Оглавление

[**1. Системы счисления. *Перевод чисел из одной системы счисления в другую.Метод преобразования с использованием весов разрядов*** 2](#_Toc534575129)

[***2. Перевод чисел из одной системы счисления в другую. Метод деления (умножения) на новое основание*** 2](#_Toc534575130)

[***3.Перевод чисел из одной системы счисления в другую. Метод с использованием особого соотношения оснований исходной и искомой систем счисления.*** 3](#_Toc534575131)

[**4.Арифметические операции над двоичными числами. Операция сложения и вычитания в 2-й СС** 4](#_Toc534575132)

[**5.Операция умножения в двоичной системе исчисления** 4](#_Toc534575133)

[**6. IEEE754. Специальные числа. Зачем нулю знак.** 5](#_Toc534575134)

[**7.Деление двоичных чисел (общие правила).** 6](#_Toc534575135)

[**10.Двоично-десятичная арифметика. Сложение и вычитание двоично-десятичных чисел .** 6](#_Toc534575136)

[**11.Кодирование алгебраических чисел . Дополнительный и обратный коды двоичных чисел** 7](#_Toc534575137)

[**12. Операции с двоичными числами в дополнительном коде и в обратном коде** 8](#_Toc534575138)

[**13. Модифицированные коды** 9](#_Toc534575139)

[**15. Арифметические сдвиги положительных двоичных чисел, представленных в прямом коде. . Арифметические сдвиги двоичных чисел, представленных в обратном коде.** 10](#_Toc534575140)

[**16.Арифметические сдвиги двоичных чисел, представленных в дополнительном коде. Сдвиг отрицательных чисел с переполнением.** 11](#_Toc534575141)

[**17. Представление чисел с фиксированной точкой. Арифметические операции над числами, представленными с фиксированной точкой.** 12](#_Toc534575142)

[**18.Представление чисел с плавающей точкой. Сложение чисел, представленных в формате с плавающей точкой.** 13](#_Toc534575143)

[**19. Умножение чисел, представленных в формате с плавающей точкой. Деление чисел, представленных в формате с плавающей точкой.** 14](#_Toc534575144)

[**22. Денормализованные числа. Подводные камни в арифметике с плавающей запятой** 16](#_Toc534575145)

[**24.Основные понятия алгебры логики. Способы задания логической функции.** 17](#_Toc534575146)

[**25.Законы и правила алгебры Буля** 19](#_Toc534575147)

[**27. Полная система логических функций.** 20](#_Toc534575148)

[**29. Минимизация булевых функций. Основные понятия. Наиболее известные методы минимизации. Минимизация системы логических функций. Минимизация частично определенных функций** 22](#_Toc534575149)

[**30. Минимизация логических выражений методом Квайна** 26](#_Toc534575150)

[**Второй этап (табличный) (получение минимальной формы)** 28](#_Toc534575151)

[*Импликантная матрица* 29](#_Toc534575152)

[**Использование метода для получения минимальной** 30](#_Toc534575153)

[**31. Минимизация логических выражений с диаграммами Вейча.** 30](#_Toc534575154)

[**32. Логический базис И-НЕ. Синтез логических схем по логическому выражению в базисе И-НЕ.** 32](#_Toc534575155)

[**33. Код Грея. Обратная польская запись** 37](#_Toc534575156)

[Преобразование двоичного кода в код Грея 39](#_Toc534575157)

[Преобразование кода Грея в двоичный код 40](#_Toc534575158)

[**Необычные вариации кода Грея** 40](#_Toc534575159)

[*Сбалансированный код Грея* 40](#_Toc534575160)

[*Однодорожечный код Грея* 40](#_Toc534575161)

[*Двухмерный код Грея* 41](#_Toc534575162)

[Перестановки 41](#_Toc534575163)

[Генерирование перестановок 42](#_Toc534575164)

[34.АЦП и ЦАП 44](#_Toc534575165)

[**35. Искусство управления сложностью. цифровая абстракция** 45](#_Toc534575166)

[**36.Логические элементы. Таблицы истинности. Обозначения в разных представления.** 47](#_Toc534575167)

[**Таблицы истинности для основных двоичных логических функций** 47](#_Toc534575168)

[5. Логическое отрицание (инверсия) делает истинное высказывание ложным и, наоборот, ложное – истинным. **37.** **За пределами цифровой абстракции. Напряжение питания. Логические уровни. Допускаемые Уровни Шумов. Передаточная Характеристика. Статическая Дисциплина.** 48](#_Toc534575169)

[**38. Биполярные и КМОП транзисторы. Полупроводники. Конденсаторы. n-МОП и p-МОП-транзисторы** 50](#_Toc534575170)

[**39. Логический вентиль НЕ и другие на КМОП-транзисторах. Псевдо n-МОП-Логика Потребляемая мощность** 53](#_Toc534575171)

[**40. Проектирование комбинационной логики. От логики к логическим элементам, Что такое Х и Z: способы сопряжения микросхем в ЭВМ** 56](#_Toc534575172)

[**42. Базовые комбинационные блоки. Мультиплексоры. Логика на мультиплексорах. Дешифраторы** 59](#_Toc534575173)

[**41. Временные характеристики цифровых микросхем. Задержка распространения и задержка реакции. Импульсные помехи** 62](#_Toc534575174)

[**43. Проектирование последовательностной логики. ЗАЩЕЛКИ И ТРИГГЕРЫ. RS-триггер. D-защелка. D-Триггер. Регистр.** 66](#_Toc534575175)

[**44. Триггер с функцией разрешения. Триггер с функцией сброса. Проектирование синхронных логических схем. Синхронные последовательностные схемы. Синхронные и асинхронные схемы.** 70](#_Toc534575176)

[**46.Конечные автоматы. Кодирование состояний. Автоматы Мура и Мили. Декомпозиция конечных автоматов. Восстановление конечных автоматов по электрической схеме (320-339)** 75](#_Toc534575177)

[**47.Синхронизация последовательностных схем. Временные характеристики системы. Расфазировка тактовых сигналов. Метастабильность. Синхронизаторы (346-379)** 76](#_Toc534575178)

[**48.Параллелизм** 81](#_Toc534575179)

[**49.корректирующие коды. Код Хэ́мминга. Область применения. Презентация** 82](#_Toc534575180)

[**50.Языки описания аппаратуры. ПЛИС (FPGA) модули** 86](#_Toc534575181)

[**51. Типы триггеров. Классификация триггеров. RS-триггер на элементах И-НЕ и ИЛИ–НЕ. T-, JK-, D-триггеры** 87](#_Toc534575182)

[**52. Параллельные и последовательные регистры. Мультиплексоры и демультиплексоры. Отличия в обозначения цифровых элементов в разных стандартах** 92](#_Toc534575183)

[**53.Сумматор. Многоразрядный сумматор.** 93](#_Toc534575184)

[**54. Устройства ЭВМ. Состав АЛУ. Типы памяти**. 95](#_Toc534575185)

# **1. Системы счисления. *Перевод чисел из одной системы счисления в другую.Метод преобразования с использованием весов разрядов***

Все **системы счисления**(далее СС) можно подразделить на **позиционные** и **непозиционные**.

В **непозиционных** СС «доля» цифры или ее вес в количественном измерении записанного числа не зависит от местоположения данной цифры в записи этого числа. Примером такой СС является римская.I V X L C D M (1 5 10 50 100 500 1000 – их десятичные эквиваленты). При количественной оценке числа его значение определяется как сумма значений цифр, составляющих запись числа, кроме пар, состоящих из цифры меньшего веса, предшествующей цифре большего веса, значение которой определяется как разность веса большей и меньшей цифр.

**Количественная оценка числа**, записанного в позиционной СС, определяется как сумма произведений значения цифр, составляющих запись числа, умноженных на вес позиции, в которой располагается цифра. Примером такой СС является широко используемая десятичная система счисления.

***Десятичная система счисления***является также системой с равномерно распределенными весами, которые характеризуются тем, что соотношение весов двух любых соседних разрядов имеет для такой системы одинаковое значение. Это соотношение называется **основанием системы счисления**, которое далее будем обозначать как «*q*».

Общая запись числа в системе с равномерно распределенными весами имеет вид

*Nq = Аn Аn–*1 *... А*2 *А*1 *А*0(кодированная запись). Значение такого числа определяется как

*Nq =Аn* ⋅*qn +Аn–*1 ⋅*qn–*1 *+Аn*–2 ⋅*qn*–1 *+...А*2 ⋅*q*2 *+А*1 ⋅*q*1 *+А*0 ⋅*q*0(расширенная запись), где *Аi* – цифра записи числа, удовлетворяющая условию 0 ≤ *Аi* ≤ (*q*–1), где *q* – основание СС.

При *q* =10 *А* изменяется в диапазоне от 0 до 9.

Помимо *q* = 10 (десятичная СС), возможны другие значения для основания СС:− двоичная, восьмеричная, шестнадцатеричная и т. д. Человек в своей практической деятельности наиболее часто использует *десятичную* СС. *Двоичная СС* является удобной для обработки информации в ЭВМ. Промежуточное место между ними занимает *двоично*-*десятичная СС*, которая является десятичной, но отдельные десятичные цифры в ней записываются в виде набора двоичных разрядов.

Наличие различных систем счисления предполагает использование способов перевода записи числа из одной системы в другую. Для этой цели применяются следующие методы преобразований:

- преобразования с использованием весов разрядов в исходной и в искомой записи числа;

- деления (умножения) на новое основание;

- с использованием особого соотношения заданной и искомой систем счисления.

***Метод преобразования с использованием весов разрядов***

Метод преобразования с использованием весов разрядов записи числа в исходной и в искомой системах предполагает применение расширенной записи числа (1.2) в некоторой системе счисления.

Метод имеет две разновидности в зависимости от того, какая система счисления (исходная или искомая) является более привычной. Если более привычной является искомая система, то на основании расширенной записи исходного числа подсчитываются значения ее отдельных разрядов в новой системе счисления. Далее полученные значения суммируются. Например, при преобразовании целого двоичного числа *N2* = 110011010 в десятичную систему счисления исходное число представляется в расширенной записи *N* = 28 + 27 + 24 + 23 + 21 и рассчитывается вес отдельных (ненулевых) двоичных разрядов в десятичной системе счисления.

# ***2. Перевод чисел из одной системы счисления в другую. Метод деления (умножения) на новое основание***

Наличие различных систем счисления предполагает использование способов перевода записи числа из одной системы в другую. Для этой цели применяются следующие методы преобразований:

- преобразования с использованием весов разрядов в исходной и в искомой записи числа;

- деления (умножения) на новое основание;

- с использованием особого соотношения заданной и искомой систем счисления.

Метод деления (умножения) имеет две разновидности соответственно для преобразования целых и дробных чисел.

***1.Преобразование целых чисел***

Задачу представления числа *N*, заданного в системе *q*1, в системе счисления с основанием *q*2 можно рассматривать как задачу поиска коэффициентов полинома, представляющего собой расширенную запись числа *N* в системе счисления *q*2:

*Nq*1 *= а*0 *+ а*1 *× q* 21 *+ а*2 *× q*22 *+....+ аn–*2 *× q* 2*n–*2*+ аn–*1 *× q*2*n–*1*+*

*+ аn × q*2*n = Nq*2 *.* (1.3)

Введем скобочную форму для выражения (1.3):

*Nq*1 = *Nq*2 = *а*0 + *q*2 (*а*1 +q2 (*а*2 +*q*2 (*а*3 + . +*q*2(*an*-1 + *q*2(*аn*))…));

Обозначим выражение в первой скобке как *N*1, выражение во второй скобке как *N*2, в третьей – как *N3* и т. д., выражение в (*n–*1)*-*й скобке – как *N*(*n–*1), выражение в *n-*й скобке – как *Nn*. Теперь, основываясь на выражении (1.3), можно утверждать, что при делении *Nq*1*/q*2 будет получена целая часть частного int(*Nq*1/*q*2) и остаток rest(*Nq*1/*q*2). Это можно записать:

*Nq*1/*q*2 → int(*Nq*1/*q*2) — целая часть частного *N*1, и остаток rest(*Nq*1/*q*2), равный *а*0.

Аналогично для остальных скобок: *N1*/*q* 2 → int(*N*1/*q*2) равное *N*2 и остаток rest(*N*1/*q*2), равный а1; *N*2/*q*2 → int(*N*2/*q*2) равное *N*3, и остатк rest(*N*2/*q*2), равный а2; *N*(*n–*2)/*q* 2 → int(*N*(*n–*2)/*q*1) равное *N*(*n–1)*, остаток rest(*N*(*n* = 2)/*q*1), равный а(*n–*2); *N*(*n–1*)/*q*2 → int(*N*(*n–*1)/*q*2) = *Nn* = а*n* и остаток rest(*N*(*n–*1)/*q*2), равный

*а*(*n–1*), при этом *Nn*< *q*2.

Отсюда вытекает правило формирования коэффициентов полинома (1.3) или разрядов записи заданного числа *N* в системе счисления с основанием *q2*:

- необходимо разделить исходное число *N* *q*1 на новое основание *q*2, при этом получив целое частное и остаток;

- полученный остаток снова необходимо разделить на *q*2, процесс деления продолжается до тех пор, пока частное будет не меньше нового основания *q*2*.* Если очередное сформированное частное будет меньше, чем *q*2, то процесс формирования записи заданного числа в новой системе с основанием *q*2 считается законченным, а в качестве искомых разрядов новой записи числа используются результаты выполненных операций деления следующим образом:

- в качестве старшего разряда берется значение последнего частного, для остальных разрядов используются значения остатков в порядке, обратном порядку их получения.

***2. Преобразование дробных чисел***

Задача представления дробного числа *Mq*1, заданного в системе *q*1, в системе счисления с основанием *q*2, можно рассматривать как задачу поиска коэффициентов полинома, представляющего собой расширенную запись числа *M* в системе счисления *q*2:

*B*1 *× q*2*–*1 *+ B*2 *× q*2*–*2 *+ B*3 *× q*2*–*3 *+ ....* *+ Bn–*2 *× q*2*–(n–*2*) + B n–*1 *× q*2*–(n–*1*) + Bn × q*2*–n* = *Mq* (1.4)

Введем скобочную форму для выражения (1.4). Обозначим выражение в первой скобке как *M*1, выражение во второй *–* как *M*2, в третьей скобке *–* как *M*3 и т. д., выражение в (*n –* 1)*-*й скобке как *M* *n–*1, выражение в *n-*й скобке – как *Mn*:

*Mq*2 *= Mq*1 *= q*2*–*1(*B*1 *+ q*2*–*1(*B*2 *+ q*2*–*1(*B*3*+...+q*2*–*1(*Bn–*1 *+ q2–*1(*Bn*))...))).

Число *Mq*1 *–* правильная дробь, поэтому при умножении *Mq*1 *×* *q*2 будет получено произведение, в общем случае состоящее из целой части int(*Mq*1 *× q*2) и дробной части DF (*M* *q*1 *×* q2). Использование введенных обозначений позволяет записать:

*Mq*1 *×* *q*2 *=* (int(*M* *q*1 *× q*2) *= B*1) *+* (DF(*Mq*1 *×* *q*2) *= M*1 ),

аналогично для остальных скобок будем иметь следующее:

*M*1 *× q*2 = (int(*M*1 *× q*2) = *B*2) + (DF(*M*1 *× q*2) = *M*2);

*M*2 *× q*2 = (int(*M*2 *× q*2) *= B*3) + (DF(*M*2 *× q*2) *= M*3);

*M*3 *× q*2 = (int*(M*3 *× q*2) *= B*4) *+* (DF(*M*3 *× q*2) *= M*4);

*Mn–*2 *× q*2 = (int(*Mn*–2 *× q*2) *= Bn–*1) *+* (DF*(Mn–*2 *× q*2) *= Mn–*1);

*Mn–*1*× q*2 *=* (int(*Mn–*1 *× q*2) *= Bn*) *+* (DF(*Mn–*1 *× q*2) *= Mn*);

*Mn × q*2 = (int(*Mn× q*2) *= Bn+*1) *+* (DF(*Mn × q*2) *= Mn*+1).

Отсюда вытекает следующее правило формирования коэффициентов полинома, которые одновременно являются разрядами записи заданного числа *M* в системе счисления с основанием *q*2:

- определяется количество разрядов «*n*» в записи числа *Mq*2 в новой системе счисления;

- исходное число *Mq*1 умножается на *q*2, при этом будет получено смешанное число;

- дробная часть полученного произведения снова умножается на *q*2 и т. д.; процесс умножения повторяется *n* +1 раз. В качестве искомых разрядов новой записи числа используются результаты выполненных операции деления следующим образом:

- в качестве первого старшего разряда искомой записи числа в новом основании берется значение целой части первого произведения, в качестве второго старшего разряда искомой записи числа в новом основании берется значение целой части второго произведения и т. д.

