода. Этот рисунок называется временной диаграммой; он изображает переходную характеристику схемы буфера при изменении входа. Переход от НИЗКОГО уровня к ВЫСОКОМУ называется положительным перепадом или фронтом. Аналогично, переход от ВЫСОКОГО уровня к НИЗКОМУ (на рисунке не показан) называется соответственно отрицательным перепадом или срезом. Синяя стрелка показывает, что положительный фронт сигнала Y вызывается положительным фронтом сигнала A. Величина задержки измеряется от момента времени, когда входной сигнал А достигает уровня 50%, до момента достижения уровня 50% выходным сигналом Y. Уровень 50% – это точка, в которой сигнал находится ровно посередине между НИЗКИМ и ВЫСОКИМ логическими уровнями.

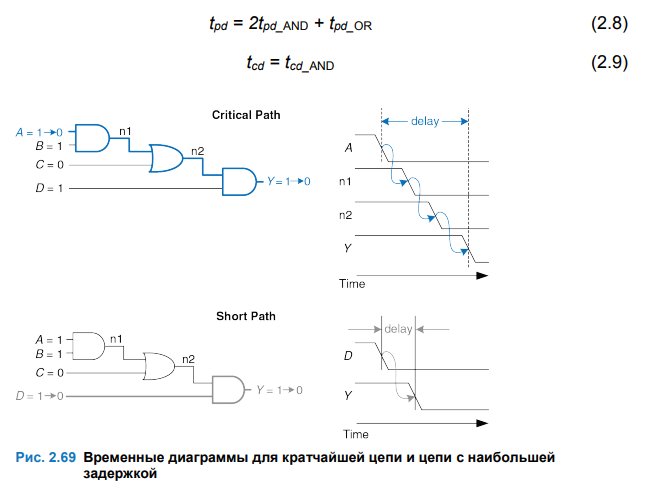
**Задержка распространения и задержка реакции.**

Когда разработчики говорят о задержке схемы, они в большинстве случаев имеют в виду наибольшее возможное значение задержки (задержку распространения), если только из контекста не следует другое.

Комбинационная логика характеризуется задержкой распространения (propagation delay) и задержкой реакции, или отклика (contamination delay). Задержка распространения tpd – это максимальное время от начала изменения входа до момента, когда все выходы достигнут установившихся значений. Задержка реакции tcd – это минимальное время от момента, когда вход изменился, до момента, когда любой из выходов начнет изменять свое значение. На Рис. 2.67 синим и серым цветом показаны соответственно задержки распространения и задержка реакции буфера. На рисунке показано, что вход A изначально имел или ВЫСОКОЕ, или НИЗКОЕ значение, и оно изменяется на противоположное в определенный момент времени; нас интересует только факт, что оно (значение A) изменилось, но не его конкретное значение. В ответ, спустя некоторое время, меняется Y. Стрелки показывают, что Y может начать меняться через временной интервал tcd после изменения A, и что Y точно установится в новое значение не позднее, чем через интервал tpd.

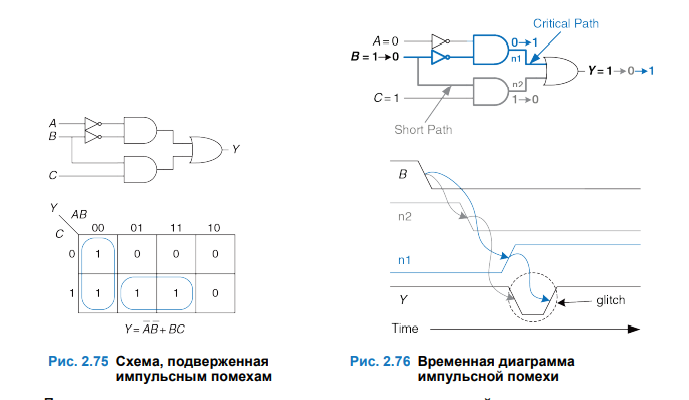
****

** **

****

**Импульсные помехи.**

До сих по мы обсуждали случай, когда одиночное изменение входного сигнала вызывает одиночное изменение выхода. Однако может оказаться, что одиночное изменение на входе вызывает несколько выходных изменений. Это называется импульсной помехой или паразитным импульсом. Хотя паразитный импульс обычно не вызывает проблем, важно понимать, что он есть, и уметь распознавать его на временных диаграммах. На Рис. 2.75 показана схема, подверженная паразитным импульсам, и карта Карно для нее. Логическое уравнение минимизировано корректно, однако посмотрите, что происходит, когда A = 0, C = 1 и B меняется из 1 в 0. Рис. 2.76 иллюстрирует этот сценарий. Короткий путь (показан серым) проходит через два элемента: И и ИЛИ. Критический путь (показан синим) проходит через инвертор и два элемента: И и ИЛИ. Как только B переключится из 1 в 0, n2 (в коротком пути) опустится в 0 до того, как n1 (в критическом пути) сможет установиться в 1. До подъема n1 оба входа элемента ИЛИ будут принимать значение 0, и его выход сбросится в 0. Когда n1 в конце концов поднимется, Y вернется в 1. Как показано на временных диаграммах на Рис. 2.76, Y начинается с 1 и заканчивается 1, но на короткое время переключается в 0.

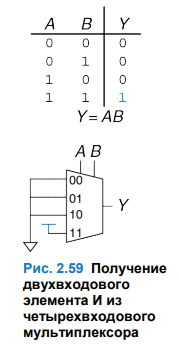
****

**50. Базовые комбинационные блоки. Мультиплексоры. Логика на мультиплексорах. Дешифраторы.**

Комбинационные логические элементы часто группируются в «строительные блоки», используемые для создания сложных систем. Это позволяет абстрагироваться от излишней детализации уровня логических элементов и подчеркнуть функцию «строительного блока».

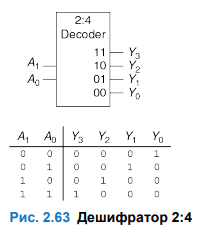
**Мультиплексоры** являются одними из наиболее часто используемых комбинационных схем. Они позволяют выбрать одно выходное значение из нескольких входных в зависимости от значения сигнала выбора.

Логика на мультиплексорах Мультиплексоры могут использоваться как таблицы преобразования (lookup tables) для выполнения логических функций. На Рис. 2.59 показан четырехвходовой мультиплексор, используемый для реализации двухвходового элемента И. Входы A и B служат управляющими линиями. Входы данных мультиплексора подключены к 0 и 1 согласно соответствующей строке таблицы истинности. Вообще, 2N -входовой мультиплексор можно запрограммировать для выполнения любой N-входовой логической функции, используя 0 и 1 для соответствующих входов данных. Действительно, изменением входных данных мультиплексор может быть перепрограммирован для выполнения различных функций. Немного смекалки, и мы сможем уменьшить размер мультиплексора наполовину, используя только 2N-1 -входовой мультиплексор для выполнения любой N-входовой логической функции. Способ заключается в том, чтобы подать один из литералов, так же как 0 и 1, на вход данных мультиплексора.

****Для иллюстрации этого принципа на Рис. 2.60 показаны функции двухвходовых элементов И и ИСКЛЮЧАЮЩЕЕ ИЛИ, реализованных на двухвходовых мультиплексорах. Мы начали с обычной таблицы истинности и затем скомбинировали пары строк, чтобы исключить самую правую входную переменную (B), и выразить выход в термах этой переменной. Например, в случае элемента И, когда A = 0, то Y = 0 вне зависимости от B. Когда A = 1, то Y = 0, если B = 0, и Y = 1, если B = 1, так что Y = B. Затем мы используем мультиплексор как таблицу подстановки согласно этой новой уменьшенной таблице истинности**.**

****

**Дешифраторы.**

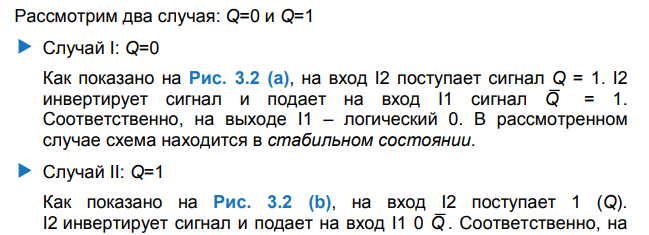
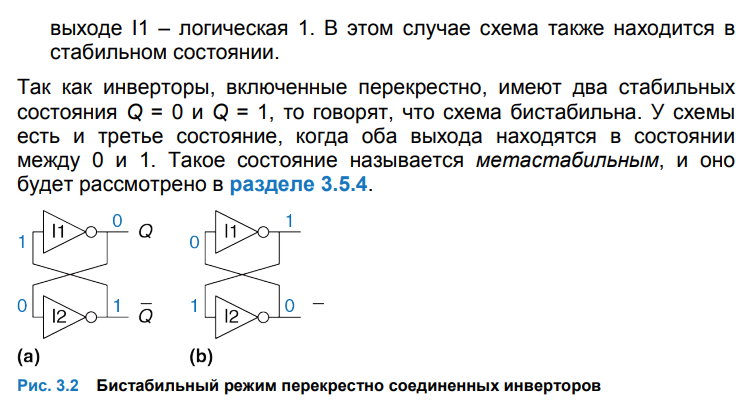
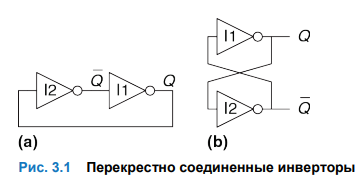
****В общем случае у дешифратора имеется N входов и 2N выходов. Он выдает единицу строго на один из выходов в зависимости от набора входных значений. На Рис. 2.63 показан дешифратор 2:4. Когда A[1:0] = 00, Y0 = 1. Когда A[1:0] = 01, Y1 = 1 и так далее. Выходы образуют прямой унитарный код (one-hot code), называемый так потому, что в любое время только один из выходов может принимать высокий уровень.