# ***3.Перевод чисел из одной системы счисления в другую. Метод с использованием особого соотношения оснований исходной и искомой систем счисления.***

Данный метод применим в тех случаях, когда исходное *q*1 и новое *q*2 основания могут быть связаны через целую степень, т.е. когда выполняются условия: *q*1*m* = *q*2 (*условие 1*) или *q*2 *m* = *q*1 (*условие 2*). Если имеет место *условие 2*, то для заданного в системе с основанием *q*1 числа *Nq*1 *= аn аn*-1 *аn*-2... *а*1*а*0 запись его в системе с новом основании *q*2 определяется следующим образом:

- каждому разряду *ai* исходной записи числа ставится в соответствие его *m-*разрядный эквивалент в системе счисления с основанием *q*2;

- искомая запись всего заданного числа формируется за счет объединения всех полученных *m*-разрядных групп.

Если имеет место *условие 1*, то запись заданного числа *N* =*аnаn*-1*аn-*2...*а*1*а*0 в системе с новом основании *q*2 формируется следующим образом:

- исходная запись числа разбивается на группы по *m* разрядов, двигаясь от точки вправо и влево (недостающие разряды в крайних группах (слева и справа) дополняются нулями;

- каждой полученной группе ставится в соответствие цифра новой системы счисления;

- искомая запись заданного числа в новой системе счисления образуется из цифр, соответствующих группам, на которые была разбита исходная запись.

# **4.Арифметические операции над двоичными числами. Операция сложения и вычитания в 2-й СС**

При выполнении любой операции результат ищется согласно соответствующим правилам, которые удобно представлять в табличной форме, где для всех возможных комбинаций значений одноразрядных операндов приводятся значения результата.

Сложение. В общем случае при формировании значения в текущем разряде результата приходится дважды применять приведенную таблицу сложения: первый раз при сложении соответствующих разрядов операндов, формируя так называемую поразрядную сумму, и второй раз *–* при сложении разряда сформированной поразрядной суммы и переноса, пришедшего из ближайшего младшего разряда.

При машинной реализации операции сложения сначала формируется поразрядная сумма операндов без учета переноса, далее формируется код переноса и затем с помощью специальных логических цепей учитываются возникшие переносы. При этом перенос, возникший в некотором разряде, может изменить не только ближайший старший разряд, но и целую группу старших разрядов. В худшем случае перенос, возникший в самом младшем разряде, может изменить значение старших разрядов сформированной поразрядной суммы вплоть до самого старшего.

При формировании поразрядной суммы и учете возникших переносов используется следующая классификация разрядов складываемых операндов:

- разряд, генерирующий перенос (оба операнда в этом разряде имеют «1»);

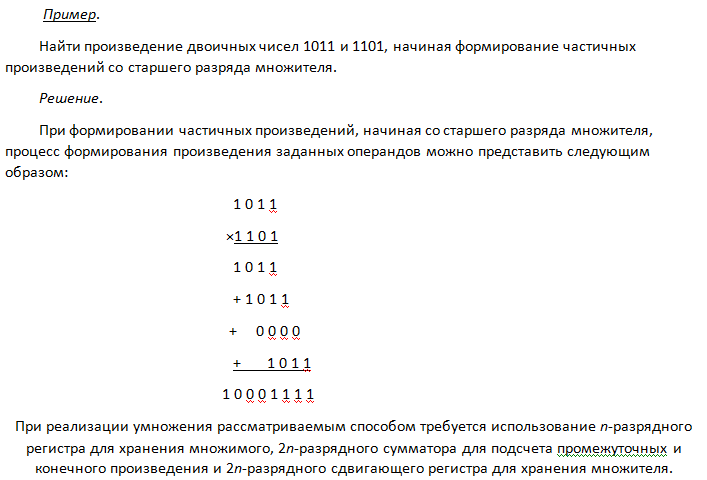
- разряд, пропускающий перенос (операнды в этом разряде име­ют разные значения);

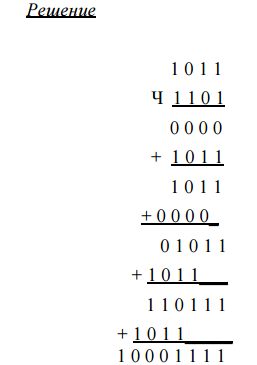
- разряд, блокирующий распространение переноса (операнды в этом разряде имеют одинаковые значения).

Вычитания. В ЭВМ никогда в перечне выполняемых операций арифметического устройства не присутствует одновременно операция сложения и операция вычитания. При этом, как правило, присутствует только операции сложения. Что же касается операции вычитания, то она реализуется за счет прибавления к уменьшаемому значения вычитаемого,   
взятого с противоположным знаком.

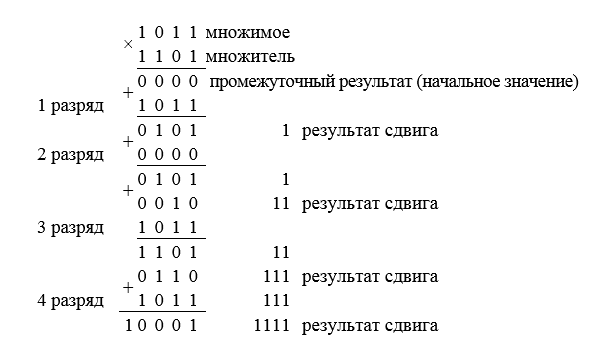
# **5.Операция умножения в двоичной системе исчисления**

*начиная со старшего разряда множителя со сдвигом множимого.*

*начиная с младшего разряда множителя со сдвигом множимого.*

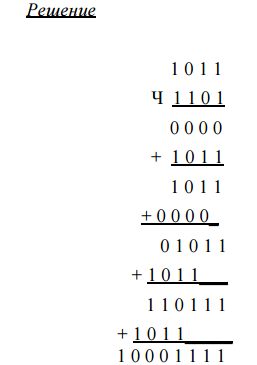
**

*начиная со старшего разряда множителя со сдвигом промежуточных результатов*



В первой колонке приведены номера отрабатываемых разрядов множителя начиная с младшего. Эти номера отмечают строки, в которых учитывается частичное произведение, соответствующее этому разряду множителя. В этой же первой колонке расположены единицы переполнения, возникающие при суммировании промежуточного результата и очередного частичного произведения, сформированного для соответствующего отрабатываемого разряда множителя (в данном случае единица переполнения имеется при отработке четвертого, самого старшего, разряда множителя). Вторая колонка отражает длину основной разрядной сетки (n = 4). В третьей колонке представлены разряды промежуточных и конечного произведений, «вытолкнутых» за пределы основной разрядной сетки в процессе выполнения очередного сдвига промежуточного произведения. Из приведенного примера видно, что выталкиваемые за пределы разряды промежуточных произведений в дальнейшем не изменяются и их значение не влияет на значение суммы, формируемой в пределах основной разрядной сетки. Поэтому для реализации этого метода умножения требуется n-разрядный сумматор, обеспечивающий суммирование только в пределах основной разрядной сетки. Аналогично умножению начиная с младшего разряда множителя, при умножении со старших разрядов можно заменить сдвиг вправо множителя на сдвиг влево промежуточного произведения. Операция умножения в общем случае дает точный результат – 2nразрядное произведение, где n-разрядность операндов.

н*ачиная с младшего разряда множителя со сдвигом частичных произведений*



# **6. IEEE754. Специальные числа. Зачем нулю знак.**

Разработчики «K-C-S» победили и теперь их детище воплотилось в стандарт IEEE754. Числа с плавающей запятой в нем представлены в виде знака (s), мантиссы (M) и порядка (E) следующим образом:(-1)s × 1.M × 2E  
 *Замечание.* В новом стандарте IEE754-2008 кроме чисел с основанием 2 присутствуют числа с основанием 10, так называемые *десятичные* (decimal) числа с плавающей запятой.  
 тип данных с одинарной точностью (float). Числа с половинной, двойной и расширенной точностью обладают теми же особенностями, но имеют другой диапазон порядка и мантиссы. В числах одинарной точности (float/single) порядок состоит из 8 бит, а мантисса – из 23. Эффективный порядок определяется как E-127. Например, число 0,15625 будет записано в памяти как   
  
  
В этом примере:

-Знак s=0 (положительное число)

-Порядок E=011111002-12710 = -3

-Мантисса M = 1.012 (первая единица не явная)

-В результате наше число F = 1.012e-3 = 2-3+2-5 = 0,125 + 0,03125 = 0,15625

**Чуть более подробное объяснение**

Здесь мы имеем дело с двоичным представлением числа «101» со сдвигом запятой на несколько разрядов влево. 1,01 — это двоичное представление, означающее 1×20 + 0×2-1 + 1×2-2. Сдвинув запятую на три позиции влево получим 1,01e-3 = 1×2-3 + 0×2-4 + 1×2-5 = 1×0,125 + 0×0,0625 + 1×0,03125 = 0,125 + 0,03125 = 0,1562

**Специальные числа: 0, бесконечность и неопределенность**  
 В IEEE754 число «0» представляется значением с порядком, равным E=Emin-1 (для single это -127) и нулевой мантиссой. Введение нуля как самостоятельного числа (т.к. в нормализованном представлении нельзя представить ноль) позволило избежать многих странностей в арифметике. И хоть операции с нулем нужно обрабатывать отдельно, обычно они выполняются быстрее, чем с обычными числами.   
 Также в IEEE754 предусмотрено представление для специальных чисел, работа с которыми вызывает исключение. К таким числам относится бесконечность (±∞) и неопределенность (NaN). Эти числа позволяет вернуть адекватное значение при переполнении. Бесконечности представлены как числа с порядком E=Emax+1 и нулевой мантиссой. Получить бесконечность можно при переполнении и при делении ненулевого числа на ноль. Бесконечность при делении разработчики определили исходя из существования пределов, когда делимое и делитель стремиться к какому-то числу. Соответственно, c/0==±∞ (например, 3/0=+∞, а -3/0=-∞), так как если делимое стремиться к константе, а делитель к нулю, предел равен бесконечности. При 0/0 предел не существует, поэтому результатом будет неопределенность.  
 *Неопределенность* или NaN (от not a number) – это представление, придуманное для того, чтобы арифметическая операция могла всегда вернуть какое-то не бессмысленное значение. В IEEE754 NaN представлен как число, в котором E=Emax+1, а мантисса не нулевая. Любая операция с NaN возвращает NaN. При желании в мантиссу можно записывать информацию, которую программа сможет интерпретировать. Стандартом это не оговорено и мантисса чаще всего игнорируется.   
 Как можно получить NaN? Одним из следующих способов:

-∞+(- ∞)

-0 × ∞

-0/0, ∞/∞

-sqrt(x), где x<0

По определению NaN ≠ NaN, поэтому, для проверки значения переменной нужно просто сравнить ее с собой.

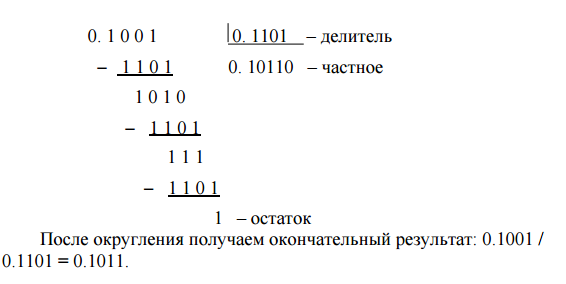
**Зачем 0 знак (или +0 -0)**  
 Любознательный читатель вероятно уже заметил, что в описанном представлении чисел с плавающей запятой существует два нуля, которые отличаются только знаком. Так, 3·(+0)=+0, а 3·(-0)=-0. Но при сравнении +0=-0. В стандарте знак сохранили умышленно, чтобы выражения, которые в результате переполнения или потери значимости превращаются в бесконечность или в ноль, при умножении и делении все же могли представить максимально корректный результат. Например, если бы у нуля не было знака, выражение 1/(1/x)=x не выполнялось бы верно при x=±∞, так как 1/∞ и 1/-∞ равны 0.

# **7.Деление двоичных чисел (общие правила).**

Деление в принципе является неточной операцией, поэтому при её выполнении прежде всего устанавливается количество разрядов частного, которые подлежат определению. Деление в двоичной системе счисления может выполняться точно так же, как и в десятичной, однако формирования частного двоичных операндов реализуется гораздо проще, чем в десятичной системе, т.к.:

• Упрощается процедура подбора очередной цифры вследствие того, что в двоичной системе очередной цифрой может быть одна из двух - либо 0, либо 1;

• Упрощается процедура умножения найденной цифры частного на делитель.



**10.Двоично-десятичная арифметика. Сложение и вычитание двоично-десятичных чисел .**

В ЭВМ часто предусматривается обработка чисел не только в двоичной системе счисления, но в двоично-десятичной. При этом, как правило, стремятся реализовать двоично-десятичную арифметику по правилам двоичной с введением ограниченного количества коррекций.

\* \*\*

0011 1001 0010 0111 – D

+ 0100 1000 0101 0110 – C

1000 0001 0111 1101 – двоичная сумма

+ 0110 0110 – коррекция

1000 0111 1000 0011 – двоично-десятичная сумма

Для получения двоично-десятичной суммы A на основании результата сложения операндов по правилам двоичной арифметики необходимо добавить шестерку (0110) в те тетрады, из которых был перенос. В данном примере это вторая тетрада (отмечена \*). Необходимость такой коррекции обусловливается тем, что перенос, сформированный по правилам двоичного суммирования, унес из тетрады шестнадцать, а для десятичного сложения перенос должен был унести десять, т.е. перенос, сформированный по правилам двоичной арифметики, унес лишнюю шестерку. Кроме этого шестерка добавляется в те тетрады, в которых получено значение, большее девяти. Такая коррекция обуславливается тем, что по правилам десятичной арифметики в таких тетрадах должен быть выработан перенос и, чтобы его выработать по правилам двоичной арифметики, в тетраду нужно добавить шестерку. Для рассмотренного примера такой тетрадой является и четвертая тетрада (отмечена \*\*).

Пример

Найти разность двух десятичных чисел с использованием двоично-десятичной системе счисления:

A = C - D, где D = 3927, C = 4856.

Решение

Составляем двоично-десятичную запись для чисел D и C:

D = 3927 = 0011 1001 0010 0111:

C = 4856 = 0100 1000 0101 0110.

Найти значение В можно, реализовав следующую последовательность операций из двоичного сложения и операции коррекции:

\* \*

0100 1000 0101 0110 – C

- 0011 1001 0010 0111 – D

0000 1111 0010 1111 – двоичная сумма

- 0110 0110 – коррекция

0000 1001 0010 1001 – двоично-десятичная сумма

Для получения двоично-десятичной разности «A» на основании результата вычитания операндов по правилам двоичной арифметики необходимо вычесть шестерку (0110) из тетрад, в которые пришел заем. Это обусловливается тем, что заем, сформированный по правилам двоичного вычитания, приносит в тетраду шестнадцать, а для десятичного сложения заем должен был принести в тетраду десять, т.е. заём, сформированный по правилам двоичной арифметики, принес лишнюю шестерку. Для рассмотренного примера тетрадами, в которые пришел заем и в которых необходимо выполнить коррекцию (вычесть шестерку), являются вторая и четвертая тетрады (отмечены \*).

# **11.Кодирование алгебраических чисел . Дополнительный и обратный коды двоичных чисел**

Для представления чисел со знаком используются специальные коды:

- прямой код;

- дополнительный код;

- обратный код.

Во всех трёх случаях используется следующий формат представления числа, содержащий два поля - поле знака и поле модуля*. Поле знака* представлено одним разрядом, в котором устанавливается 0, если число положительное, и 1, если число отрицательно. *Поле модуля* отражает количественную оценку числа и для каждого кода формируется по*–*разному. Количество разрядов поля модуля определяется диапазоном изменения отображаемых чисел или точностью их представления.

Для случая правильной дроби запись числа *А* в дополнительном коде имеет вид:



где 1 – максимальная невключенная граница диапазона изменения представляемых чисел, т. е. диапазон изменение чисел *А* определяется как 1 > |*A* | ≥ 0.

Для случая правильной дроби запись числа *А* в обратном коде имеет вид



где *n –* разрядность поля модуля;

1 *- q-n* – верхняя включенная граница представляемых чисел.

Т. о., диапазон изменения чисел *А* определяется с. о.:

(1 *– q-n*) ≥ | *А* | ≥ 0.

При переводе двоичных чисел в качестве включенной и не включенной границы диапазона изменения абсолютных значений представляемых чисел используется соответственно 2n и 2n - 1.

Представление двоичных чисел в прямом и обратном кодах поясняется следующими примерами.

**Пример**

Найди запись чисел **А = 532 и В = - 150** в прямом, дополнительном и обратном двоичных кодах.

**Решение**

Найдем **запись** заданных **чисел в двоичной системе:**

А = 532 = 10000101002, В = - 150 = - 100101102.

Если считать, что представляются в заданных кодах только А и В, то разрядность n модульного поля должна соответствовать разрядности двоичной записи большего числа, т.е. n = 10.

Найдем запись заданных чисел **в прямом коде:**

[A]пк = 0.1000010100, [B]пк = 1.0010010110.

Найдем запись заданных чисел в **дополнительном коде:**

[A]дк = 0.1000010100,

[B]дк: для определения модульной части **прибавим к невключенной границе диапазона** (2n = 10000000000) **число В:**

10000000000 +(-10010110) = 1101101010.

Тогда [B]дк = 1.1101101010.

Найдем запись **заданных чисел в обратном коде**:

[A]ок = 0.1000010100,

[B]ок: для определения модульной **части прибавим к включенной границе диапазона** (2n - 1= 1111111111) **число В**:

1111111111 + (- 10010110) = 1101101001,

и тогда [B]ок = 1.1101101001.

Обратный код **положительного** числа **совпадает с его прямым кодом,** обратный код **отрицательного** числа формируется **по правилам:** в знаковом разряде **записывается “1**”; цифровые **значения меняются** на противоположные(**0 на 1,1 на 0),т.е.** необходимо **проинвертировать модуль прямого кода.**

Переход от обратного кода отрицательного числа к представлению в прямом коде осуществляется по тому же правилу, т.е. необходимо проинвертировать модуль записи числа в дополнительном коде.

*Дополнительный код числа*, имеет такое же назначение, как и обратный код числа. Формируется по следующим правилам: положительные числа в дополнительном коде выглядят также как и в обратном и в прямом коде, т.е. не изменяются. Отрицательные числа кодируются следующим образом: к обратному коду отрицательного числа (к младшему разряду) прибавляется 1, по правилу двоичной арифметики.

Переход от дополнительного кода отрицательного числа к прямому осуществляется по тому же правилу, т.е. необходимо проинвертировать модуль записи числа в дополнительном коде, и к полученному коду прибавить 1 в младший разряд.

При выполнении операций над числами со знаком в ЭВМ используются прямой, обратный и дополнительный коды. Как правило, информация в памяти хранится в прямом коде, а при выполнении операций применяется или обратный, или дополнительный код.

# **12. Операции с двоичными числами в дополнительном коде и в обратном коде**

При использовании дополнительного или обратного кода операция вычитания заменяется операцией сложения с изменением знака второго операнда. При сложении чисел, представленных в дополнительном коде, выполняется сложение разрядов по правилам двоичной арифметики по всей длине записи чисел, не обращая внимание на границу, разделяющую знаковое и модульные поля. Переполнение знакового поля игнорируется!

1. Исходя из абсолютного значения операндов, разрядность представления модульной части n должна быть равна количеству разрядов большего операнда. Учитывай то, что мы используем две операции: сложение и вычитание, поэтому возможно переполнение из-за переноса из старшего разряда, БЕРИ длину модульной части на один разряд больше, т.е n+1

Пример: у тебя есть число в двоичном представлении

A=100110 оно имеет 6 разрядов

B=11010101 оно имеет 8 разрядов  
Количество разрядов между двумя операндами должно совпадать, поэтому в прямом коде выглядит так:

A=0.**00**100110

B=0.11010101

2. Избавься от операции вычитания, т.е если есть такая запись:  
С1 = А + В, С2 = А – В, С3 = В – А, С4 = –А – В, то нужно сделать так:  
С1 = А + В, С2 = А + (–В), С3 = В + (–А), С4 = (–А) + (-В)

Таким образом, у тебя будут величины: A, –A, B, –B

3. Далее свои величины A и B представляешь в дополнительном коде.(инвертировать +1)

4. Используя сформированный дополнительный код, реализуем выражения для С1, С2, С3, С4.   
**НЕ ЗАБЫВАЙ!** Если при выполнении сложения у тебя возникла единица переполнения знакового поля. При работе с дополнительным кодом она игнорируется (в примере она подчёркнута).

С4:

1. 1 1 1 0 0 0 1 1 1 – [–A]дк

* 0. 0 1 1 0 1 0 0 1 0 – [–В]дк

-------------------------

**1** 0. 0 1 0 0 1 1 0 0 1 – [–С4]дк

КОНЕЦ.

**обратном коде**

При сложении чисел, представленных в *обратном* коде, выполняется сложение разрядов, представляющих запись операндов, по правилам двоичной арифметики по всей длине записи чисел, не обращая внимания на границу, разделяющую знаковое и модульные поля. Переполнение знакового поля, т.е. перенос, возникший из крайнего левого разряда, должен быть учтен как +1 в младший разряд полученной суммы. В результате такого сложения будет получен *обратный* код суммы заданных операндов.

**Пример**

Найти значения для *С*1, *С*2, *С*3, *С*4, определяемых выражениями

*С*1 = *А* + *В*, *С*2 = *А* - *В*, *С*3 = *В* - *А*, *С*4 = - *А* - *В*,

если А=5710, В = - 21010. При выполнении операций использовать двоичный обратный код. Результат представить в прямом коде.

**Решение**

В данном примере используются те же выражения и те же операнды, что и в предыдущем примере, поэтому при его решение используются уже найденные ранее двоичные представления операндов и их прямые коды.

Обратные коды операндов имеют вид

[*А*]ок = 0.000111001, [-*А*]ок = 1.111000110,

[*В*]ок = 1.100101101, [-*В*]ок = 0.011010010.

# **13. Модифицированные коды**

При расчете разрядности n модульного поля весьма трудно бывает учесть диапазон значений результатов, особенно когда последовательность операции, представленных в подлежащих реализации выражениях, достаточно сложны.

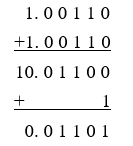
При несоответствии выбранной разрядности n диапазону изменения представляемых чисел при выполнении операции сложении чисел с одинаковыми знаками возможно появление ситуации переполнения, когда подлежащий представлению результат выходит за диапазон представления, определенный некорректно выбранной разрядностью n поля модуля.

Например, в случае сложения двух чисел, представленных в обратном коде:

[D1]ок = 1.00110 и [D2]ок = 1.00110.

Сумма этих чисел F1 = D1 + D2 будет подсчитана следующим образом:

F1:



Пример, выполненный по всем формальным правилам, дал абсурдный результат, так как получена положительная сумма двух отрицательных операндов. Аналогичная ситуация может возникать и при использовании дополнительного кода.

Ситуацию переполнения можно обнаруживать по факту появления «абсурдного» результата, но для этого необходимо помнить то, что в суммировании принимают участие операнды с одинаковыми знаками и знак полученного при этом результата отличен от знака операндов.

Более просто ситуация переполнения определяется при применении модифицированного кода (обратного или дополнительного). Модифицированные коды отличаются от базовых кодов только тем, что поле знака операндов имеет два разряда, и эти разряды имеют одинаковые значения:

00 – для положительных чисел;

11 – для отрицательных чисел.

Если в результате сложения чисел в модифицированном коде полученный результат имеет в поле знака одинаковые значения в обоих разрядах (00 или 11), то переполнения нет, если же разряды знакового поля имеют не одинаковые значения (10 или 01), то имеет место переполнение. При этом, если в поле знака имеет место значение 01 – результат положительный, а если 10, то полученный результат отрицательный (основным носителем знака числа является левый разряд знакового поля).

Пример

Найти значения выражений

С1 = А + В, С2 = А - В, С3 = В - А, С4 = - А - В,

используя модифицированный обратный код, если

[А]пк = 0.1010011,

[B]пк = 1.0111001.

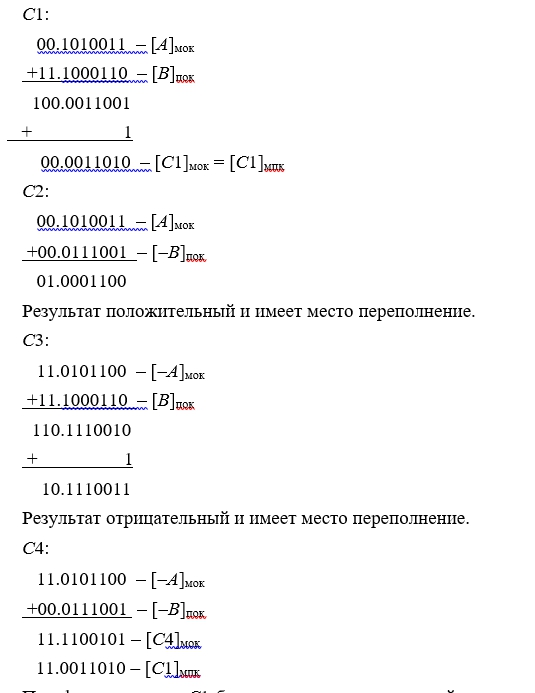
Решение

Модифицированный обратный код для всех операндов, используемых в приведенных выражениях, имеет вид

[А]мпк = 00.1010011, [А]мок = 00.1010011, [-А]мок = 11.0101100,

[B]мпк = 11.0111001, [B]мок =11.1000110, [-B]мок = 00.0111001.

Выполним действия, указанные в приведенных выражениях:



При формировании С1 был получен положительный результат (без переполнения).

При формировании С4 был получен отрицательный результат (без переполнения).

Факт переполнения при формировании С3 и С2 устанавливается по наличию в разрядах знакового поля различных значений.

**14. Логические операции с двоичными кодами: логическое суммирование, логическое умножение, логическое отрицание, суммирование по модулю два, логические сдвиги.**

Над двоичными кодами могут выполняться различные логические операции, среди которых особое место занимают:

1) *логическое суммирование* (обозначения – ИЛИ, ОR, «∨»);

2) *логическое умножение* (обозначения – И, AND, «∧»);

3) *отрицание* (обозначения – НЕТ, NOT, «*x*», т.е. штрих над отрицаемым *x*);

4) *суммирование по модулю* 2 (обозначается mod 2, « ⊕ »);

5) *операции сдвига*.

**4.1. Логические операции**

Операция *логического суммирования* выполняется над двумя кодами и генерирует код той же разрядности, что и операнды, у которого в некотором *i*-м разряде находится единица, если хотя бы в одном операнде в *i*-м разряде имеет место единица.

*Пример:*

10001101 ∨ 11110000 = 11111101.

Операция *логического умножения* выполняется над двумя кодами и генерирует код той же разрядности, что и операнды, у которого в некотором *i*-м разряде находится единица, если оба операнда в этом *i*-м разряде имеются единицу, и ноль во всех других случаях.

*Пример*:

10001101 ∧ 11110000 = 10000000

Операция *суммирования по модулю 2* выполняется над двумя кодами и генерирует код той же разрядности, что и операнды, у которого в некотором *i*-м разряде находится единица, если два заданных операнда в *i*-м разряде имеют противоположные значения. Иногда эта операция называется «исключающее ИЛИ».

*Пример:*

10001101 ⊕ 11110000 = 01111101.

Операция логического отрицания выполняется над одним кодом и генерирует результирующий код той же разрядности, что и операнд, в некотором *i*-м разряде которого находится значение, противоположное значению в *i*-м разряде отрицаемого кода.

Операции *сдвига* в свою очередь, подразделяются на:

1) логические сдвиги, которые имеют разновидности – сдвиг вправо, сдвиг влево, циклический сдвиг вправо, циклический сдвиг влево;

2) арифметические сдвиги вправо и влево, выполнение которых зависит от знака и кода сдвигаемого числа.

# **15. Арифметические сдвиги положительных двоичных чисел, представленных в прямом коде. . Арифметические сдвиги двоичных чисел, представленных в обратном коде.**

Для представления чисел со знаком используются специальные коды:

– прямой код;

– дополнительный код;

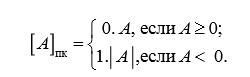
– обратный код.

Во всех трех случаях используется следующий формат представления числа, содержащий два поля – поле знака и поле модуля

|  |  |
| --- | --- |
| Поле знакаПоле знака | Поле модуля |

Общий формат представления числа в ЭВМ

В прямом коде запись целого числа А формируется по следующему правилу:



Сдвиг, при котором уходящий бит исчезает, не влияя на оставшиеся биты, а на месте появившегося бита записывается бит **0**. В большинстве процессоров уходящий бит сохраняется вофлаге переноса.

Арифметические сдвиги обеспечивают выполнение умножения (сдвиги влево) или операции деления (сдвиги вправо) двоичных кодов на два, точно так же, как сдвиги вправо и влево десятичного числа обеспечиваю выполнение деления и умножение на 10. Если сдвигается положительное число, то сдвиг (вправо или влево) выполняется как соответствующий логический сдвиг (влево или вправо), с той лишь разницей, что предусматриваются средства определения факта переполнения при сдвиге влево, что реализуется и при всех других арифметических операциях. При любом сдвиге вправо предусматриваются средства для округления после завершения нужного количества сдвигов и средства обнаружения обнуления сдвигаемой величины после очередного сдвига. Арифметические сдвиги влево положительных двоичных чисел выполняются независимо от используемого кода (прямого, обратного, дополнительного).

Пример: Найти результат арифметического сдвига влево на три разряда двоичного прямого кода числа [А]пк = 00.00000101

Решение: Процесс выполнения заданного сдвига дает следующие промежуточные и конечное значения:

первый сдвиг: 00.00000101 ← 00.00001010;

второй сдвиг: 00.00001010 ← 00.00010100;

третий сдвиг: 00.00010100 ← 00.00101000.

**двоичных чисел**

При арифметическом сдвиге влево отрицательного двоичного числа, представленного в обратном коде, осуществляется циклический сдвиг исходного кода с контролем за переполнением, например, сдвиг влево отрицательного двоичного числа 11.1100110 (2510), представленного в обратном коде, дает в результате 11.1001101 (5010).

При арифметическом сдвиге вправо отрицательного двоичного числа, представленного в обратном коде, осуществляется сдвиг только модульной части записи числа с установкой единицы в освобождающийся разряд. При этом может осуществляется контроль за обнулением результата сдвига (появление единичных значений во всех разрядах) и округление результата после выполнения заданного количества сдвигов.

*Пример3*

Выполнить сдвиг вправо на четыре разряда двоичного числа 11.1001101 (десятичный эквивалент – 5010), представленного в обратном коде.

Первый сдвиг дает 11.11001101 (5010) à 11.11100110 (2510).

Второй сдвиг дает 11.11100110 (2510) à 11.11110011 (1210).

Третий сдвиг дает 11.11110011 (1210) à 11.11111001 (610).

Четвертый сдвиг дает 11.11111001 (610) à 11.11111100 (310).

При выполнении сдвига вправо нечетного числа результат получается с точностью до младшего разряда кода, причем ошибка отрицательная.

После выполнения последнего, четвертого сдвига выполняется округление, при котором, если последний «вытолкнутый» разряд имел значение 0, к результату последнего сдвига прибавляется – 1.

Данное округление можно выполнить за счет прибавления единицы к прямому коду, соответствующему результату последнего сдвига исходного обратного кода.

В рассмотренном примере корректировать на единицу результат четвертого сдвига не надо, так как «вытолкнутый» разряд при последнем (четвертом) сдвиге равен единице. В данном случае конечный результат сдвига заданного отрицательного числа, представленного в обратном коде, равен 11.11111100.

# **16.Арифметические сдвиги двоичных чисел, представленных в дополнительном коде. Сдвиг отрицательных чисел с переполнением.**

При арифметическом сдвиге влево отрицательного двоичного числа, представленного в дополнительном коде, осуществляется логический сдвиг влево модуля исходного кода (освобождающийся разряд заполняется нулем) с контролем за переполнением, например, сдвиг влево отрицательного двоичного числа 11.11001110 (50 в 10 с/c), представленного в дополнительном коде, дает в результате 11.10011100 (100 в 10 с/с).

При арифметическом сдвиге вправо отрицательного двоичного числа, представленного в дополнительном коде, осуществляется логический сдвиг вправо модуля записи числа с установкой единицы в освобождающийся разряд. При этом может осуществляется контроль за обнулением результата сдвига (появление единичных значений во всех разрядах).

**Пример**

Выполнить сдвиг вправо на четыре разряда двоичного числа 11.11001110 (десятичный эквивалент – 50 в 10 с/c), представленного в дополнительном коде.