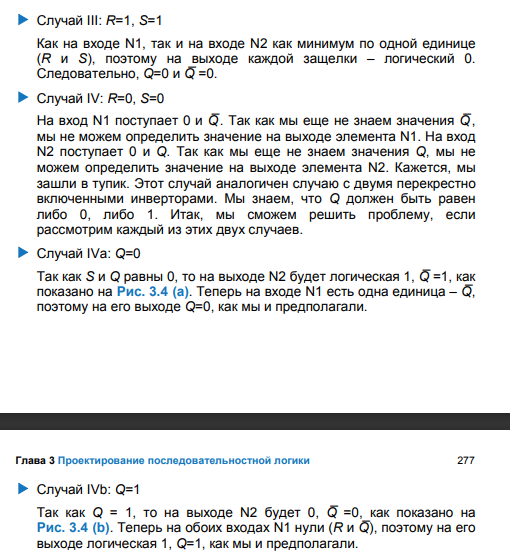
**51. Проектирование последовательностной логики. Защёлки и триггеры. RS-триггер. D-защёлка. D-Триггер. Регистр.**

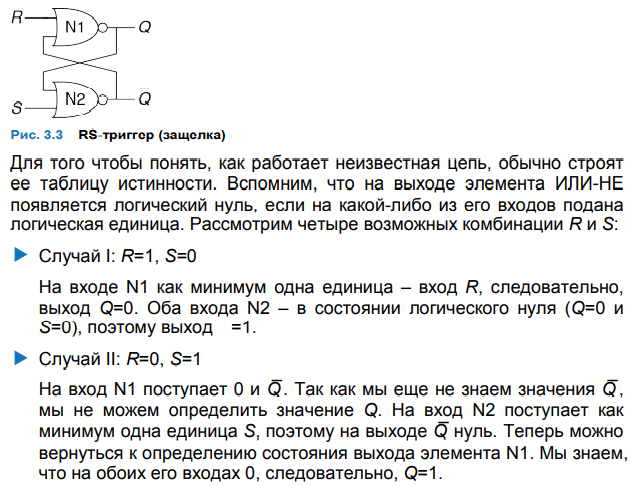
Значение на выходе последовательностной логической схемы зависит как от текущих, так и от предыдущих входных значений, следовательно, последовательнстные логические схемы обладают памятью. Последовательностные логические схемы могут явно запоминать предыдущие значения определенных входов, а могут «сжимать» предыдущие значения определенных входов в меньшее количество информации, называемое состоянием системы. Состояние цифровой последовательностной схемы – набор бит, называемый переменными состояния. Эти биты содержат всю информацию о прошлом, необходимую для определения будущего поведения схемы. Защелки и триггеры - являются простыми последовательностными схемами, запоминающими один бит информации. Вообще говоря, последовательностные схемы достаточно сложно анализировать. С целью упрощения проектирования мы ограничимся только синхронными схемами, состоящими из комбинационной логики и набора триггеров, хранящих информацию о состоянии системы. В главе описываются конечные автоматы, с помощью которых можно легко и просто проектировать последовательностные схемы. Наконец, мы проанализируем быстродействие последовательностных схем и обсудим параллельные вычисления как способ повышения быстродействия.

**ЗАЩЕЛКИ И ТРИГГЕРЫ**

Основным блоком для построения памяти является бистабильная ячейка – элемент с двумя устойчивыми состояниями. На Рис. 3.1 (а) показана простая бистабильная ячейка, состоящая из пары инверторов, замкнутых в кольцо. Эту схему можно перерисовать так, чтобы рисунок выглядел симметрично (Рис. 3.1 (b)). Теперь видно, что инверторы соединены перекрестно, то есть вход I1 соединен с выходом I2 и наоборот. У схемы нет ни одного входа, зато есть два выхода Q и Q¯. Анализ этой схемы отличается от анализа комбинационной схемы, так как схема является циклической: Q зависит от Q¯ , а Q¯ зависит от Q.

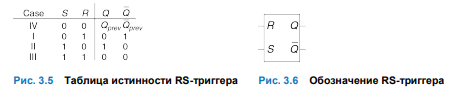
****

**RS-триггер.** Одной из простейших последовательностных схем является RS-триггер, (от англ. Reset и Set), состоящий, как показано на Рис. 3.3, из двух перекрестно включенных элементов ИЛИ-НЕ. У защелки есть два входа – R и S и два выхода Q и Q¯ . Принципы работы RS-триггера и схемы с перекрестно включенными инверторами аналогичны, но состояние защелки контролируются R и S входами, которые сбрасывают и устанавливают выход Q.

****

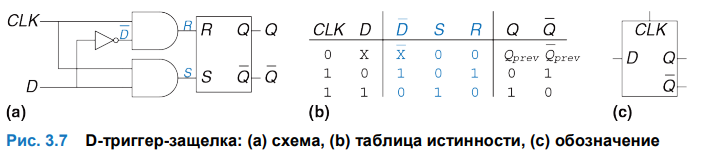
****

Установить бит означает перевести его в логическую единицу, а сбросить – в логический нуль. Обычно Q¯ является булевым дополнением Q. Когда поступает команда сброса R=1, выход Q принимает значение 0, а выход Q¯ – противоположное (лог. 1). Когда поступает команда установки бита S=1, выход Q становится единицей, а Q¯ – нулем. Если ни на один из входов не поступает логическая единица, на обоих выходах сохраняется предыдущее значение Qпред. Подача на входы одновременно R=1 и S=1 не имеет особого смысла, так как это означает, что выход должен быть одновременно и установлен и сброшен, что невозможно. Защелка, не зная, что ей делать, выставляет как на прямом, так и на инверсном выходе логический 0. Условное обозначение RS-триггера представлено на Рис. 3.6. Условные обозначения используются при модульном проектировании схемы с целью абстрагирования от внутренней структуры элемента.

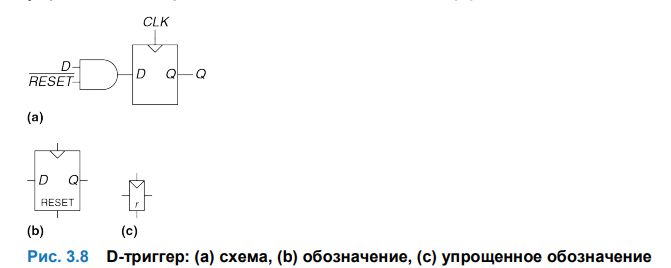
****

**D-защелка**

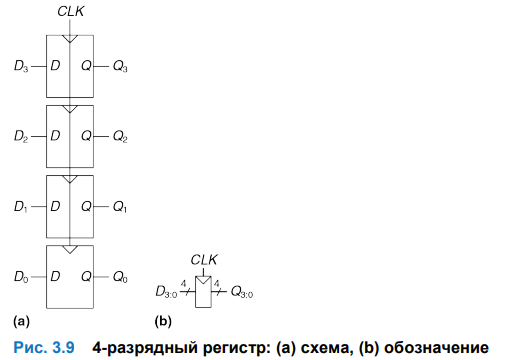
RS-триггер неудобен из-за необычного поведения, если на оба входа триггера одновременно поступает высокий уровень сигнала. Более серьезная проблема состоит в том, что вопросы ЧТО и КОГДА в контексте изменения состояния триггера объединены его R и S входами. Подача логической единицы на эти входы определяет не только, ЧТО произойдет, но и КОГДА это произойдет. Разработка схем упрощается, если эти вопросы ЧТО и КОГДА разделены. D-триггер-защелка (Рис. 3.7 (а)) решает эти проблемы. У триггера есть два входа: вход данных D, определяющий, каким будет следующее состояние, и вход тактового сигнала CLK, определяющий, когда оно изменится.

****

D-Триггер D-триггер, триггер синхронизируемый фронтом (далее – триггер), может быть построен из двух включенных последовательно D-защелок. Как показано на Рис. 3.8 (a), тактовые сигналы, которые подаются на них, являются булевыми дополнениями друг друга. Первую защелку называют ведущей (master), а вторую – ведомой (slave). Защелки соединены линией N1. Условное обозначение D-триггера приведено на Рис. 3.8 (b). Когда выход не используется, обозначение может быть упрощено до представленного на Рис. 3.8 (c).