Решение

Первый сдвиг дает 11.11001110 → 11.11100111 (25 в 10 с/c).

Второй сдвиг дает 11.11100111 → 11.11110011 (13 в 10 с/c).

Третий сдвиг дает 11.11110011 → 11.11111001 (7 в 10 с/c).

Четвертый сдвиг дает 11.11111001 → 11.11111100 (4 в 10 с/c).

При выполнении сдвига вправо нечетного целого числа результат получается с точностью до младшего разряда кода, причем ошибка положительная.

Арифметический сдвиг вправо может выполняться над отрицательными числами с переполнением (такие числа в модифицированном прямом, обратном или дополнительном коде имеют в знаковом поле 10). В этом случае после сдвига в знаковом поле будет 11, а в старшем разряде – 0, если число представлено в обратном или дополнительном коде, или 1, если число представлено в прямом коде.

**Пример1**

Выполнить сдвиг вправо на 2 разряда числа [А]пк = 10.01000110 (А10 = 326).

**Решение**

1-й сдвиг: 10. 01000110 → 11.10100011 (-163 в 10 с/c);

2-й сдвиг: 11.10100011 → 11.11010001 (-81 в 10 с/c) и последний вы-толкнутый разряд равен 1).

С учетом округления имеем окончательный результат:

[А2]пк=11.100100101

**Пример2**

Выполнить сдвиг вправо на 2 разряда числа [А]ок = 10.10111001 (А10 = – 326).

**Решение**

1-й сдвиг: 10.10111001 → 11.01011100 (-163 в 10 с/c);

2-й сдвиг: 11.01011100 → 11.10101110 (-82 в 10 с/c).

**Пример3**

Выполнить сдвиг вправо на 2 разряда число [А]ок= 10. 1011101

(А10 = - 326).

**Решение**

1-й сдвиг: 10.0111010 → 11.01011101 (-163 в 10 с/c);

2-й сдвиг: 11.01011101 → 11.10101110 (-81 в 10 с/c) и последний вытолкнутый разряд равен 1).

С учетом округления имеем окончательный результат

[А2]ок = 11.10101101.

# **17. Представление чисел с фиксированной точкой. Арифметические операции над числами, представленными с фиксированной точкой.**

Числовая информация представляется в машине в форме с фиксированной или с плавающей точкой. При представлении с фиксированной точкой положение последней в записи числа фиксировано.Как правило, при использовании фиксированной точки числа представляются в виде целого числа или правильной дроби, форматы которых приведены на рис. 1.3.К заданному виду (целым числам или правильной дроби) исходные числа приводятся за счет введения масштабных коэффициентов.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Зн | 1р | 2р | 3р | 4р | .... | (*n*-1) | *n*р | «.» |

*а*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Зн | «.» | 1р | 2р | 3р | 4р | .... | (*n*­1) | *n*р |

*б*

Рис. 1.3. Представление чисел с фиксированной точкой: *а* – формат целого числа; *б –* формат дробного числа

Точка в записи числа не отображается, а так как она находится всегда в одном месте, то указание на её положение в записи числа отсутствует. При *n*-разрядном представлении модульной части формат с фиксированной точкой обеспечивает диапазон изменения абсолютного значения числа *А*, для которого выполняется неравенство2*n* > ú*A*ú ≥ 0.

Одним из важнейших параметров представления чисел является ошибка представления. Ошибка представления может быть абсолютной (D) или относительной (d). Для фиксированной точки максимальные значения этих ошибок определяются следующим образом.

В случае целых чисел:

Dmax = 0.5; dmax = Dmax / *А* min = 0.5, где *А*min – минимальное, отличное от нуля, значение числа.

В случае дробных чисел: Dmax = 0.5×2*n* = 2(*n*+1); dmax = Dmax / *А*min = 2(*n*+1) / 2-*n* = 0.5,

т.е. в худшем случае относительная ошибка при фиксированной точке может достигать сравнительно большого значения – 50%.

5.2. Арифметические операции над числами, представленными с фиксированной точкой

К числу основных арифметических операций, непосредственно реализуемых в ЭВМ, относятся операции сложения, умножения, деления. Остальные операции (например, такие, как возведение в степень, извлечение квадратного корня) реализуются программным способом.

Выполнение операций с числами, представленными с фиксированной точкой, рассмотрено в рамках материала по выполнению операций с алгебраическими числами (подраздел 1.3). Выполнение длинных операций, таких, как умножение и деление, реализуется в два этапа:

– на первом этапе формируется знак искомого результата,

– на втором этапе, используя абсолютные значения операндов, ищем результата (произведение или частное), которому затем присваивается предварительно определенный знак.

Операнды, как правило, представлены в прямом коде, и знак результата, не зависимо от того, частное это или произведение, ищется за счет сложения по модулю 2 знаковых разрядов операндов. В результате этого знак результата положителен, если операнды имеют одинаковые знаки, или отрицательный, если операнды имеют разные знаки.

# **18.Представление чисел с плавающей точкой. Сложение чисел, представленных в формате с плавающей точкой.**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1р | 2р. | 3р. | ... | *к* р. | «.» | (*к*+1) р. | (*к*+2) р. | ... | (*n*­1) р. | *n*р |

При представления числа *с плавающей точкой* число в общем случае представляет собой смешанную дробь и имеет формат, приведенный на рис. 1.4

Рис. 1.4. Формат представления числа с плавающей точкой

Местоположение точки в записи числа может быть различным, а так как сама точка в записи числа не присутствует, то для однозначного задания числа необходима не только его запись, но и информация о том, где в записи числа располагается точка, отделяющая целую и дробную части.

Поэтому в случае с плавающей точкой число *Х* представляется в виде двух частей:

*мантисса* (*х*м), отображающая запись числа, представляется в виде правильной дроби с форматом фиксированной точки;

*порядок* (*х*п), отображающий местоположение в этой записи точки, представляется в виде целого числа с форматом фиксированной точки.

Количественная оценка числа *Х* определяется как

*Х* = *qx*п × *х*м ,

где *q* – основание системы счисления.

Для двоичной системы счисления имеет место

*Х* = *2x*п × *х*м.

При *s*-разрядном представлении модуля записи мантиссы и *k*-разрядном представлении модуля записи порядка форма с плавающей точкой обеспечивает диапазон изменения абсолютного значения числа *А*, для которого выполняется неравенство:

2ú *х*пúmax × ú*х*мúmax = 2*p* × (1 - 2-*s*) ≥ ú*Х*ú ≥ 0 ,

где *p* = 2*k* - 1.

В ЭВМ числа с плавающей точкой представляются в так называемой нормализованной форме, при которой в прямом коде мантисса нормализованного числа в старшем разряде модуля имеет ненулевое значение, а для двоичной системы счисления – нормализованная мантисса должна иметь в старшем разряде модуля прямого кода значение 1, т.е. для двоичной системы мантисса должна удовлетворять неравенству:

1 > ú*x*мú ≥ 0.5.

Для плавающей точки максимальные значения абсолютной и относительной ошибок определяются следующим образом.

Максимальная абсолютная погрешность представления чисел:

Dmax = 2-(*s*+1) × 2*p*;

Максимальная относительная погрешность:

dmax = D max / *А*min = 2-(*s*+1) × 2*p* / (*х*м min × 2*p*)= 2­(s+1) × 2*p* / ( 2-1 × 2*p*) = 2­(*s*+1) / (2-1) = =2-*s*.

Отсюда видно, что относительная ошибка при представлении чисел в форме с плавающей точкой существенно меньше, чем в случае с фиксированной точкой. Это, а также больший диапазон изменения представляемых чисел, является основным преимуществом представления чисел с плавающей точкой.

Операция сложения

Операция сложения чисел предполагает наличие одинаковых масштабов складываемых величин. Для случая представления чисел с плавающей точкой это предполагает наличие одинаковых порядков у операндов, подлежащих суммированию. Поэтому при выполнении операции сложения чисел с плавающей точкой в общем случае должно быть реализовано три этапа:

- выравнивание порядков;

- сложение мантисс операндов, имеющих одинаковые порядки;

- определение нарушения нормализации и при необходимости её устранение.

Пример

Найти разность С1 чисел А и В, представленных с плавающей точкой, если А и В представлены в виде порядков, соответственно [ап]пк и [вп]пк и мантисс, соответственно [ам]пк и [вм]пк, где [ап]пк = = 1.001, [ам]пк = 1.11001, [вп]пк = 0.001, [в м]пк = 0.11100.

При выполнении операций использовать дополнительный модифицированный код.

Решение

Начнем с выравнивания порядков.

Для этого из порядка первого числа вычитается порядок второго числа:

1.111 – [ап]дк

+ 1.111 – [-вп]дк

1.110 – разность порядков в дополнительном коде,

1.010 – разность порядков в прямом коде.

Так как знак разности порядков отрицательный, то в качестве общего порядка, а следовательно, и предварительного значения порядка искомого результата С1п`, берется порядок второго числа (вп). Для того чтобы взять в качестве порядка первого числа порядок второго числа, т.е. увеличить его порядок на 2, необходимо мантиссу этого меньшего числа умножить на 2-2, т.е. выполнить её арифметический сдвиг на два разряда вправо.

Таким образом, будем иметь после выравнивания следующую форму представления операндов: [ам`]пк = 1.00110, [в м`]пк = 0.11100. После выравнивания порядков можно определить предварительное значение мантиссы С1` как

С1` = [aм`]пк - [вм`]пк.

11.11010 – [ам`]мдк

+ 11.00100 – [-вм`]мдк

~~1~~10.11110 – [С1`]мдк

10.00010 – [С1`]мпк.

Из записи [С1`]дк, полученной после вычитания мантисс операндов с выравненными порядками, видно, что нормализация представления результата нарушена. Поэтому для данного примера необходимо выполнить этап устранения нарушения нормализации.

В данном случае нарушение нормализации слева от точки, так как получено [С1`]пк с ненулевой целой частью (неодинаковые разряды в поле знака использованного модифицированного дополнительного кода). Для того чтобы привести полученную предварительную мантиссу к нормализованной форме, достаточно её разделить на 2, то есть выполнить её арифметический сдвиг вправо. В результате будем иметь окончательное значение мантиссы:

С1 = С1` 2 - 1=10.00010 2 - 1 = 11.10001.

Деление мантиссы С1` на 2 сопровождается изменением ранее найденного предварительного значения порядка результата С1п` на +1.

00.001 – [сп`]мпк

+ 00.001 – +1

00.010 – [с1п]мпк

После устранения нарушения нормализации окончательный результат будет иметь вид

С1 {[c1п]пк = 00.010, [c1м]пк = 11.10001}.

**19. Умножение чисел, представленных в формате с плавающей точкой. Деление чисел, представленных в формате с плавающей точкой.**

С точки зрения представления чисел с плавающей точкой поиск произведения

*С*2 =*А* × *В*

сводится к поиску *С*2п и *С*2м, соответственно порядку и мантиссе произведения на основании порядка *а*п и мантиссы *а*м множимого и порядка *в*п и мантиссы *в*м множителя. Учитывая общую запись чисел с плавающей точкой, произведение двух операндов представляется в виде

*С*2 = *А* × *В* = 2*а*п × *а*м × 2*в*п × *в*м = 2*а* п+вп × *а*м× *в*м = 2*с*

п × *с*2м.

Отсюда вытекает, что порядок произведения определяется как сумма порядков сомножителей, а мантисса произведения – как произведение мантисс сомножителей. Однако, учитывая возможность нарушения нормализации при умножении мантисс, в результате указанных действий будет найдено предварительное значения порядка и мантиссы искомого произведения, и окончательное значение произведения будет найдено только после устранения нарушения нормализации.

Таким образом, имеем:

*С*2п` = *а* п + *в* п;

*С*2м`= *а* м × *в*м.

Отсюда последовательность действий, обеспечивающих получение произведение двух чисел, заключается в следующем:

- определяется знак произведения как сумма по модулю двух знаковых разрядов мантисс сомножителей;

- определяется предварительное значение порядка произведения посредством суммирования порядков сомножителей;

- определяется предварительное значение мантиссы произведения как произведения мантисс операндов;

- устраняется нарушение нормализации мантиссы произведения (если нарушение имеет место) соответствующей корректировкой предварительного значения порядка и мантиссы искомого произведения.

При формировании мантиссы произведения нормализованных чисел с плавающей точкой возможен только один вид нарушения нормализации – нарушение нормализации справа от точки с появлением нуля только в старшем разряде мантиссы.

*Пример*

Найти произведение *С* чисел *А* и *В*, представленных с плавающей точкой, если *А* и *В* представлены в виде порядков, соответственно [*а*п]пк и [*в*п]пк и мантисс, соответственно [*а*м]пк и [*в*м]пк,

где [*а*п]пк = 1.010, [*а*м]пк = 1.1010, [*в*п]пк = 0.001, [*в*м]пк = 0.1001.

При выполнении операций использовать *обратный код*. При умножении мантисс использовать метод умножения, начиная *со старшего разряда* множителя со сдвигом промежуточного результата.

*Решение*

Знак искомого произведения, представляемого знаком его мантиссы, отрицательный, так как знаки мантисс сомножителей неодинаковые.

Предварительное значение порядка произведения определяется следующим образом:

*С*п' = *а* п + *в*п:

11.101 – [*а*п]мок

+ 00.001 – [*в*п]мок

11.110 – [*С*п`]мок

11.001 – [*С* п`]мпк, т.е. [*С*п`]пк = 1.001.

Абсолютное значение предварительного значения мантиссы произведения определяется следующим образом:

[*С*м`]:

0.1010 – œ *а*м œ

× 0.1001 – œ *в*м œ

.0000 – начальное значение промежуточного произведения

+ 1010 – первый младший разряд множителя равен единице

1010 – промежуточное произведение с учетом первого разряда

01010 – сдвинутое промежуточное произведение

001010 – второй разряд множителя равен нулю, поэтому выполняется только сдвиг

0001010 – третий разряд равен нулю, поэтому выполняется только сдвиг

+ 1010 – четвертый разряд равен единице

1011010 – промежуточное произведение с учетом старшего разряда

01011 010 – сдвинутое промежуточное произведение.

Таким образом,

[*С*м`]пк = 0.01011010.

С учетом округления имеем

[*С* м]пк = 0.01011.

Мантисса произведения ненормализованная, поэтому необходимо сдвинуть мантиссу влево на один разряд, а предварительное значение порядка произведения уменьшить на единицу. После нормализации с учетом ранее полученного знака окончательные значения мантиссы и порядка произведения будут следующими:

[*С*м]пк = 1.1011.

[*С*п]пк = 1.010.

**Деление чисел, представленных в формате с плавающей точкой.**

С точки зрения формирования частного представления чисел с плавающей точкой поиск частного С3 =А / В сводится к поиску С3п и С3м, соответственно порядку и мантиссы частного на основании порядка ап и мантиссы ам делимого и порядка вп и мантиссы вм делителя. Учитывая общую запись чисел с плавающей точкой, произведение двух операндов представляется в виде

С3 = А / В = 2aп ам / (2вп вм) = 2ап - вп (ам / вм) = 2сп  см.

Отсюда следует, что порядок частного определяется как разность порядка делимого и делителя, а мантисса – как частное от деления мантиссы делимого на мантиссу делителя. Однако, учитывая то, что при делении мантисс может произойти нарушение нормализации, в результате указанных действий будет найдено предварительное значения порядка и мантиссы искомого частного. Окончательные значения порядка и мантиссы частного будут определены после устранения нарушения нормализации в предварительном результате.

При формировании мантиссы частного нормализованных чисел с плавающей точкой возможен только один вид нарушения нормализации - нарушение нормализации слева от точки.

Пример

Найти частное С3 от деления чисел А на В, представленных с плавающей точкой, если А и В представлены в виде порядков, соответственно

[ап]пк и [вп]пк и мантисс, соответственно [ам]пк и [вм]пк,

где [ап]пк = 1.010, [а м]пк = 1.1010, [вп]пк = 0.001, [вм]пк= 0.1001.

При выполнении операций использовать обратный код. При делении мантисс использовать метод деления без восстановления остатка. При вычитании порядков и формирования мантиссы частного использовать модифицированный обратный код.

Решение

Знак искомого частного, представляемого знаком его мантиссы, отрицательный, так как знаки мантисс сомножителей не одинаковые. Предварительное значение порядка [С3п`]ок частного определяется следующим образом:

С3п` = ап - вп:

11.101 – [ап]мок

+ 11.110 – [вп]мок

111.011

+ 1

11.011 – [С3п`]мок, т.е. [С3п`]пк = 1.011.

Абсолютное значение предварительного значения мантиссы частного ищется за счет выполнения шести тактов деления следующим образом:

00.1010 – [œ амœ]ок ,

+ 11.0110 – [-œ вмœ]ок ,

100.0000

+1 – учет переноса (переполнения знакового поля) при сложении в обратном коде,

00.0001 – положительный остаток первого такта,

00.0010 – сдвинутый остаток,

+ 11.0110 – [-œвмœ]мок,

11.1000 – отрицательный остаток второго такта,

11.0001 – остаток после арифметического сдвига влево,

+ 00.1001 – [œвмœ]мок,

11.1010 – отрицательный остаток третьего такта,

11.0101 – остаток после арифметического сдвига влево,

+ 00.1001 – [œ вмœ]мок,

11.1110 – отрицательный остаток четвертого такта,

11.1101 – остаток после арифметического сдвига влево,

+ 00.1001 – [œ вмœ]мок,

100.0110

+ 1

00.0111 – положительный остаток пятого такта

00.1110 – остаток после арифметического сдвига влево,

+ 11.0110 – [-œ вмœ]мок,

100.0100

+ 1

00.0101 – положительный остаток шестого такта,

00.1010 – остаток после арифметического сдвига влево.

Таким образом, учитывая знаки остатков, полученных на шести тактах, абсолютное предварительное значение мантиссы искомого частного равно:

[œ С3м`œ]пк= 1.00011,

с учетом округления:

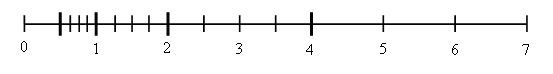
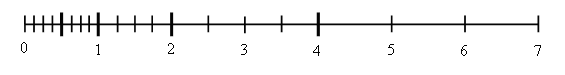
[œ С3м`œ]пк = 1.0010.

Мантисса частного не нормализованная (нарушение нормализации слева от точки), поэтому необходимо сдвинуть мантиссу вправо на один разряд, а предварительное значение порядка частного увеличить на единицу. После нормализации окончательное значение мантиссы и порядка частного равны:

[С3м]пк = 0.1001,

[С3п]пк = 0.000.

# **22. Денормализованные числа. Подводные камни в арифметике с плавающей запятой**

Что такое денормализованные (subnormal) числа рассмотрим на простом примере. Пусть имеем нормализованное представление с длиной мантиссы |M|=2 бита (+ один бит нормализации) и диапазоном значений порядка -1≤E≤2. В этом случае получим 16 чисел:  
  
  
 Крупными штрихами показаны числа с мантиссой, равной 1,00. Видно, что расстояние от нуля до ближайшего числа (0 — 0,5) больше, чем от этого числа к следующему (0,5 — 0,625). Это значит, что разница двух любых чисел от 0,5 до 1 даст 0, даже если эти числа не равны. Что еще хуже, в пропасть между 0,5 и 0 попадает разница чисел, больших 1. Например, «1,5-1,25=0» (см. картинку).  
 В «околонулевую яму» подпадает не каждая программа. Согласно статистике 70-х годов в среднем каждый компьютер сталкивался с такой проблемой один раз в месяц. Учитывая, что компьютеры приобретали массовость, разработчики «K-C-S» посчитали эту проблему достаточно серьезной, чтобы решать ее на аппаратном уровне. Предложенное ими решение состояло в следующем. Мы знаем, что при E=Emin-1 (для float это «-127») и нулевой мантиссе число считается равным нулю. Если же мантисса не нулевая, то число считается не нулевым, его порядок полагается E=Emin, причем неявный старший бит мантиссы полагается равным нулю. Такие числа называются *денормализованными*.  
 Строго говодя, числа с плавающей запятой теперь имеют вид:  
(-1)s × 1.M × 2E, если Emin≤E≤Emax (нормализованные числа)  
(-1)s × 0.M × 2Emin, если E=Emin-1. (денормализованные числа)  
 Вернемся к примеру. Наш Emin=-1. Введем новое значение порядка, E=-2, при котором числа являются денормализованными. В результате получаем новое представление чисел:  
  
 Интервал от 0 до 0,5 заполняют денормализованные числа, что дает возможность не проваливаться в 0 рассмотренных выше примерах (0,5-0,25 и 1,5-1,25). Это сделало представление более устойчиво к ошибкам округления для чисел, близких к нулю.   
 Но роскошь использования денормализованного представления чисел в процессоре не дается бесплатно. Из-за того, что такие числа нужно обрабатывать по-другому во всех арифметических операциях, трудно сделать работу в такой арифметике эффективной. Это накладывает дополнительные сложности при реализации АЛУ в процессоре. И хоть денормализованные числа очень полезны, они не являются панацеей и за округлением до нуля все равно нужно следить. Поэтому эта функциональность стала камнем преткновения при разработке стандарта и встретила самое сильное сопротивление.

**Подводные камни в арифметике с плавающей точкой**

*Округление*

С ошибками из-за погрешностей округления в современной арифметике с плавающей запятой встретиться сложно, особенно если использовать двойную точность. Правило округления в стандарте IEEE754 говорит о том, что результат любой арифметической операции должен быть таким, как если бы он был выполнен над точными значениями и округлен до ближайшего числа, представимого в этом формате. Это требует от АЛУ дополнительных усилий и некоторые опции компилятора (такие как «-ffast-math» в gcc) могут отключить такое поведение. Особенности округления в IEEE754:

* Округление до ближайшего в стандарте сделано не так как мы привыкли. Математически показано, что если 0,5 округлять до 1 (в большую сторону), то существует набор операций, при которых ошибка округления будет возрастать до бесконечности. Поэтому в IEEE754 применяется правило округления до четного. Так, 12,5 будет округлено до 12, а 13,5 – до 14.
* Самая опасная операция с точки зрения округления в арифметике с плавающей запятой — это вычитание. При вычитании близких чисел значимые разряды могут потеряться, что  
  может в разы увеличить относительную погрешность.
* Для многих широко распространенных математических формул математики разработали специальную форму, которая позволяет значительно уменьшить погрешность при округлении. Например, расчет формулы «x2-y2» лучше вычислять используя формулу «(x-y)(x+y)».

*Неассоциативность арифметических операций*

В арифметике с плавающей запятой правило (a\*b)\*c = a\*(b\*c) не выполняется для любых арифметических операций. Например,  
  
(1020+1)-1020=0 ≠ (1020-1020)+1=1

*Числовые константы*

Помните, что не все десятичные числа имеют двоичное представление с плавающей запятой. Например, число «0,2» будет представлено как «0,200000003» в одинарной точности. Соответственно, «0,2 + 0,2 ≈ 0,4». Абсолютная погрешность в отдельном случае может и не высока, но если использовать такую константу в цикле, можем получить накопленную погрешность.

*Выбор минимума из 2 значений*

Допустим из двух значений нам нужно выбрать минимальное. В Си это можно сделать одним из следующих способов:

1. x < y? x: y
2. x <= y? x: y
3. x > y? y: x
4. x >= y? y: x

Часто компилятор считает их эквивалентными и всегда использует первый вариант, так как он выполняется за одну инструкцию процессора. Но если мы учтем ±0 и NaN, эти операции никак не эквивалентны:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | y | x < y? x: y | x <= y? x: y | x > y? y: x | x >= y? y: x |
| +0 | -0 | -0 | +0 | +0 | -0 |
| NaN | 1 | 1 | 1 | NaN | NaN |

*Сравнение чисел*

Очень распространенная ошибка при работе с float-ами возникает при проверке на равенство. Например,

float fValue = 0.2;

if (fValue == 0.2) DoStuff();

Ошибка здесь, во-первых, в том, что 0,2 не имеет точного двоичного представления, а во-вторых 0,2 – это константа двойной точности, а переменная fValue – одинарной, и никакой гарантии о поведении этого сравнения нет.   
 Лучший, но все равно ошибочный способ, это сравнивать разницу с допустимой абсолютной погрешностью.

Недостаток такого подхода в том, что погрешность представления числа увеличивается с ростом самого этого числа. Так, если программа ожидает «10000», то приведенное равенство не будет выполняться для ближайшего соседнего числа (10000,000977). Это особенно актуально, если в программе имеется преобразование из одинарной точности в двойную.

Выбрать правильную процедуру сравнения сложно и заинтересованных читателей я отсылаю к статье [Брюса Доусона](http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm). В ней предлагается сравнивать числа с плавающей запятой преобразованием к целочисленной переменной. Это — лучший, хотя и не портабельный способ.

# **24.Основные понятия алгебры логики. Способы задания логической функции.**

Алгебра логики используется при анализе и синтезе схем ЭВМ **по двум причинам.** **Во-первых**, это объясняется соответствием представления переменных и функций алгебры логики. **Во-вторых**, двоичным представлением информации и характером работы отдельных компонентов вычислительной техники. Эти компоненты могут пропускать или не пропускать ток, иметь на выходе высокий или низкий уровень сигнала (напряжения или тока).

Приведем основные понятия алгебры логики.

**Логическая переменная** — это такая переменная, которая может принимать одно из двух значений: истинно или ложно (да или нет, единица или ноль).

**Логическая константа** — это такая постоянная величина, значением которой может быть истинно или ложно (да или нет, единица или ноль).

**Логическая функция** — это такая функция, которая может принимать одно из двух значений: истинно или ложно (да или нет, единица или ноль) в зависимости от текущих значений ее аргументов, в качестве которых используются логические переменные.

Логическая функция может быть одного **(n = 1)** или нескольких **(n > 2)** аргументов. Значение логической функции определяется комбинацией конкретных значений переменных, от которых она зависит. Комбинация конкретных значений переменных (аргументов функции) называется набором. Количество различных наборов N для «n» переменных вычисляется по формуле **N = 2^n.**

Зависимость логической функции от переменных может задаваться по–разному:

– словесным описанием;

– таблицей истинности;

– логическим выражением.

Словесное описание используется в случае сравнительно несложной логической функции.

Таблица истинности является универсальным средством задания логической функции. Она включает все наборы для заданного количества переменных, определяющих значение логической функции, с указанием значений, которые принимает функция для каждого набора. В одной таблице истинности может задаваться несколько логических функций, зависящих от одних и тех же переменных. Таблица истинности для нескольких функций y трёх переменных **х1, х2, х3** может быть задана следующим образом (табл. 2.1)

Таблица 2.1

**Таблица истинности трех переменных**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | х1 | х2 | х3 | y1 | y2 | y3 | … | yn |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 |  | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 |  | 1 |
| 2 | 0 | 1 | 0 | 1 | 1 | 1 |  | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 | 0 |  | – |
| 4 | 1 | 0 | 0 | 1 | 0 | 1 |  | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 | 0 |  | 1 |
| 6 | 1 | 1 | 0 | 0 | 0 | 1 |  | – |
| 7 | 1 | 1 | 1 | 1 | 1 | 0 |  | 1 |

В приведенной таблице истинности во второй, третей и четвертой колонках, помеченных соответственно х1, х2, х3 , приведены все возможные наборы этих переменных. В следующих колонках приводятся значения функций y1, y2, yn для каждого набора.

Логическая функция называется «полностью определенной», если для нее заданы значения по всем возможным наборам. Функция называется «частично определенной», если для некоторых наборов значения функции не заданы. В приведенной таблице истинности функции y1, y2, y3 являются полностью определенными, а функция **yn – частично определенная** (знак «–» означает неопределенность значения функции).

**Максимальное количество полностью определенных функций** от «n» переменных определяется как **M = (2^2)^n**

**Логическим выражением** называется комбинация логических переменных и констант, связанных элементарными базовыми логическими функциями (или логическими операциями), которые могут разделяться скобками.

**Например**, логическую функцию у1, определенную в вышеприведенной таблице истинности, можно представить в виде логического выражения

**Набор элементарных логических операций**, с помощью которых можно задать любую, сколь угодно сложную логическую функцию, **называется функционально полной системой логических функций**. Иногда такую систему называют базисом.

В качестве элементарных логических функций функционально полных систем этих функций используются функции одной или двух логических переменных.

Все возможные функции одной переменной приведены в табл. 2.2.

Таблица 2.2

**Функции одной переменной**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | y0 | y1 | y2 | y3 |
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |

Из таблицы видно, что:

y0 = 0 – константа; y1 равна значению переменной; y2 равна значению, обратному значению переменной « х »; y3 = 1 – константа.

С точки зрения базовых функций интерес представляет только функция y2, она называется функцией отрицания, читается как «не х» и обозначается как «», т. е. можно записать y2 =.

Все возможные функции двух переменных приведены в табл. 2.3

Таблица 2.3

**Функции двух переменных**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | х1 | х2 | y0 | y1 | y2 | y3 | y4 | y5 | y6 | y7 | y8 | y9 | y10 | y11 | y12 | y13 | y14 | y15 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 2 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 3 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

**25.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



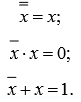
Если в качестве А используется простая конъюнкция, т. е. конъюнкция, представляющая собой логическое произведение переменных и их отрицаний, то имеет место



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:

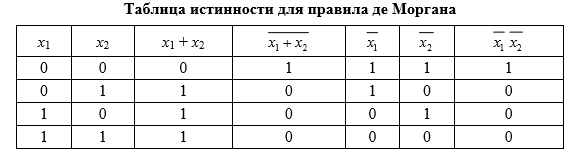
(х1 + х2)(х1 + х3) = х1х2 + х1х1 + х1х3 + х2х3 = = х1х2 + х1 + х1х3 + х2х3 = х1(х2 + 1 + х3) + х2х3.

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 произве

дению 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».

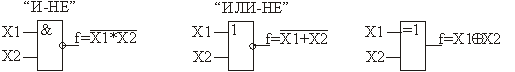
# **27. Полная система логических функций.**

Функционально полная система логических функций представляет собой набор логических функций, с помощью которых можно записать любую, сколь угодно сложную функцию. В этом случае говорят, что этот набор образует базис. Функционально полными являются 3 базиса:

1) "И-ИЛИ-НЕ" (базис конъюнкции, дизъюнкции, инверсии)

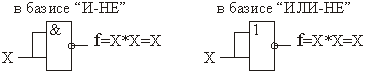
2) "И-НЕ" (базис Шеффера)

3) "ИЛИ-НЕ" (базис Пирса или функция Вебба).

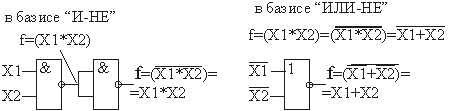


Элементы, реализующие операцию "И-НЕ", “ИЛИ-НЕ” и “Исключающее ИЛИ” на принципиальных и структурных схемах изображаются так:

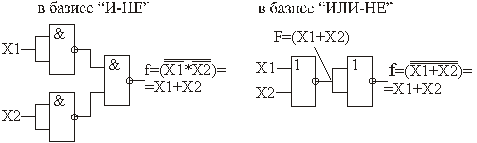
Примеры реализации логических операций в базисах “И-НЕ” и “ИЛИ-НЕ”.



**Реализация операции “НЕ”:**



**Реализация операции “И”:**



**Реализация операции “ИЛИ”:**

**Пример** реализации комбинационного устройства в базисе "И-НЕ". Пусть задана функция, реализуемая комбинационным устройством, в аналитической форме

https://ok-t.ru/helpiksorg/baza4/1997974660.files/image062.png.

Используя закон де Моргана и с учетом закона двойного инвертирования, запишем эту функцию в виде

https://ok-t.ru/helpiksorg/baza4/1997974660.files/image063.png.

Как следует из полученного аналитического выражения, логическое устройство должно содержать три двухвходовых и один трехвходовой элемент И-НЕ. Функциональная схема комбинационного устройства, построенная в базисе И-НЕ, показана на рис. 1.10.



**39. Нормальная и совершенные нормальные логических функций**

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

* Совершенная конъюнктивная (лог.оп. и) нормальная форма (СКНФ)
* Совершенная дизъюнктивная (лог.оп. или) нормальная форма (СДНФ)

Совершенная дизъюнктивная нормальная форма представляет собой дизъюнкцию (дизъюнкция - логическое ИЛИ - даёт значение 1 когда хотя бы один из элементов равен 1, иначе 0) простых конъюнкций (конъюнкция - логическое И - результат операции равен 1, если все операнды равны 1, в остальных случаях результат равен 0), где под термином простая конъюнкция имеется в виду конъюнкция переменных или их отрицаний. В СДНФ простые конъюнкции содержат все переменные в своей прямой или инверсной форме и отражают собой наборы, на которых представляемая функция имеет единичное значение. Такие конъюнкции называются конституентами единицы рассматриваемой функции. Поэтому СДНФ представляет собой дизъюнкцию (логическую сумму), слагаемыми которой являются конституенты единицы. Общая запись СДНФ функции «y» имеет вид

y1 = ;

y2 = ;

y3 = .