****

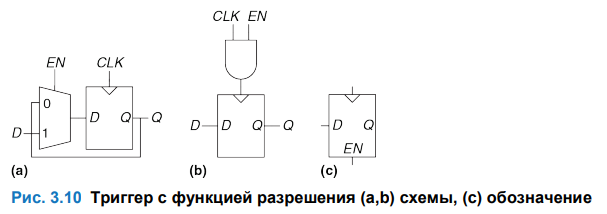
N-разрядный регистр – набор из N триггеров с общим тактовым сигналом. Таким образом, все биты регистра обновляются одновременно. Регистр является ключевым блоком при построении большинства последовательностных схем. На Рис. 3.9 показана схема и обозначение 4-разрядного регистра со входами D3:0 и выходами Q3:0. D3:0 и Q3:0 являются 4-разрядными шинами.

****

**52. Триггер с функцией разрешения. Триггер с функцией сброса. Проектирование синхронных логических схем. Синхронные последовательностные схемы. Синхронные и асинхронные схемы.**

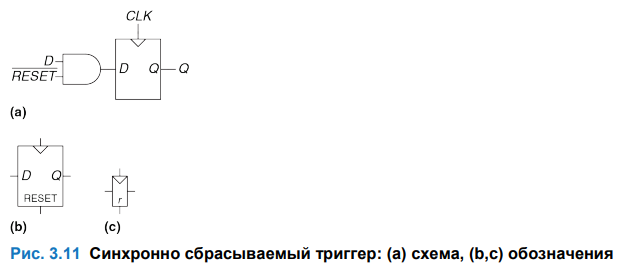
**Триггер с функцией разрешения**

У некоторых триггеров имеется еще один вход, называемый EN, или ENABLE (разрешить). Этот вход определяет, будут ли данные загружены по фронту или нет. Когда на EN подается логическая единица, то такой D-триггер ведет себя так же как и обычный D-триггер. Если же на EN поступает логический нуль, то триггер игнорирует тактовый сигнал и сохраняет свое состояние. Такие триггеры полезны, если мы хотим загружать значения в триггер только на протяжении какого-то времени, а не по каждому фронту тактовому импульсу.

****

**Триггер с функцией сброса**

В триггере с функцией сброса добавляется еще один вход, называемый RESET (сброс). Когда на RESET подан 0, сбрасываемый триггер ведет себя как обычный D-триггер. Когда на RESET подана 1, такой триггер игнорирует вход D и сбрасывает выход в 0. Триггеры с функцией сброса полезны, когда мы хотим ускорить установление определенного состояния (т.е. 0) во всех триггерах системы при первом включении. Такие триггеры могут сбрасываться как синхронно, так и асинхронно. Синхронно сбрасываемые триггеры сбрасываются только по фронту сигнала CLK. Асинхронно сбрасываемые триггеры сбрасываются сразу же при поступлении логической единицы на вход RESET, вне зависимости от тактового сигнала.

****

**ПРОЕКТИРОВАНИЕ СИНХРОННЫХ ЛОГИЧЕСКИХ СХЕМ**

Вообще говоря, последовательностные схемы включают в себя все схемы, которые не являются комбинационными, то есть последовательностные схемы – это те, значение выхода которых нельзя однозначно определить, зная лишь текущие значения входов. Поведение некоторых последовательностных схем может быть весьма сложным.

**Синхронные последовательностные схемы.**

Это скорее последовательностные, чем комбинационные схемы. В комбинационной логике нет циклических путей и нет зависимостей состояния выхода от времени прохождения сигнала. Если на входы комбинационной логической схемы поданы определенные сигналы, то ее выход спустя некоторое время всегда установится в определенное корректное состояние. Однако, в последовательностных схемах с циклическими путями может появиться нежелательная нестабильность или гонки. Проверка таких схем требует много времени, и многие выдающиеся проектировщики делали подобные ошибки. Во избежание таких проблем разработчики разрывают циклические пути и добавляют в разрыв регистры. Это превращает схему в набор комбинационной логики и регистров. В регистрах содержится состояние системы, изменяющееся только по фронту тактового импульса. В этом случае говорят, что состояние синхронизировано с тактовым сигналом. Если период тактового сигнала достаточно большой, чтобы все входы регистров успели установиться до фронта следующего тактового импульса, то эффекты, связанные с гонками, устраняются. Следование правилу «всегда использовать регистры в обратной связи» приводит нас к формальному определению синхронной последовательностной схемы.

**Синхронные и асинхронные схемы**

Теоретически, из-за отсутствия временных ограничений, накладываемых на систему тактирующимися регистрами, при проектировании асинхронных схем разработчик обладает большей свободой, чем при проектировании синхронных. Таким же образом, как аналоговые схемы менее формализованы по сравнению с цифровыми, из-за того, что в аналоговых схемах могут использоваться произвольные напряжения, асинхронные схемы менее формализованы, чем синхронные, так как обратная связь в них может быть любой. Однако, оказывается, что синхронные схемы проектировать и использовать проще, чем асинхронные, так же как цифровые схемы проще проектировать, чем аналоговые. Несмотря на многолетнее научные исследования асинхронных схем, почти все современные цифровые схемы являются синхронными. Асинхронные схемы иногда используются для связи между собой систем с разными тактовыми сигналами или для считывания значений со входов в произвольное время, так же как аналоговые схемы необходимы для взаимодействия с реальным миром аналоговых (непрерывных) напряжений. Более того, среди разработок в области асинхронных схем есть действительно выдающиеся, некоторые из них могут также улучшить характеристики синхронных схем.

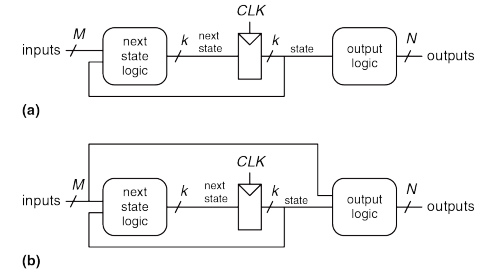
**53. Конечные автоматы. Пример проектирования конечного автомата.**

**КОНЕЧНЫЕ АВТОМАТЫ**

Последовательностные логические схемы могут быть изображены в форме, представленной на Рис. 3.22. Такие представления называются конечными автоматами (КА). Они получили свое название из-за того, что схема с k-регистрами может находиться в одном из 2k , то есть в конечном числе, состояний. У КА М входов, N выходов и k бит состояний. На вход КА так же подается тактовый сигнал и, возможно, сигнал сброса. КА состоит из двух блоков комбинационной логики: логики перехода в следующее состояние и выходной логики, – и из регистра, в котором хранится текущее состояние. По фронту каждого тактового импульса автомат переходит в следующее состояние, которое определяется текущим состоянием и значениями на входах. Существует два основных класса конечных автоматов, которые отличаются своими функциональными описаниями. В автомате Мура выходные значения зависят лишь от текущего состояния, в то время как в автомате Мили выход зависит как от текущего состояния, так и от входных данных. Конечные автоматы предоставляют систематический способ проектирования синхронных последовательностных схем по заданному функциональному описанию. Этот метод будет описан ниже, а сейчас мы рассмотрим простой пример.

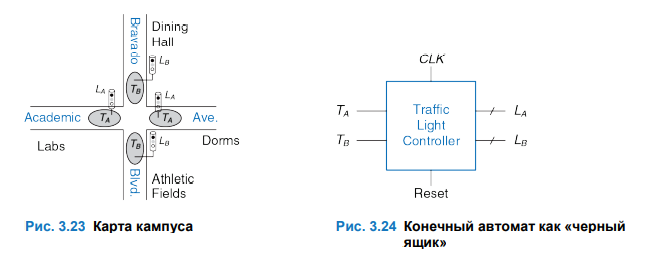
**Пример проектирования конечного автомата**

Для того чтобы проиллюстрировать процесс проектирования конечного автомата, рассмотрим проблему создания контроллера светофора для загруженного перекрестка в студенческом городке. Студенты-инженеры гуляют по Академической улице, на которой расположены учебные корпуса и общежитие. У них нет времени читать про конечные автоматы, и они не смотрят под ноги во время передвижения. Футболисты носятся между спортзалом и столовой по Беговой улице. Они гоняют мяч туда-сюда и тоже не смотрят под ноги. Несколько студентов уже получили серьезные травмы на перекрестке, и декан попросил Бена Битдидла установить светофор, пока не произошли инциденты с летальным исходом.

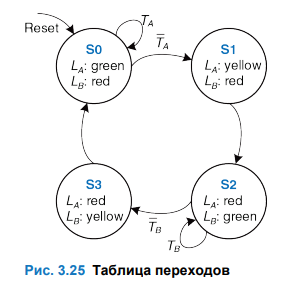
****

**Рис 3.22**

Бен решил справиться с проблемой с помощью конечного автомата. Он установил два датчика движения, TА и TБ, на Академической и Беговой улицах соответственно. Каждый датчик выдает единицу, если студенты присутствуют на улице и нуль, если никого нет. Он также установил два светофора для управления движением, LА и LБ. Каждый светофор получает входной цифровой сигнал, определяющий, каким светом он должен светить: красным, желтым или зеленым. Следовательно, у КА есть два входа, TА и TБ, и два выхода, LА и LБ. Перекресток с двумя светофорами и датчиками показан на Рис. 3.23. Бен подает тактовые импульсы раз в 5 секунд. По переднему фронту каждого импульса цвет светофора может измениться в зависимости от показаний датчиков движения. Также присутствует кнопка сброса, чтобы техники могли сбрасывать контроллер после подачи питания в известное исходное состояние. На Рис. 3.24 автомат изображен в виде «черного ящика».