Совершенная конъюнктивная нормальная форма - это конъюнкция простых дизъюнкций, где под термином простая дизъюнкция имеется в виду дизъюнкция переменных или их отрицаний. В СКНФ простые дизъюнкции содержат все переменные в своей прямой или инверсной форме и представляют собой отрицание конституент нуля. Общая запись СКНФ функции «y» имеет вид:

y1 = ;

y2 = ;

y3 = .

СКНФ строится на основе конституентa нуля. Конституент нуля представляет набор логических переменных, на котором логическая функция принимает значение «0». Каждая скобка в приведенных выражениях представляет собой отрицание конституенты нуля соответствующей функции, а запись функции в виде конъюнкция таких скобок представляет собой условие, при котором отсутствуют все конституенты нуля определяемой функции, при выполнении которого функция имеет единичное значение.

Например, для y1 выражение первой скобки представляет собой отрицание набора значений переменных второй строки, на котором функция y1 имеет нулевое значение, выражение второй скобки представляет собой отрицание набора значений переменных четвертой строки, на котором функция y1 также имеет нулевое значение, выражение третьей скобки представляет собой отрицание набора значений переменных пятой строки, на котором функция y1 имеет нулевое значение.

Из вышеизложенного следует, что любую функцию можно представить или в СДНФ, или в СКНФ, а так как эти формы представлены в базисе Буля, то отсюда значит, что этот базис (базис И, ИЛИ, НЕ) является функционально полным.

Если функция задана в СДНФ и требуется найти ее СКНФ, то такой переход можно выполнить, составив по заданной СДНФ таблицу истинности для этой функции, а на основе полученной таблицы составить СКНФ заданной функции.

Однако в некоторых случаях может оказаться более удобным подход, который поясняется следующим примером.

По заданной СДНФ функции y3 = 

Найти запись этой функции в СКНФ.

*Решение:*

Запишем логические выражение отрицания заданной функции, т.е. найдем логическое условие, при котором эта функция имеет нулевое значение. В качестве такого выражения можно взять дизъюнкцию конъюнкций, где каждая конъюнкция представляет собой конституенту нуля заданной функции. Очевидно, что конституенты нуля это те наборы, которые не являются наборами, соответствующими конституентам единицы, которые использованы в СДНФ. Таким образом, можно записать:



Эту запись можно интерпретировать как словесное описание функции: функция у равна нулю, если имеет место хотя бы одна из конституент нуля.

В этой записи представлена дизъюнкция тех наборов, которые не использовались в записи функции y3. Возьмем отрицание правой и левой частей полученного уравнения и применим к правой части правило де Моргана.



Применим правило Де Моргана к отрицаниям конъюнкций, полученным в правой части:

.

P.S. Правила Де Моргана:

Отрицание суммы равно произведению отрицаний: 

Отрицание произведения равно сумме отрицаний: 

# **карн29. Минимизация булевых функций. Основные понятия. Наиболее известные методы минимизации. Минимизация системы логических функций. Минимизация частично определенных функций**

Учитывая то, что одну и ту же логическую функцию можно представить различными выражениями, перед реализацией функции в виде логической схемой весьма важным является выбор из всех возможных выражений, соответствующих данной функции, самого простого. Решить эту проблему можно за счет использования процедуры минимизации логического выражения.

**Методы минимизации:**

- *минимизация* методом Квайна;

- *минимизация* с использованием диаграмм Вейча (или карт Карно);

- *минимизация* не полностью определенных ( частично определенных) функций;

- *минимизация* конъюнктивных нормальных форм;

- *минимизация* методом кубического задания функций алгебры логики;

- *минимизация* методом Квайна–Мак-Класски;

- *минимизация* с использованием алгоритма извлечения (Рота);

- *минимизация* ФАЛ методом преобразования логических выражений.

" При проэктировании цифровых автоматов широко используются методы минимизации булевых функций, позволяющие получать рекомендации для построения экономичных схем цифровых автоматов. Общая задача минммизации булевых функций может быть сформулирована следующим образом: найти аналитическое выражение заданой булевой функции в форме, содержащей минимально возможное число букв. Следует отметить, что в общей постановке данная задача пока не решена, однако достаточно хорошо исследована в классе дизъюнктивно - конъюнктивных форм.

**Определение.  
Элементарной конъюнкцией называется конъюнкция конечного числа различных между собой булевых переменных, каждая из которых может иметь или не иметь отрицания.**

**Определение.  
Дизъюнктивной нормальной формой (ДНФ) называется дизъюнкция элементарных конъюкций.**

**Определение.  
Минимальной дизъюнктивной нормальной формой булевой функции называется ДНФ, содержащая наименьшее число букв (по отношению ко всем другим ДНФ, представляющим заданную булеву функцию).**

**Определение.  
Булева функция g(x1,...,xn) называется импликантой булевой функции f(x1,...,xn), если для любого набора переменных, на котором g=1, справедливо f=1.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Таблица 4.1** |  |  |  |  |  |  |  |  |
| x3x2x1 | f | g1 | g2 | g3 | g4 | g5 | g6 | g7 |
| 000 001 010 011 100 101 110 111 | 0 0 0 1 0 0 1 1 | 0 0 0 0 0 0 0 1 | 0 0 0 0 0 0 1 0 | 0 0 0 0 0 0 1 1 | 0 0 0 1 0 0 0 0 | 0 0 0 1 0 0 0 1 | 0 0 0 1 0 0 1 0 | 0 0 0 1 0 0 1 1 |

f = /x1x2x3 v x1x2/x3 v x1x2x3 = g7;  
g1 = x1x2x3;  
g2 = x1x2/x3;  
g3 = x1x2x3 v x1x2/x3 = x1x2 (x3 v x3) = x1x2;  
g4 = /x1x2x3;  
g5 = /x1x2x3 v x1x2x3 = x2x3;  
g6 = /x1x2x3 v x1x2/x3;

**Определение.  
Импликанта g булевой функции f, являющаяся элементарной конъюнкцией, называется простой, если никакая часть импликанты g не является импликантой функции f.**

Из примера видно, что импликанты g3 = x1x2 и g5 = x2x3 являются простыми импликантами функции f. Импликанты g1, g2, g4, g6 не являются простыми, так как их части являются импликантами функции f, например g3является частью g1. Приведем без доказательства два утверждения, полезные при получении минимальной ДНФ.

1. Дизъюнкция любого числа импликант булевой функции f также является импликантой этой функции.
2. Любая булева функция f эквивалентна дизъюнкции всех своих простых импликант. Такая форма представления булевой функции называется сокращенной ДНФ.

Перебор всех возможных импликант для булевой функции f из рассмотренного примера дает возможность убедиться, что простых импликант всего две: g3 и g5. Следовательно, сокращенная ДНФ функции f имеет вид

f = g3 v g5 = x1x2 v x2x3.

Как видно из табл. 4.1, импликанты g3, g5 в совокупности покрывают своими единицами все единицы функции f. Получение сокращенных ДНФ является первым этапом отыскания минимальных форм булевых функций. Как уже отмечалось, в сокращенную ДНФ входят все простые импликанты булевой функции. Иногда из сокращенной ДНФ можно убрать одну или несколько простых импликант, не нарушая эквивалентности исходной функции. Такие простые импликанты назовем лишними. Исключение лишних простых импликант из сокращенных ДНФ - второй этап минимизации.

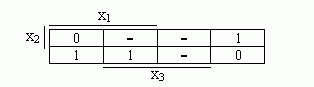
**Определение.  
Сокращенная ДНФ булевой функции называется тупиковой, если в ней отсутствуют лишние простые импликанты.**

Устранение лишних простых импликант из сокращенной ДНФ булевой функции не является однозначным процессом, т. е. булева функция может иметь несколько тупиковых ДНФ.

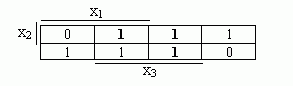
**Утверждение.  
Тупиковые ДНФ булевой функции f, содержащие минимальное число букв, являются минимальными. Минимальных ДНФ тоже может быть несколько.**

#### Минимизация частично определенных булевых функций.

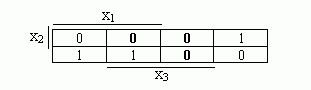
" В реальных задачах очень часто бывает так, что значение булевой функции на некоторых наборах не определено и может доопределяться произвольно. В этом случае доопределение функции было бы целесообразно производить таким образом, чтобы ее минимальная нормальная форма имела наименьшее число букв из всех возможных вариантов доопределения. Рассмотрим простой пример. Функция задана диаграммой Вейча, представленной (табл. 4.7.1(a)).



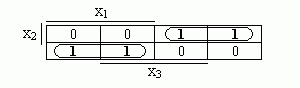
Доопределение функции на неопределенных наборах единицами (табл 4.7.1(b))



или нулями (табл 4.7.1(c))



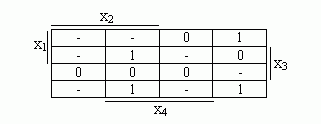
приводит к разным минимальным ДНФ Однако более простая минимальная ДНФ получается, если произвести доопределение так, как это сделано на диаграмме Вейча (табл. 4.7.2)



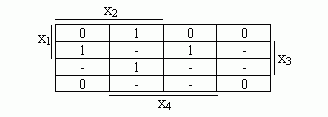
Алгоритм поиска минимальной ДНФ частично определенной функции f можно представить следующим образом

* Найти любым известным способом сокращенную ДНФ функции, получающейся доопределением единицами исходной функции f на всех неопределенных наборах.
* Выбрать минимальную ДНФ по импликантной матрице, где в столбцах выписаны лишь те конституенты единицы функции f, которые соответствуют полностью определенным единичным наборам.

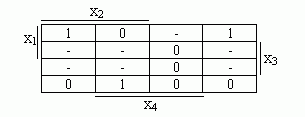
Аналогичный алгоритм (с доопределением нулевыми наборами) может быть предложен для поиска КНФ При этом доопределение таблицы истинности функции f может быть произведено по разному для КНФ и ДНФ.  
Заметим, что для решения рассматриваемой задачи практически достаточно тех навыков, которые были получены при минимизации полностью определенных булевых функций непосредственно по диаграмме Вейча. Приведем несколько примеров. В случаях, когда минимальных форм несколько, приводится одна из них.  
Для функции, представленной табл. 4.7.3:



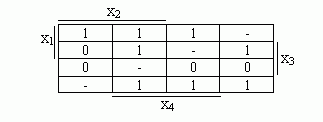
ДНФ: x1x2 v /x1/x3 v /x3/x4;  
КНФ: (x1 v /x3)(/x3 v x4)(x2 v /x4).  
Для функции, представленной табл. 4.7.4



ДНФ: x3 v x2x4;  
КНФ: (x2 v x3)(x3 v x4).  
Для функции, представленной табл. 4.7.5:



ДНФ: x1/x4 v /x1x2x4;  
КНФ: (/x1 v /x4)(x1 v x2)(x1 v x4).  
Для функции, представленной табл. 4.7.6:



ДНФ: x3 v x1x4 v x1/x2;  
КНФ: (x1 v /x3)(/x2 v /x3 v x4)."

#### Минимизация систем булевых функций.

"На практике очень часто приходится реализовывать совокупности булевых функций. Если произвести минимизацию булевых функций, входящих в систему, независимо друг от друга, то общая схема будет состоять из изолированных подсхем. Ее можно иногда упростить за счет объединения участков подсхем, реализующих одинаковые члены, входящие в несколько булевых функций системы. Задача минимизации систем булевых функций хорошо исследова на в классе функционально полных систем: "дизъюнкция", "конъюнк ция", "отрицание". Рассмотрим один из наиболее распространенных методов минимизации. Пусть задана система полностью определенных булевых функций, представленных в дизъюнктивной нормальной форме, например:

f1(x1,x2,x3) = x1/x3 v x1/x2 v /x1x3;  
f2(x1,x2,x3) = x1/x2 v /x1x3 v /x1x2;  
f3(x1,x2,x3) = x1x2 v /x1/x2/x3,

Все различные элементарные конъюнкции системы функций объединим в множество А, которое назовем полным множеством элементарных конъюнкций cистемы функций. В нашем случае А = {x1/x3; x1/x2; /x1x3; /x1x2; x1x2; /x1/x2/x3}. Сумма рангов (число букв) элементарных конъюнкций множества А является удобным критерием оценки сложности заданной системы булевых функций.

**Определение.  
Система дизъюнктивных нормальных форм булевых функций называется минимальной, если ее полное множество элементарных конъюнкций содержит минимальное количество букв, а каждая дизъюнктивная нормальная форма булевой функции системы включает минимальное число элементарных конъюнкций наименьшего ранга. При этом дизъюнктивная нормальная форма представления булевой функции в минимальной системе в общем случае не совпадает с ее минимальной дизъюнктивной нормальной формой.**

Минимизация систем полностью определенных булевых функций может производиться по алгоритму, аналогичному алгоритму метода Квайна с небольшими отличиями. Алгоритм минимизации следующий.

* Построить полное множество А элементарных конъюнкций минимизируемой системы функций, считая, что вначале каждая из функций системы представлена в СДНФ. Каждой конституенте единицы множества А присвоить признак, содержащий номера функций системы, в которые входит рассматриваемая конституента.
* Произвести минимизацию СДНФ функции f, конституентами единицы которой являются все элементы множества A. При выполнении склеивания двух конституент единицы каждой вновь образуемой элементарной конъюнкции присвоить признак, состоящий из номеров функций, общих для двух склеиваемых конституент единицы (см. примеры). Последнее справедливо и для двух склеиваемых элементарных конъюнкций с признаками. Если признаки склеиваемых конституент единицы не содержат общих номеров, то склеивание не производится. Поглощение производится только для элементарных конъюнкций с одинаковыми признаками. Полученные в результате склеивания и поглощения конъюнкции называются простыми импликантами системы функций.
* Построить импликантную матрицу функции f, аналогичную мат- риае Квайна с той разницей, что для каждой конституенты единицы выделяется столько столбцов, сколько различных номеров функций со- держит ее признак. Покрытие матрицы импликантами производится аналогично методу Квайна.

Пусть система булевых функций задана таблицей истинности (табл. 5.1). Найдем минимальную ДНФ системы булевых функций. 

|  |  |  |
| --- | --- | --- |
| **Таблица 5.1** |  |  |
| x1x2x3 | f1 | f2 |
| 000 001 010 011 100 101 110 111 | 1 0 0 0 0 1 1 1 | 1 0 1 1 0 1 0 0 |

Представим каждую из функций системы в СДНФ:

f1 = /x1/x2/x3 v x1/x2x3 v x1x2/x3 v x1x2x3;  
f2 = /x1/x2/x3 v /x1x2/x3 v /x1x2x3 v x1/x2x3.

1.   Построим полное множество А элемейтарных конъюнкций полученной системы, приписывая каждой конституенте единицы признак вхождения в функции f1 и f2:

A = {/x1/x2/x3(1,2); /x1x2/x3(2); /x1x2x3(2); x1/x2x3(1,2); x1x2/x3(1); x1x2x3(1)}.

2.   Построим СДНФ функции ф:

ф = /x1/x2/x3(1,2) v /x1x2/x3(2) v /x1x2x3(2) v x1/x2x3(1,2) v x1x2/x3(1) v x1x2x3(1).

Для удобства выполнения склеивания пронумеруем каждую конституенту единицы из СДНФ функции ф и выполним все склеивания:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ф = | /x1/x2/x3(1,2) v | /x1x2/x3(2) v | /x1x2x3(2) v | x1/x2x3(1,2) v | x1x2/x3(1) v | x1x2x3(1); |
|  | 1 | 2 | 3 | 4 | 5 | 6 |

1 - 2: /x1/x3(2) v /x1x2x/3(2) v /x1/x2/x3(1,2);  
2 - 3: /x1x2(2) v /x1x2x3(2) v /x1x2x3(2);  
4 - 6: x1x3(1) v x1/x2x3(1,2) v x1x2x3(1);  
5 - 6: x1x2(1) v x1x2/x3(1) v x1x2x3(1);  
После проведения всех поглощений, с учетом признака каждой конъюнкции, получим:

ф = /x1/x3(2) v x1x3(1) v /x1x2(2) v x1x2(1) v x1/x2x3(1,2) v /x1/x2/x3(1,2);

Дальнейшие склеивания и поглощения невозможны. Получены простые импликанты минимизируемой системы булевых функций.  
3.   Строим импликантную матрицу (табл. 5.2). Столбцы матрицы помечаем конституентами единицы из СДНФ функции ф. Для каждой конституенты единицы отводим столько столбцов матрицы, сколько различных номеров функций содержит признак конституенты. Строки матрицы помечаем простыми импликантами системы булевых функций. Заполнение матрицы аналогично методу Квайна. Ядром функции ф, очевидно, являются простые импликанты /x1/x2/x3(1,2); /x1x2(2); x1/x2x3(1,2); x1x2(1), где для соответствующих конституент единицы функции ф имеется единственная отметка на пересечении столбца и строки импли-кантной матрицы. Выделенное ядро покрывает все конституенты единицы из СДНФ функции ф. 

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Таблица 5.2** |  | | | | | | | |
| Простые импликанты системы функций | Конституенты единици функции Ф | | | | | | | |
| /x1/x2/x3 | | /x1x2/x3 | /x1x2x3 | x1/x2x3 | | x1x2x3 | x1x2/x3 |
| 1 | 2 | 2 | 2 | 1 | 2 | 1 | 1 |
| /x1/x3 (2) |  | **X** | **X** |  |  |  |  |  |
| x1x3 (1) |  |  |  |  | **X** |  | **X** |  |
| /x1x2 (2) |  |  | **X** | **X** |  |  |  |  |
| x1x2 (1) |  |  |  |  |  |  | **X** | **X** |
| x1/x2x3 (1,2) |  |  |  |  | **X** | **X** |  |  |
| /x1/x2/x3 (1,2) | **X** | **X** |  |  |  |  |  |  |

В соответствии с этим имеем:

ф = /x1/x2/x3(1,2) v x1/x2x3(1,2) v /x1x2(2) v x1x2(1).

Выделив для функции fi импликанты с признаком, включающим *i*, получим следующую минимальную дизъюнктивную нормальную форму системы функций:

f1 = /x1/x2/x3 v x1/x2x3 v x1x2;  
f2 = /x1/x2/x3 v x1/x2x3 v /x1x2.

К недостаткам изложенного метода следует отнести большую трудоемкость проведения операций склеивания и поглощения с признаками."

# **30. Минимизация логических выражений методом Квайна**

"Метод Квайна основывается на применении двух основных соотношений.  
Соотношение склеивания Ах V А/х = Ах V А/х V А,

где А - любое элементарное произведение.

1. Соотношение поглощения А~х V А = А,   ~х E {х; /x}.

Справедливость обоих соотношений легко проверяется. Суть метода заключается в последовательном выполнении всех возможных склеиваний и затем всех поглощений, что приводит к сокращенной ДНФ. Метод применим к совершенной ДНФ. Из соотношения поглощения следует, что произвольное элементарное произведение поглощается любой его частью.  
Для доказательства достаточно показать, что произвольная простая импликанта р = xi1xi2 ... xin может быть получена. В самом деле, применяя к р операцию развертывания (обратную операции склеивания):

A = A (x v /x) = Ax v A/x

по всем недостающим переменным xi^(k+l), ..., Xi^n исходной функции f, получаем совокупность S конституент единицы. При склеивании всех конституент из S получим импликанту р. Последнее очевидно, поскольку операция склеивания обратна операции развертывания. Множество S конституент обязательно присутствует в совершенной ДНФ функции f поскольку р - ее импликанта.

|  |  |
| --- | --- |
| **Таблица 4.1.1** |  |
| x4x3x2x1 | f |
| 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111 | 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 1 |

**Пример.** Пусть имеется булева функция, заданная таблицей истинности (табл. 9.1.1). Ее СДНФ имеет вид

f = /x1/x2/x3x4 v /x1/x2x3x4 v /x1x2/x3x4 v /x1x2x3x4 v x1x2x3/x4 v x1x2x3x4.

Для удобства изложения пометим каждую конституенту единицы из СДНФ функции f каким-либо десятичным номером (произвольно). Выполняем склеивания. Конституента 1 склеивается только с конституентой 2 (по переменной x3) и с конституентой 3 (по переменной х2), конституента 2 с конституентой 4 и т. д. В результате получаем

1 - 2: /x1/x2x4;  
1 - 3: /x1/x3x4;  
2 - 4: /x1x3x4;  
3 - 4: /x1x2x4;  
4 - 6: x2x3x4;  
5 - 6: x1x2x3.  
Заметим, что результатом склеивания является всегда элементарное произведение, представляющее собой общую часть склеиваемых конституент Далее производим склеивания получаемых элементарных произведений.  
Склеиваются только те произведения, которые содержат одинаковые переменные. Имеет место два случая склеивания:

/x1/x2x4 v /x1x2x4 = /x1/x2x4 v /x1x2x4 v /x1x4;  
/x1/x3x4 v /x1x3x4 = /x1/x3x4 v /x1x3x4 v /x1x4,

с появлением одного и того же элементарного произведения /x1x4. Дальнейшие склеивания невозможны. Произведя поглощения (из полученной ДНФ вычеркиваем все поглощаемые элементарные произведения), получим сокращенную ДНФ:

x2x3x4 v x1x2x3 v /x1x4.

Переходим ко второму этапу. Для получения минимальной ДНФ необходимо убрать из сокращенной ДНФ все лишние простые импликанты. Это делается с помощью специальной импликантной матрицы Квайна. Строки такой матрицы отмечаются простыми импликантами булевой функции, т. е. членами сокращенной ДНФ, а столбцы - конституентами единицы, т. е. членами СДНФ булевой функции.

**Пример**(продолжение). Импликантная матрица имеет вид (табл. 4.1.2).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Таблица 4.1.2** |  | | | | | |
| Простые импликанты | Конституенты единицы | | | | | |
| /x1/x2/x3x4 | /x1/x2x3x4 | /x1x2/x3x4 | /x1x2x3x4 | x1x2x3/x4 | x1x2x3x4 |
| /x1x4 | **X** | **X** | **X** | **X** |  |  |
| x2x3x4 |  |  |  | **X** |  | **X** |
| x1x2x3 |  |  |  |  | **Х** | **Х** |

Как уже отмечалось, простая импликанта поглощает некоторую конституенту единицы, если является ее собственной частью. Соответствующая клетка импликантной матрицы на пересечении строки (с рассматриваемой простой импликантой) и столбца (с конституентой единицы) отмечается крестиком (табл. 9.1.2). Минимальные ДНФ строятся по импликантной матрице следующим образом:

1. ищутся столбцы импликантной матрицы, имеющие только один крестик. Соответствующие этим крестикам простые импликанты называются базисными и составляют так называемое ядро булевой функции. Ядро обязательно входит в минимальную ДНФ.
2. рассматриваются различные варианты выбора совокупности простых импликант, которые накроют крестиками остальные столбцы импликантной матрицы, и выбираются варианты с минимальным суммарным числом букв в такой совокупности импликант.

**Пример** (продолжение). Ядром нашей функции являются импликанты x1x2x3; /x1x4. Импликанта x2x3x4 - лишняя, так как ядро накрывает все столбцы импликантной матрицы. Поэтому функция имеет единственную тупиковую и минимальную ДНФ:

f = x1x2x3 v /x1x4

Следует отметить, что число N крестиков в одной строке всегда является степенью 2. Более того, читатель может легко убедиться в том, что

N = 2n-k  
где k - число букв, содержащихся в простой импликанте.  
Заметим также, что используя различные соотношения, можно расширить область применения метода Квайна за пределы совершенной ДНФ."  
Применение этих операций продемонстрировано в таблице:

|  |  |  |  |
| --- | --- | --- | --- |
| x1 | x2 | x3 | **f** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

[СДНФ](https://ru.wikipedia.org/wiki/%D0%A1%D0%94%D0%9D%D0%A4) выглядит так:

Результат операции склеивания нужен для преобразования функции на втором этапе (поглощения)





Членами результата склеивания являются



Член  поглощает те члены исходного выражения, которые содержат , то есть первый и четвёртый. Эти члены вычёркиваются. Член поглощает второй и третий, а член  — пятый член исходного выражения.

Повторение обеих операций приводит к следующему выражению:

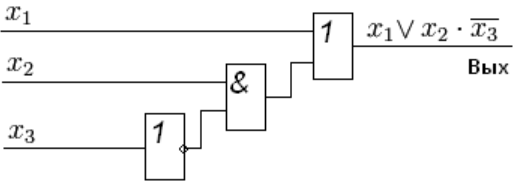


Здесь склеивается пара членов  и  (склеивание пары членов  и  приводит к тому же результату), результат склеивания  поглощает 2-, 3-, 4-, 5-й члены выражения. Дальнейшее проведение операций склеивания и поглощения оказывается невозможным, сокращённая форма выражения заданной функции (в данном случае она совпадает с минимальной формой)



Структурная схема функции

Члены сокращённой формы (в нашем случае это  и   называются *простыми импликантами* функции. В итоге, мы получили наиболее простое выражение, если сравнивать его с начальной версией — [СДНФ](https://ru.wikipedia.org/wiki/%D0%A1%D0%94%D0%9D%D0%A4). Структурная схема такого элемента показана на рисунке справа.



Структурная схема функции

## **Второй этап (табличный) (получение минимальной формы)**

Как и на первом этапе, в полученном равенстве могут содержаться члены, устранение которых никаким образом не повлияет на конечный результат. Следующий этап минимизации — удаление таких переменных. Таблица, представленная ниже, содержит значения истинности функции. По ней будет собрана следующая [СДНФ](https://ru.wikipedia.org/wiki/%D0%A1%D0%94%D0%9D%D0%A4).

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **x1** | **x2** | **x3** | **x4** | **f** |  |  | **x1** | **x2** | **x3** | **x4** | **f** |
| 0 | 0 | 0 | 0 | 1 |  |  | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |  |  | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |  |  | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |  |  | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |  |  | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |  |  | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |  |  | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |  |  | 1 | 1 | 1 | 1 | 1 |

[СДНФ](https://ru.wikipedia.org/wiki/%D0%A1%D0%94%D0%9D%D0%A4), собранная по этой таблице выглядит следующим образом:





Конечное выражение достигается за счёт повторного использования операций склеивания и поглощения:



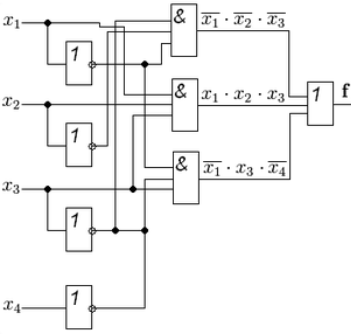
Члены этого выражения являются **простыми импликантами** выражения. Переход от сокращённой формы к минимальной осуществляется с помощью импликантной [матрицы](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)).

### *Импликантная матрица*

Члены [СДНФ](https://ru.wikipedia.org/wiki/%D0%A1%D0%94%D0%9D%D0%A4) заданной функции вписываются в столбцы, а в строки — простые импликанты, то есть члены сокращённой формы. Отмечаются столбцы членов [СДНФ](https://ru.wikipedia.org/wiki/%D0%A1%D0%94%D0%9D%D0%A4), которые поглощаются отдельными простыми импликантами. В следующей таблице простая импликанта   поглощает члены  (в первом и во втором столбцах поставлены крестики).



Вторая импликанта поглощает первый и третий члены [СДНФ](https://ru.wikipedia.org/wiki/%D0%A1%D0%94%D0%9D%D0%A4) (указано крестиками) и т. д. Импликанты, не подлежащие исключению, образуют *ядро*. Такие импликанты определяются по вышеуказанной матрице. Для каждой из них имеется хотя бы один столбец, перекрываемый только этой импликантой.



В нашем примере ядро составляют импликанты  и  (ими перекрываются второй и шестой столбцы). Исключение из сокращённой формы одновременно всех импликант, не входящих в ядро, невозможно, так как исключение одной из импликант может превратить другую в уже нелишний член.   
Для получения минимальной формы достаточно выбрать из импликантов, не входящих в ядро, такое минимальное их число с минимальным количеством букв в каждом из этих импликант, которое обеспечит перекрытие всех столбцов, не перекрытых членами ядра. В рассматриваемом примере необходимо импликантами, не входящими в ядро, перекрыть третий и четвёртый столбцы матрицы. Это может быть достигнуто различными способами, но так как необходимо выбирать минимальное число импликант, то, очевидно, для перекрытия этих столбцов следует выбрать импликанту .

[Минимальная дизъюнктивная нормальная форма](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B8%D0%BD%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B4%D0%B8%D0%B7%D1%8A%D1%8E%D0%BD%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BD%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0&action=edit&redlink=1) (МДНФ) заданной функции:

Структурная схема, соответствующая выражению в МДНФ (второй этап) при минимизации функции методом Квайна

 **(а)**

Структурная схема, соответствующая этому выражению приведена на рисунке слева. Переход от сокращённой схемы к МДНФ был осуществлён путём исключения лишних членов — импликант и . Покажем допустимость подобного исключения членов из логического выражения.

Импликанты  и  становятся равными *лог. 1* соответственно при следующих наборах значений аргументов:  и .

Роль этих импликант в выражении сокращённой формы функции заключается лишь в том, чтобы на приведённых наборах значений аргументов присваивать функции  значение 1. Однако при этих наборах функция равна 1 из-за остальных импликант выражения. Действительно, подставляя набор значений, указанных выше в формулу **(а)**, получаем:

* при  

;

* при 



## **Использование метода для получения минимальной**

Для получения Минимальной конъюнктивной нормальной формы (МКНФ), используя метод Куайна, вводятся следующие критерии:

* для минимизации берётся не [СДНФ](https://ru.wikipedia.org/wiki/%D0%A1%D0%94%D0%9D%D0%A4), а [СКНФ](https://ru.wikipedia.org/wiki/%D0%A1%D0%9A%D0%9D%D0%A4) функции;
* склеиваемые пары членов меняются на:  или ;
* правило операции поглощения выглядит следующим образом:



Метод представляет собой формализованный на этапе нахождения простых импликант метод Квайна. Формализация производится следующим образом:

1)Все конституанты единицы из СДНФ булевой функции f записываются их двоичными номерами.

2)Все номера разбиваются на непересекающиеся группы. Признак образования *i*-й группы: *i* единиц в каждом двоичном номере конституенты единицы.

3)Склеивание производят только между номерами соседних групп. Склеиваемые номера отмечаются каким-либо знаком (зачеркиванием).

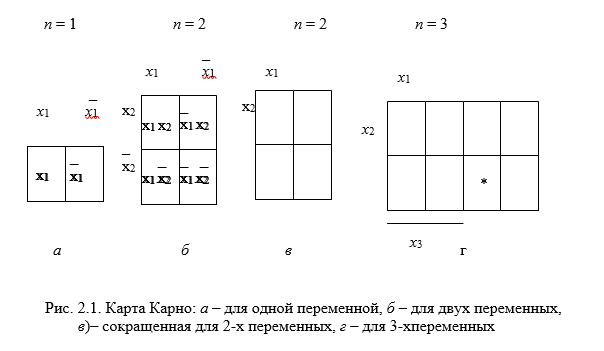
4)Склеивания производят всевозможные, как и в методе Квайна. Неотмеченные после склеивания номера являются простыми импликантами.

# **31. Минимизация логических выражений с диаграммами Вейча.**

Минимизация этим методом предполагает использование специальных форм – диаграмм Вейча (или карт *Карно*).

Карта Карно для «*n*» логических переменных представляет собой множество квадратов (клеток), объединенных в близкую к квадрату прямоугольную форму. Каждая такая клетка соответствует одному набору логических переменных, причем наборы двух соседних клеток должны отличаться на значение одной переменной (их наборы образуют склеивающиеся конъюнкции).

На рис. 2.1 приведены карты Карно для *n* = 1, 2, 3. На рис.2.1, *а*, 2.1,*б* показана разметка колонок и строк, а также указан для каждой составляющей клетки соответствующие ей набор. Разметка колонок (строк) указывает, какие значения данная переменная имеет в клетках, находящихся в данной колонке (строке). На рис. 2.1,*в* приведен пример компактной разметки карты, соответствующей карте на рис.2.1, *б*. Здесь помечаются колонки (строки), в которых соответствующая переменная имеет прямое значение. На рис. 2.1, *г* приведена карта Карно для *n* = 3, сформированная посредством зеркального отображения карты Карно для *n* = 2 (рис. 2.1, *в*) относительно правой границы. Этот прием универсальный; его можно использовать для построения карты для заданного «*n*» на основании имеющейся карты Карно для «n ˗ 1» переменной. Клетка, отмеченная знаком «\*», соответствует набору . Карты Карно используются для представления и минимизации логических функций.