****

Следующий шаг первокурсника – сделать набросок диаграммы переходов (или графа), показанный на Рис. 3.25, на котором приведены все возможные состояния системы и переходы между ними.

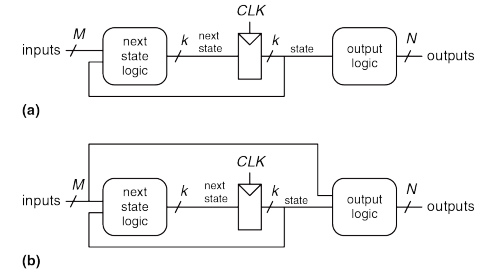
****

После сброса светофор горит зеленым на Академической улице и красным – на Беговой. Каждые 5 секунд контроллер анализирует движение и решает, что же делать дальше. Если движение присутствует на Академической улице, то цвет не меняется. Как только Академическая улица освобождается, на всех светофорах 5 секунд горит желтый, затем на Академической улице загорается красный, а на Беговой – зеленый. Аналогично, зеленый свет на Беговой улице сохраняется до тех пор, пока улица не станет свободной, затем светофор переключается на желтый, а затем – на красный. Кружки на диаграмме переходов обозначают состояния, а дуги со стрелками между ними – переходы между этими состояниями. Переходы осуществляются по переднему фронту тактового импульса. Мы не будем изображать тактовый сигнал на диаграмме, так как он всегда присутствует в синхронных логических схемах. Более того, тактовый сигнал лишь определяет, когда случится переход, тогда как диаграмма определяет, какой именно переход произойдет. Стрелка, обозначенная как Сброс, указывает на переход извне в состояние S0, отражая то, что система перейдет в это состояние сразу после сброса, независимо от того, в каком она была состоянии до этого. Если присутствует несколько стрелок, выходящих из некоторого состояния, то эти стрелки подписывают, чтобы показать, какой входной сигнал вызвал этот переход. Например, система находится в состоянии S0. Система останется в состоянии S0, если TA=1, и перейдет в состояние S1, если TA=0. Если из этого состояния выходит только одна стрелка, это означает, что такой переход произойдет вне зависимости от состояния входов. Например, из состояния S1 система всегда будет переходить в состояние S2, когда LA – красный, а LB – зеленый.

**54. Конечные автоматы. Кодирование состояний. Автоматы Мура и Мили.**

**КОНЕЧНЫЕ АВТОМАТЫ**

Последовательностные логические схемы могут быть изображены в форме, представленной на Рис. 3.22. Такие представления называются конечными автоматами (КА). Они получили свое название из-за того, что схема с k-регистрами может находиться в одном из 2k , то есть в конечном числе, состояний. У КА М входов, N выходов и k бит состояний. На вход КА так же подается тактовый сигнал и, возможно, сигнал сброса. КА состоит из двух блоков комбинационной логики: логики перехода в следующее состояние и выходной логики, – и из регистра, в котором хранится текущее состояние. По фронту каждого тактового импульса автомат переходит в следующее состояние, которое определяется текущим состоянием и значениями на входах. Существует два основных класса конечных автоматов, которые отличаются своими функциональными описаниями. В автомате Мура выходные значения зависят лишь от текущего состояния, в то время как в автомате Мили выход зависит как от текущего состояния, так и от входных данных. Конечные автоматы предоставляют систематический способ проектирования синхронных последовательностных схем по заданному функциональному описанию.

****

**Рис 3.22**

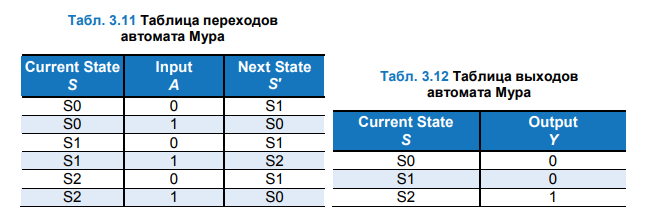
**Кодирование состояний**

В предыдущем примере кодирование состояний и выходов было выбрано произвольно. Выбор другой кодировки привел бы к другой схеме. Основная проблема заключается в том, как определить кодировку, которая потребует наименьшее количество элементов и приведет к наименьшим задержкам в схеме. К сожалению, простого способа найти самую лучшую кодировку не существует, кроме как перепробовать все возможные, что нерационально в случае, если число состояний велико. Однако зачастую возможно найти хорошую кодировку так, чтобы связанные состояния или выходы имели общие биты. При поиске набора возможных кодировок и выбора наиболее рациональной из них часто используются системы автоматизированного проектирования (САПР). Одно из важных решений в кодировании состояний – выбор между двоичным кодированием (00, 01, 10) и прямым кодированием (001, 010, 100), которое также называется кодированием «1 из N». При двоичном кодировании, как в примере с контроллером светофора, каждому состоянию ставится в соответствие двоичное число (номер этого состояния). Так как K двоичных чисел можно записать в log2K разрядах, системе с K состояниями нужно всего log2K битов состояния. В прямом кодировании для каждого состояния используется один бит состояния. По-английски оно называется one-hot, потому что только один разряд будет «горячим», то есть только в одном из разрядов содержится логическая единица в любой момент времени. Например, у КА с прямым кодированием и тремя состояниями коды состояний будут 001, 010 и 100. Каждый бит состояния хранится в триггере; таким образом, прямое кодирование требует большего количества триггеров, чем двоичное. Однако при использовании прямого кодирования схема определения следующего состояния и схема формирования выходных сигналов часто упрощается; таким образом, требуется меньше элементов. Наилучший выбор кодирования зависит от особенностей конкретного автомата.

**Автоматы Мура и Мили**

До сих пор мы рассматривали примеры автоматов Мура, выход в которых зависит лишь от состояния системы. Поэтому на диаграммах переходов для автоматов Мура значения выходов пишутся внутри кружков. Вспомним, что автоматы Мили очень похожи на автоматы Мура, но значения на их выходах могут зависеть он значений на входах таким же образом, как они зависят от текущего состояния системы. Поэтому на диаграммах переходов для автоматов Миля значения выходов пишутся над стрелками. В блоке комбинационной логики, который вычисляет выходные значения, используются значения текущего состояния и входов, как показано на Рис. 3.32 (b).

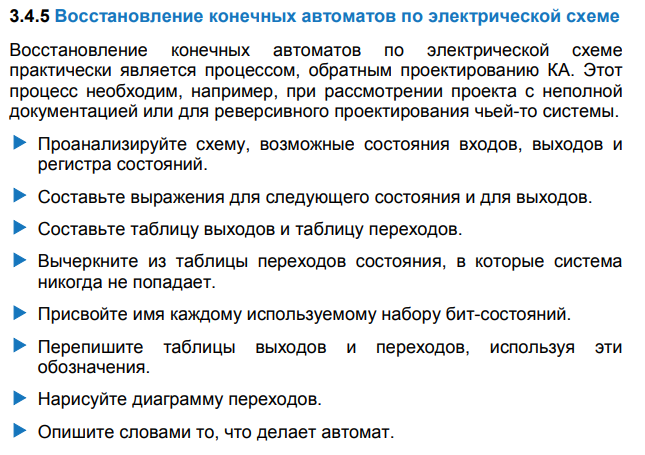
Простым способом запомнить разницу между двумя типами конечных автоматов состояний является тот факт, что у автомата Мура обычно больше (Moore – more) состояний, чем у автомата Мили, решающего ту же задачу.

****

**55. Декомпозиция конечных автоматов. Восстановление конечных автоматов по электрической схеме.**

**Декомпозиция конечных автоматов**

Проектирование сложных конечных автоматов часто упрощается, если их можно разбить на несколько более простых автоматов, взаимодействующих друг с другом таким образом, что выход одних автоматов является входом других. Такое применение принципов иерархической организации и модульного проектирования называется декомпозицией конечных автоматов.

****

На последнем шаге не бойтесь развернуто описывать цели и функции автомата, чтобы избежать простого переформулирования каждого перехода из диаграммы переходов.

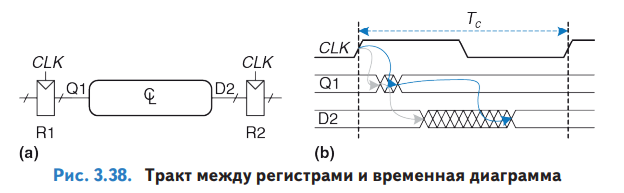
**56. Синхронизация последовательностных схем. Временные характеристики системы. Расфазировка тактовых сигналов. Метастабильность. Синхронизаторы.**

**Синхронизация последовательных схем**

Вспомните, что триггер копирует сигнал с D-входа на Q-выход по переднему фронту тактового сигнала. Этот процесс называется фиксацией (sampling) D-сигнала по фронту тактового импульса. Поведение триггера корректно, если сигнал на D-входе стабилен (равен 0 или 1 и не изменяется) в течение переднего фронта тактового сигнала. Но что произойдет, если сигнал D не будет стабилен во время изменения тактового сигнала? Эта ситуация аналогична той, которая возникает при спуске затвора фотокамеры. Представьте, что вы пытаетесь снять прыжок лягушки с плавающего листа кувшинки в озеро. Если вы нажмете на спуск перед прыжком, то на фотографии вы увидите лягушку на листе кувшинки. Если вы нажмете на спуск после прыжка, то на фотографии будет рябь на воде. Но если вы нажмете на спуск во время прыжка, то на фотографии вы увидите смазанное изображение вытянутой вдоль направления прыжка лягушки. Одной из характеристик фотокамеры является апертурное время, в течение которого фотографируемый объект должен быть неподвижен, чтобы на фотографии сформировалось его резкое изображение. Аналогично, последовательный элемент имеет апертурное время до и после фронта тактового сигнала, в течение которого его информационные входные сигналы должны быть стабильными, чтобы на выходе триггера сформировался корректный сигнал.

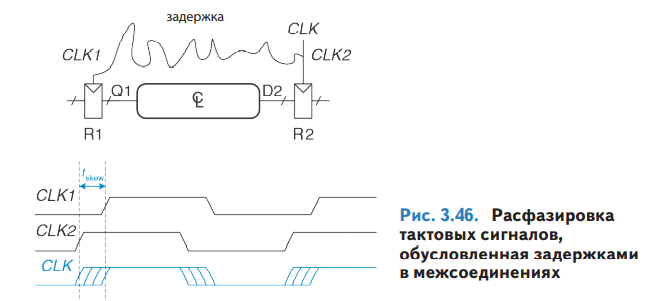
**Временные характеристики системы**

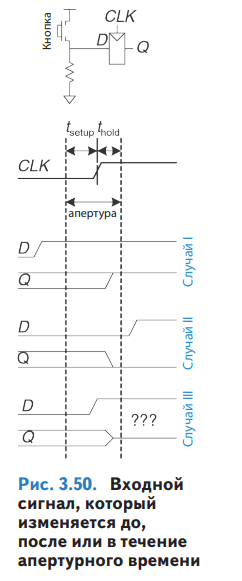
Периодом тактового сигнала или длительностью цикла синхронизации, Tc , называется промежуток времени между передними фронтами последовательных тактовых импульсов. Обратная величина, fc = 1/Tc , называется тактовой частотой. Увеличение тактовой частоты без изменения остальных параметров схемы приводит к увеличению ее производительности. Частота измеряется в герцах (Гц), или в циклах за одну секунду: 1 мегагерц (МГц) = 106 Гц и 1 гигагерц (ГГц) = 109 Гц. На Рис. 3.38 (a) показана характерная структура тракта обработки информации синхронной последовательной схемы, для которой мы рассчитаем период тактового сигнала. По переднему фронту тактового импульса на выходе регистра R1 формируется выходной сигнал (или сигналы) Q1. Эти сигналы поступают на вход блока комбинационной логики, выходные сигналы этого блока поступают на вход (или входы) D2 регистра R2. Как показано на Рис. 3.38 (b), выходной сигнал блока может начать изменяться не ранее окончания времени реакции после завершения изменения его входного сигнала и принимает окончательное значение спустя максимальное время задержки распространения от момента установления входного сигнала. Серые стрелки показывают минимальную задержку с учетом R1 и комбинационной логики, а синие – максимальную задержку распространения в тракте регистр R1 – комбинационная логика. Мы проанализируем временные ограничения с учетом времен предустановки и удержания второго регистра, R2.



**Расфазировка тактовых сигналов**

В предыдущих разделах предполагалось, что тактовые импульсы поступают на все регистры в одно и то же время. В действительности существует некоторый разброс этого времени. Эта неодновременность фронтов называется расфазировкой. Например, длина проводников, по которым тактовые сигналы поступают на разные регистры, может быть разной, это приводит к разным временам задержки, как показано на Рис. 3.46.



**Метастабильность**

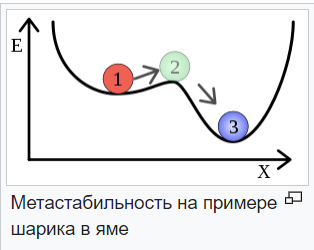
Как было указано ранее, не всегда можно гарантировать, что вход последовательной схемы будет стабилен в течение апертурного времени, особенно если входной сигнал поступает от внешнего асинхронного источника. Рассмотрим кнопку, подсоединенную к входу триггера, как показано на Рис. 3.50. Когда кнопка не нажата, D = 0. Когда кнопка нажата, D = 1. Обезьяна может нажимать кнопку в любой произвольный момент времени по отношению к фронту тактового сигнала. Мы хотим знать сигнал на выходе Q после переднего фронта сигнала CLK. В случае I, когда кнопка нажимается задолго до фронта CLK, Q = 1. В случае II, кнопка нажимается только намного позже фронта CLK, Q = 0. Но в случае III, когда кнопка нажимается в промежутке, который охватывает время предустановки перед фронтом тактовог импульса и время удержания после него, входной сигнал нарушает динамическую дисциплину и выход будет неопределенным.

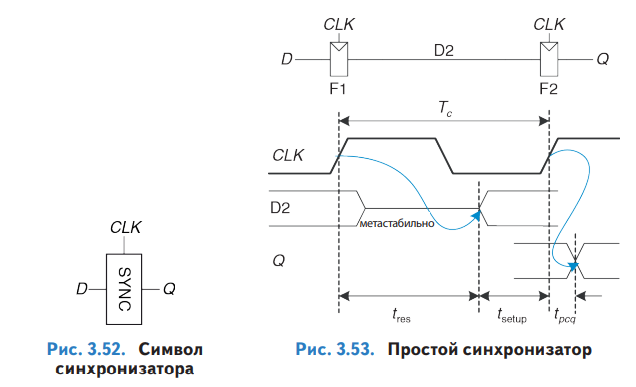
**Метастабильное состояние** (от [греч.](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B5%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) μετα «через» и [лат.](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D1%82%D0%B8%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *stabilis* «устойчивый») — состояние квазиустойчивого [равновесия](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B2%D0%BD%D0%BE%D0%B2%D0%B5%D1%81%D0%B8%D0%B5) [физической системы](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B7%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), в котором система может находиться длительное время.

Что такое метастабильное состояние, может быть понятно из приведенного справа рисунка:

* состояние 1: **метастабильное** — состояние, стабильность которого сохраняется при не очень больших возмущениях;
* состояние 2: **нестабильное** — состояние, стабильность которого нарушается при сколь угодно малых возмущениях;
* состояние 3: **стабильное** — состояние, стабильность которого сохраняется при больших возмущениях.

**Синхронизаторы**

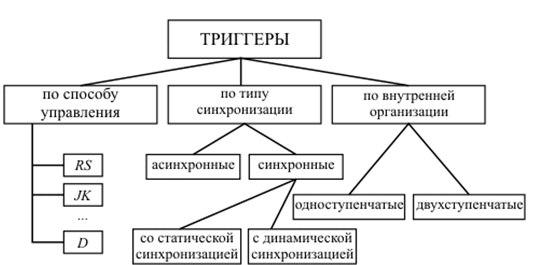
Наличие асинхронных входов цифровой системы, которые принимают информацию из внешнего мира, неизбежно. Например, сигналы, которые формирует человек, асинхронны. Такие асинхронные входы, если к ним относиться небрежно, могут привести к появлению метастабильных состояний в системе, что приведет к ее непредсказуемым отказам, которые крайне сложно отследить и исправить. При наличии асинхронных входов проектировщик системы должен обеспечить достаточно малую вероятность появления метастабильных напряжений. Смысл слова «достаточно» зависит от контекста. Для сотового телефона, вероятно, один отказ за 10 лет допустим, потому что пользователь может всегда выключить и включить телефон, если он «зависнет». Для медицинского прибора более предпочтительным является один отказ за предполагаемое время существования вселенной (1010 лет). Чтобы гарантировать корректность логических уровней, все асинхронные входы должны пройти через синхронизаторы. Синхронизатор, как показано на Рис. 3.52, является устройством, на вход которого поступает асинхронный сигнал D и тактовый сигнал CLK. За ограниченное время он формирует выходной сигнал Q, который с очень высокой вероятностью имеет корректный логический уровень. Если вход D стабилен в течение апертурного времени, то выход Q должен принять значение входа. Если D изменяется в течение апертурного времени, то Q может принять значение 0 или 1, но не должен быть метастабильным. На Рис. 3.53 показано, как из двух триггеров можно построить простой синхронизатор. Триггер F1 фиксирует значение входного сигнала D по переднему фронту тактового сигнала CLK. Если D изменяется в апертурное время, его выход D2 на некоторое время может стать метастабильным. Если период тактового сигнала достаточно велик, то с высокой вероятностью до конца периода D2 придет к корректному логическому**** уровню. Триггер F2 затем фиксирует D2, который теперь стабилен, и формирует корректный выходной сигнал.



**57. Типы триггеров. Классификация триггеров. RS-триггер на элементах И-НЕ и ИЛИ–НЕ. T-, JK-, D-триггеры.**

**Триггер**– электронная схема, обладающая двумя устойчивыми состояниями. Переход из одного устойчивого состояния в другое происходит скачкообразно под воздействием управляющих сигналов. При этом также скачкообразно изменяется уровень напряжения на выходе триггера.

Триггеры служат основой для построения регистров, счетчиков и других элементов, обладающих функцией хранения.Главной частью любого триггера является запоминающая ячейка (ЗЯ).

Типы триггеров:

Триггер называется синхронным, если его таблица переходов хотя бы по одному управляющему входу реализуется под воздействием синхронизирующего сигнала.

Основу синхронного одноступенчатого триггера составляет рассмотренная выше запоминающая ячейка (элементы 1, 2). Комбинационная схема преобразует управляющие сигналы триггера, а также, для некоторых типов триггеров, сигналы Q и с выходов ЗЯ в сигналы S и R на входах запоминающей ячейки.

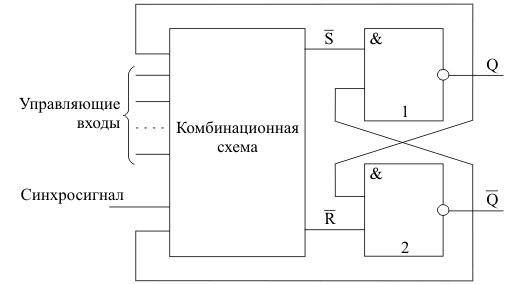
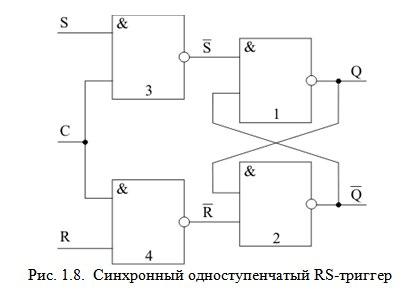
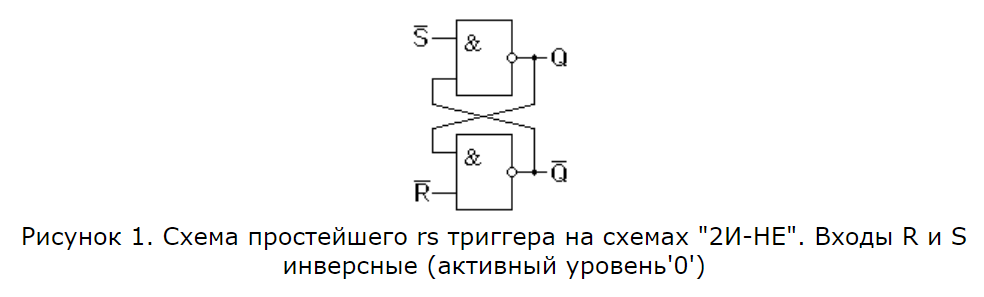
Обобщенная схема синхронного одноступенчатого триггера:

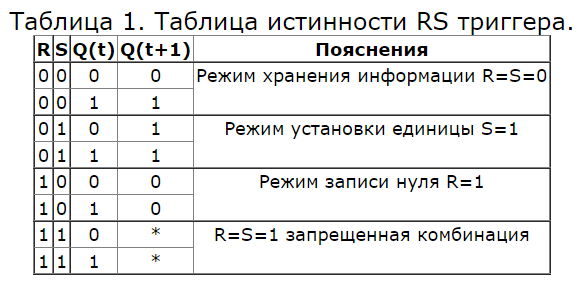
Схема синхронного одноступенчатого RS-триггера:



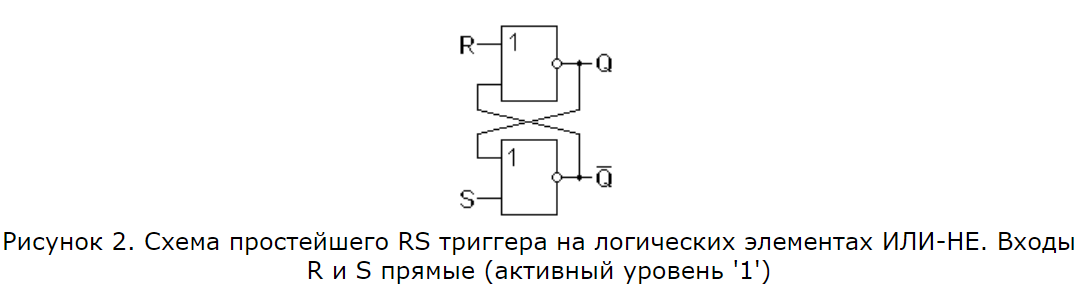
****

Рассмотрим принцип работы RS триггера, выполненный по изображенной на рисунке 1 схеме подробнее. Пусть на входы R и S подаются единичные потенциалы. Если на выходе верхнего логического элемента "2И-НЕ" Q присутствует логический ноль, то на выходе нижнего логического элемента "[2И-НЕ](https://digteh.ru/digital/TTL/)" появится логическая единица. Эта единица подтвердит логический ноль на выходе Q. Если на выходе верхнего логического элемента "2И-НЕ" Q первоначально присутствует логическая единица, то на выходе нижнего логического элемента "2И-НЕ" появится логический ноль. Этот ноль подтвердит логическую единицу на выходе Q. То есть при единичных входных уровнях схема RS триггера работает точно так же как и схема на инверторах.

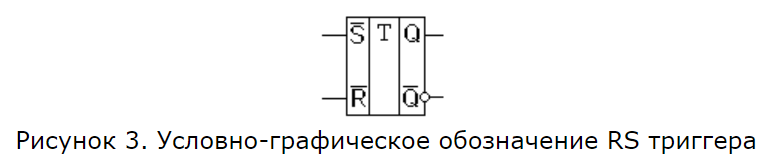
Подадим на вход S нулевой потенциал. Согласно таблице истинности логического элемента "И-НЕ" на выходе Q появится единичный потенциал. Это приведёт к появлению на инверсном выходе триггера нулевого потенциала. Теперь, даже если снять нулевой потенциал с входа S, на выходе триггера останется единичный потенциал. То есть мы записали в триггер логическую единицу.

****Точно так же можно записать в RS-триггер и логический ноль. Для этого следует воспользоваться входом R. Так как активный уровень на входах оказался нулевым, то эти входы — инверсные. Составим таблицу истинности RS триггера. Входы R и S в этой таблице будем использовать прямые, то есть и запись нуля, и запись единицы будут осуществляться единичными потенциалами (таблица 1).

RS триггер можно построить и на логических элементах "ИЛИ". Его схема приведена на рисунке 2. Принцип работы RS триггера, собранный на логических элементах "ИЛИ" будет точно таким же, как и рассмотренный ранее. Единственное отличие в работе этой схемы по сравнению с предыдущей схемой RS триггера будет заключаться в том, что сброс и установка триггера будет производиться единичными логическими уровнями. Эти особенности связаны с принципами работы инверсной логики, которые рассматривались ранее.

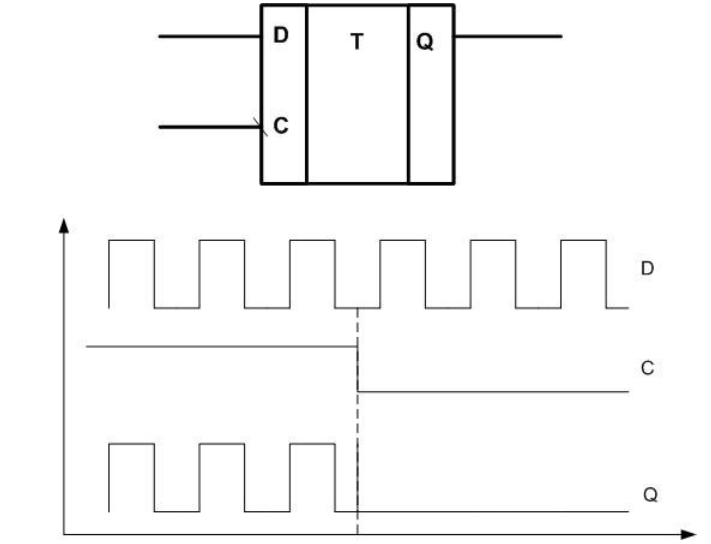
****

Так как RS триггер при реализации его на логических элементах "И" и "ИЛИ" работает одинаково (его принцип работы от схемы не зависит), то и условно-графическое изображение на принципиальных схемах тоже одинаково. [Условно-графическое изображение](https://digteh.ru/digital/DefDigCh.php) RS триггера приведено на рисунке 3.

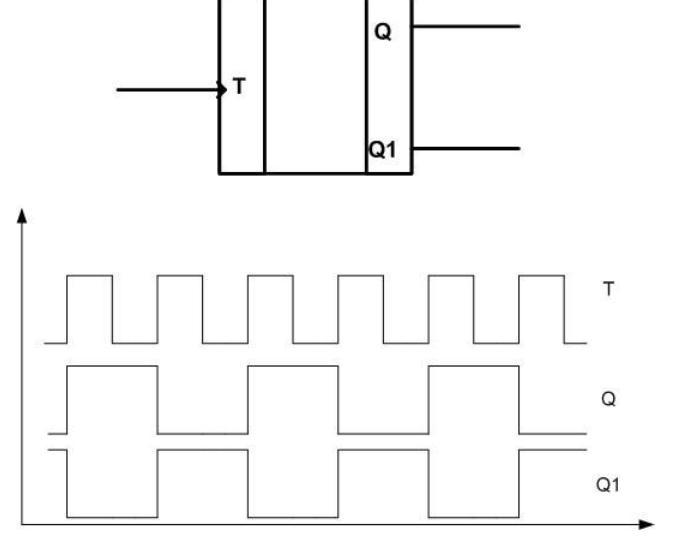
****

**D-триггер** («прозрачный триггер», «защелка», latch) относится к категории синхронных устройств, тактируемых по входу С. Также имеется вход для данных D (Data). По функциональным возможностям устройство относится к триггерам с приёмом информации по одному входу.

Пока на входе для синхронизации присутствует логическая единица, сигнал на выходе Q повторяет сигнал на входе данных (режим прозрачности). Как только уровень строба перейдет в состояние 0, на выходе Q уровень останется тем же, что был в момент перепада (защелкнется). Так можно зафиксировать входной уровень на входе в любой момент времени. Также существуют D-триггеры с тактированием по фронту. Они защёлкивают сигнал по положительному перепаду строба.

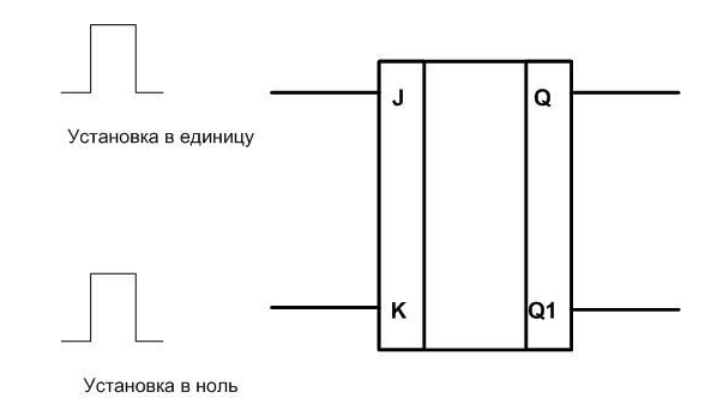


**T-триггер** относится к классу счётных бистабильных элементов. Логика его работы проста – он изменяет своё состояние каждый раз, когда на его вход приходит очередная логическая единица. Если на вход подать импульсный сигнал, выходная частота будет в два раза выше входной. На инверсном выходе сигнал будет противофазен прямому.



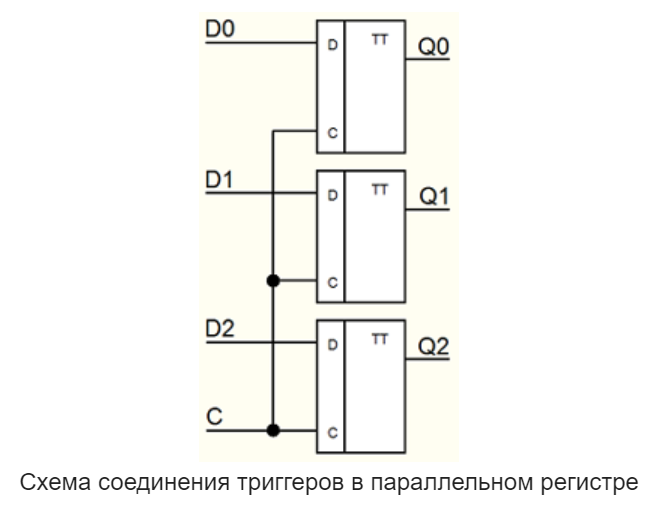
**JK-триггеры**

Этот бистабильный элемент относится к категории универсальных. Он может управляться раздельно по входам. Логика работы JK-триггера похожа на работу RS-элемента. Для установки выхода в единицу используется вход J (Job). Появление высокого уровня на выводе K (Keep) сбрасывает выход в ноль. Принципиальным отличием от RS-триггера является то, что одновременное появление единиц на двух управляющих входах не является запретным. В этом случае выход элемента меняет свое состояние на противоположное.



**58. Параллельные и последовательные регистры. Отличия в обозначения цифровых элементов в разных стандартах.**

## Параллельный регистр

****В данном типе регистров триггеры соединены параллельно, то есть каждый внутренний триггер имеет свой вход D и свой выход Q, которые не зависят от других триггеров, а также вход С, который называется **тактовым входом** и для всех входящих в регистр триггеров он является общим. Параллельные регистры бывают двух типов:

* **тактируемые регистры**, которые срабатывают по **фронту сигнала** управления (вход С);
* **стробируемые регистры**, которые срабатывают по **уровню входного сигнала** (вход С);

Наибольшее распространение получили тактируемые параллельные регистры.

### **Регистры, срабатывающие по фронту сигнала**

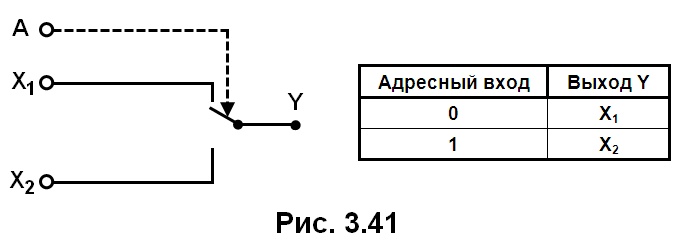
Данные типы регистров практически ничем не отличаются от D триггеров. Напомню, что на выходе Q устанавливается такой уровень напряжения, который был на входе D, в момент положительного фронта тактового сигнала на входе С. Так как регистр состоит из нескольких D триггеров, то и количество (4,6,8,16) запоминаемых сигналов (бит) у регистра больше.

### **Регистры, срабатывающие по уровню сигнала**

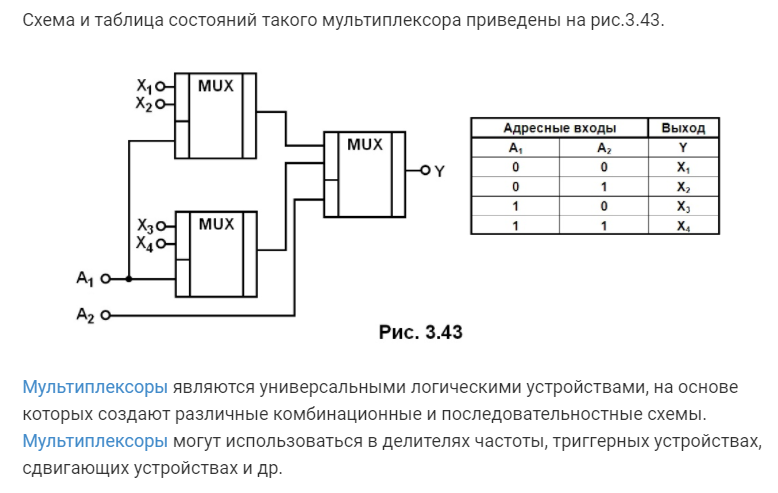
Данные типы регистров являются как бы гибридом между буфером и регистром (ещё такие регистры называют **регистры-защёлки**, англ. Latch). Принцип работы данных регистров состоит в следующем: когда на **стробирующем входе** высокий логический уровень, сигнал **поступает с входов** на выходы регистра, а если на стробирующем входе низкий уровень сигнала, регистр переходит в **режим хранения** последнего из пропущенных значений входного сигнала. Регистры-защелки не очень распространены из-за своего специфического характера работы, но в некоторых случаях удобнее, **регистров срабатывающих по фронту сигнала** на тактовом входе.

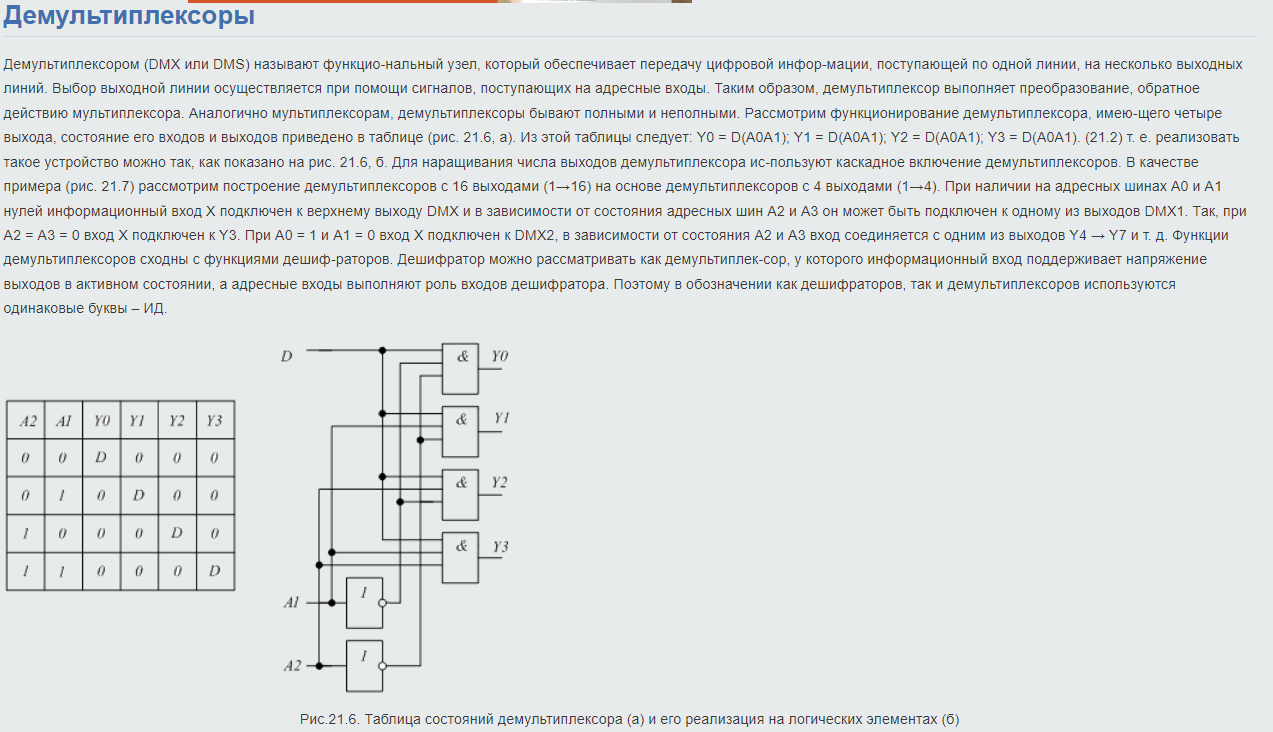
**59. Мультиплексоры и демультиплексоры. Отличия в обозначения цифровых элементов в разных стандартах.**

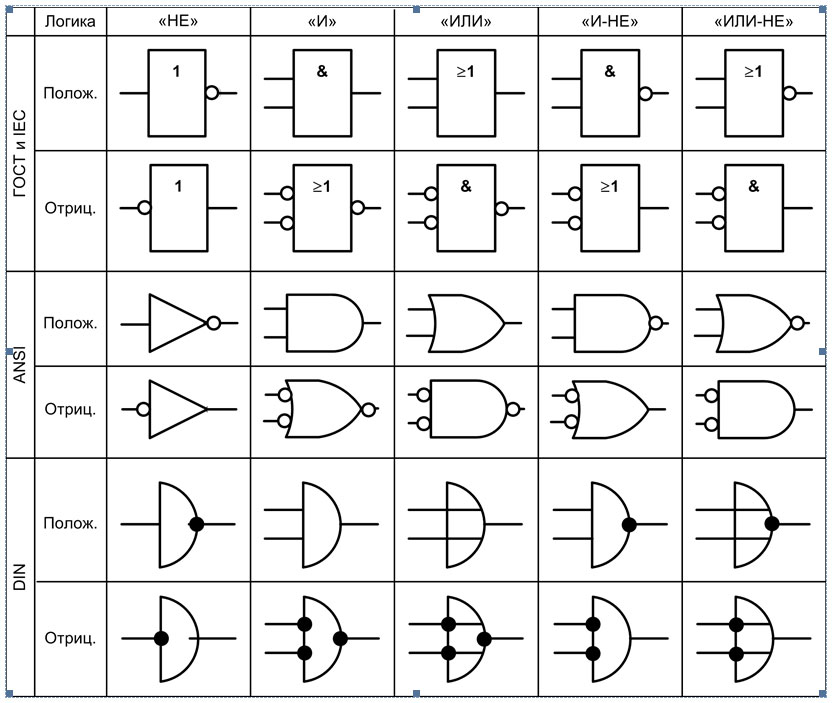
Мультиплексором — называют комбинационное устройство, обеспечивающее передачу в желаемом порядке цифровой информации, поступающей по нескольким входам на один выход. Мультиплексоры обозначают через MUX (от англ. multiplexor), а также через MS (от англ. multiplex or selector).



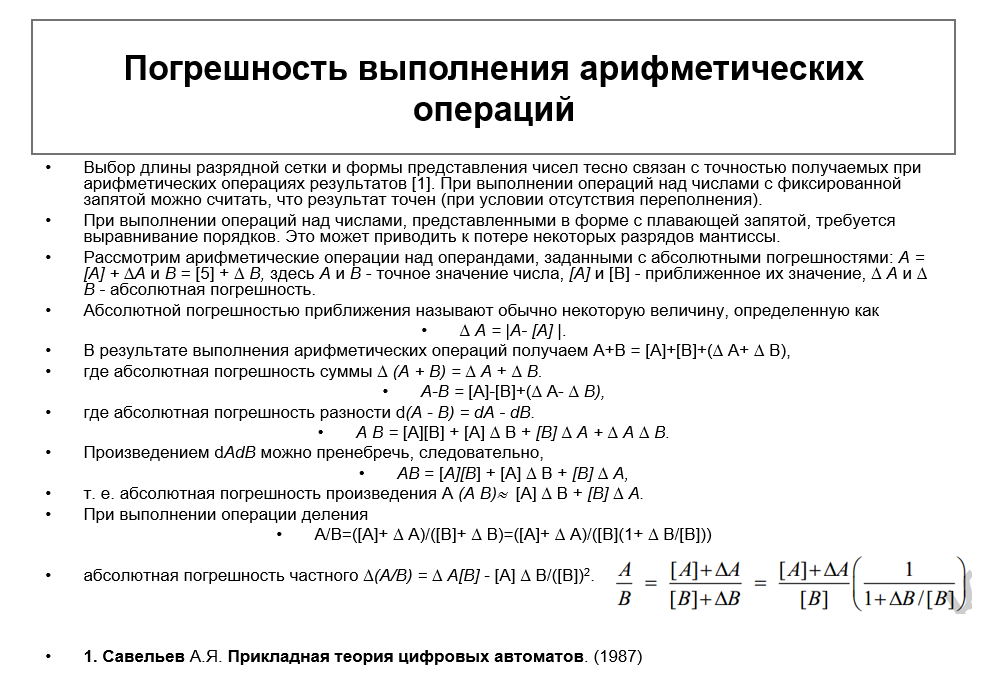
Сигналы на адресных входах определяют, какой конкретно информационный канал подключен к выходу. Если между числом информационных входов n и числом адресных входов m действует соотношение n = 2m, то такой мультиплексор называют полным. Если n< 2ь, то мультиплексор называют неполным.



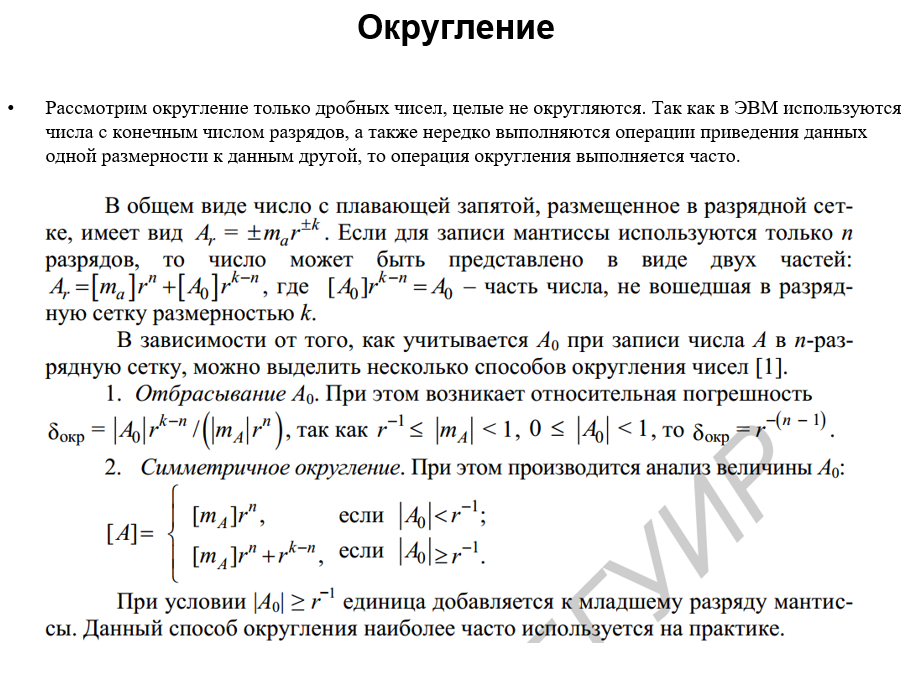
****

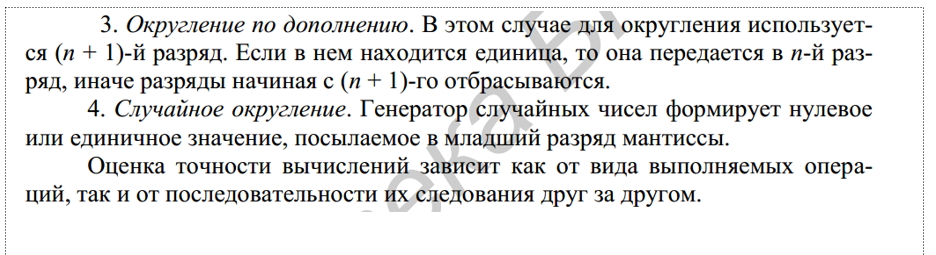


**60. Погрешность метаматематических операций в цифровых системах. Способы оценки. Округление.**

****

****

****

****