Записываемая функция должна быть представлена в СДНФ. Запись функции в карту осуществляется за счет установки «1» в клетки карты, соответствующие конституентам единиц записываемой функции. Для выполнения минимизации представленной в карте Карно функции необходимо выполнить два этапа:

- охватить множество клеток карты Карно контурами;

- записать минимальное выражение для заданной функции в виде дизъюнкции конъюнкций, где каждая конъюнкция соответствует одному из введенных на карте контуров.

Охват клеток карты контурами выполняется с соблюдением следующих правил:

- контур должен иметь прямоугольную форму;

- в контур может входить количество клеток, равное целой степени числа «2»;

- в контур могут входить клетки, являющиеся логическими соседями;

- в контур необходимо включить максимальное количество клеток с учетом вышеприведенных требований;

- контурами необходимо охватить все клетки с единичными значениями;

- контуров должно быть минимальное количество;

- количество клеток в контуре должно быть равно 2DR, где DR –разность ранга (дельта ранга) конституент единицы заданной функции и ранга конъюнкции, соответствующей контуру.

*Логическими соседями* являются такие две клетки, наборы которых отличаются только одной переменной – в одном эта переменная должна иметь прямое, в другом – обратное значение.

Для того чтобы быть логическими соседями, клеткам достаточно быть геометрическими соседями. Считая, что карта является пространственным объектом и заворачивается по горизонтали и вертикали, сливаясь своими крайними горизонтальными и крайними вертикальными границами, можно считать, что соответствующие крайние горизонтальные и вертикальные клетки являются геометрическими соседями. Логическими соседями могут быть клетки, которые не являются геометрическими соседями. К числу таких клеток относятся клетки, которые по горизонтали или вертикали симметричны относительно линий зеркального отображения, которые были использованы при переходе от «*n*» к «*n*+1» переменным.

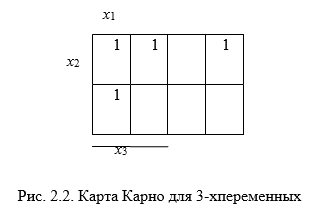
Запись минимального выражения по заданной функции имеет вид дизъюнкции простых конъюнкций, соответствующих контурам на карте, и формируется следующим образом:

- конъюнкция, соответствующая контуру, должна включать только те переменные, которые имеют постоянное значение во всех клетках, охваченных рассматриваемым контуром,

- или по другому: в конъюнкцию, соответствующую контуру, не должны входить переменные, которые имеют разные значения для клеток, охваченных рассматриваемым контуром.

Например, если задана логическая функция «*y*» трех переменных в виде выражения

то её запись в карту Карно будет иметь вид, приведенный на рис.2.2.



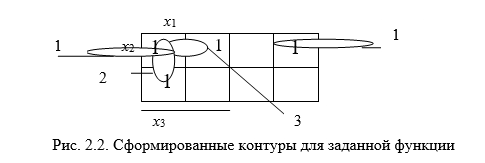
Для функции, заданной в карте Карно, приведенной на рис.2.2, контуры имеют вид, приведенный на рис.2.3.

Для примера, контур 1 представлен на рисунке в виде двух клеток: клетки, соответствующей набору *x*1*x*2*x*3, и клетки, соответствующей набору , поэтому данному контуру будет соответствовать конъюнкция x1 x2.

*Минимальное логическое выражение для функции имеет вид:*

*y* = *x*1 *x*2+ *x*1 *x*3 + *x*2 *x*3.

1 2 3



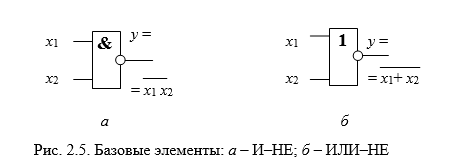
Конъюнкции минимального выражения помечены внизу цифрами, соответствующими номерам контуров, которые они представляют.

# **32. Логический базис И-НЕ. Синтез логических схем по логическому выражению в базисе И-НЕ.**

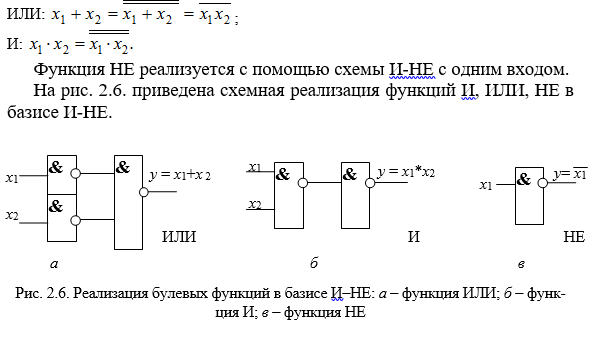
Булевый базис не является единственной функционально полной системой логических функций. Среди других наибольшее распространение получили базис И–НЕ и базис ИЛИ–НЕ.

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.



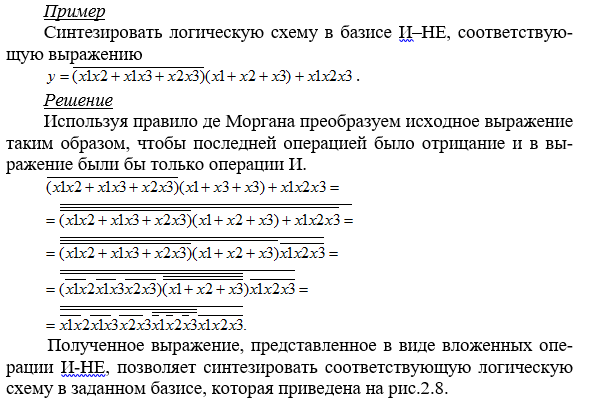
Реализация с помощью функции И-НЕ базовых функций алгебры Буля осуществляется следующим образом.

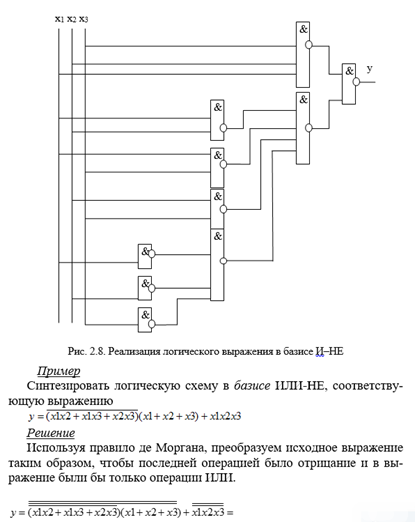


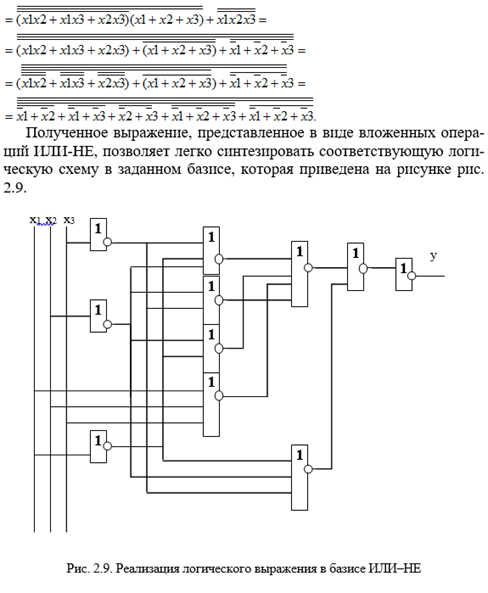
Реализация с помощью логической функции ИЛИ-НЕ базовых функций алгебры Буля осуществляется следующим образом.



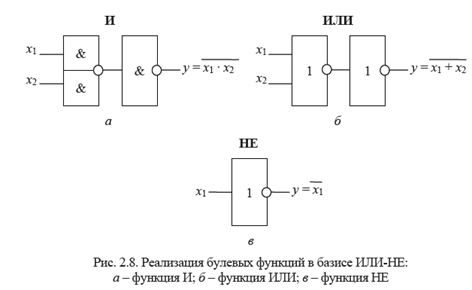
При синтезе логических схем в заданном базисе логических элементов (например, в базисах И–НЕ, или ИЛИ–НЕ) целесообразно предварительно исходное выражение привести к форме, в которой в выражении будут использованы только логические операции, соответствующие используемым логическим элементам в заданном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







**Логический базис ИЛИ-НЕ.Синтез логических схем по логическому выражению в базисе ИЛИ-НЕ.**

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

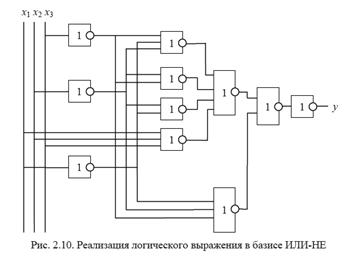
При синтезе логических схем в заданном базисе логических элементов (например, в базисах И–НЕ, или ИЛИ–НЕ) целесообразно предварительно исходное выражение привести к форме, в которой в выражении будут использованы только логические операции, соответствующие используемым логическим элементам в заданном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

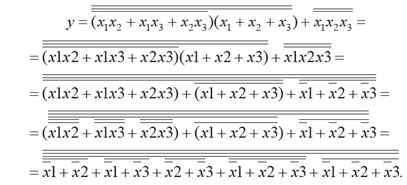
*Пример:*

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.



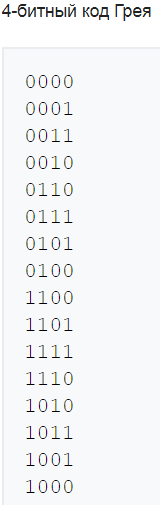
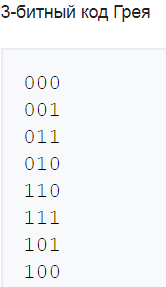
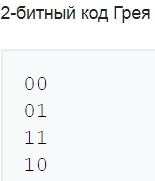
Полученное выражение, представленное в виде вложенных операций ИЛИ-НЕ, позволяет легко синтезировать соответствующую логическую схему в заданном базисе, которая приведена на рисунке рис. 2.10.

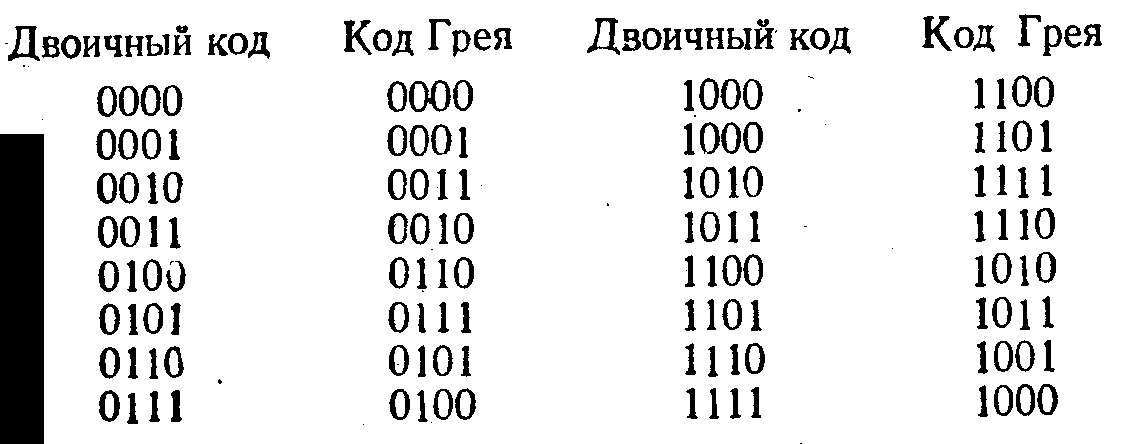
**33. Код Грея. Обратная польская запись**

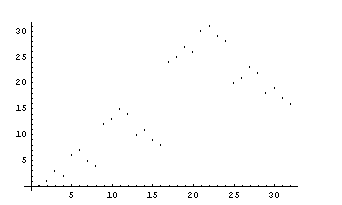
**Код Гре́я** — [двоичный код](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), в котором две «соседние» ([в упорядоченном, то есть лексикографическом, наборе](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BA%D1%81%D0%B8%D0%BA%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D0%BE%D1%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.

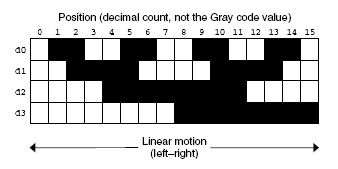
Наиболее часто на практике применяется **рефлексивный**[**двоичный код**](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4)**Грея**, хотя в общем случае существует бесконечное множество кодов Грея со значениями цифр в разрядах, взятых из различных алфавитов.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









Пример отображения Кода Грея как функции от бинарного числа

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 (см. [самоподобие](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%BC%D0%BE%D0%BF%D0%BE%D0%B4%D0%BE%D0%B1%D0%B8%D0%B5)).

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.

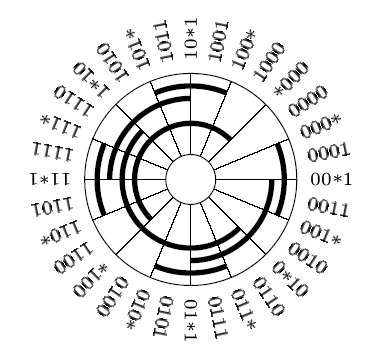
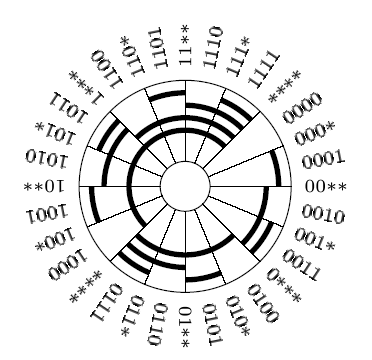
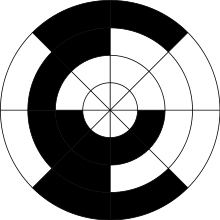
Коды Грея часто используются в датчиках-[энкодерах](https://ru.wikipedia.org/wiki/%D0%AD%D0%BD%D0%BA%D0%BE%D0%B4%D0%B5%D1%80). Их использование удобно тем, что два соседних значения шкалы сигнала отличаются только в одном разряде. Также они используются для кодирования номера дорожек в [жёстких дисках](https://ru.wikipedia.org/wiki/%D0%96%D1%91%D1%81%D1%82%D0%BA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

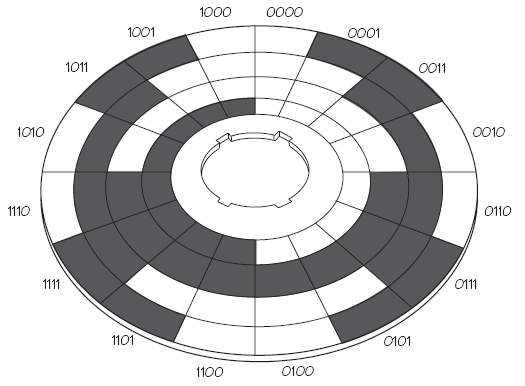
Широко применяются коды Грея и в [теории генетических алгоритмов](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) для кодирования генетических признаков, представленных целыми числами.

Код Грея используется для генерации [сочетаний](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%87%D0%B5%D1%82%D0%B0%D0%BD%D0%B8%D0%B5) [методом вращающейся двери](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B2%D1%80%D0%B0%D1%89%D0%B0%D1%8E%D1%89%D0%B5%D0%B9%D1%81%D1%8F_%D0%B4%D0%B2%D0%B5%D1%80%D0%B8&action=edit&redlink=1)[[1]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%93%D1%80%D0%B5%D1%8F#cite_note-1).

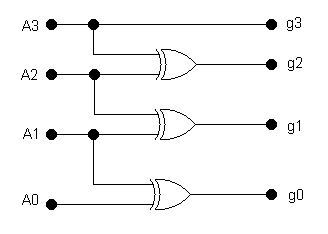
В некоторых [компьютерных играх](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5_%D0%B8%D0%B3%D1%80%D1%8B) (например, [Duke Nukem 3D](https://ru.wikipedia.org/wiki/Duke_Nukem_3D)) для успешного прохождения уровня требуется подобрать нужную комбинацию 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

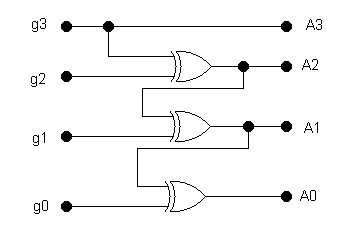




Конвертор бинарного кода в код Грея:



Конвертор кода Грея в бинарный код:



### 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.

Ниже приведён алгоритм преобразования из [двоичной системы счисления](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) в код Грея, записанный на языке [C](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)):

unsigned int grayencode(unsigned int g)

{

**return** g ^ (g >> 1);

}

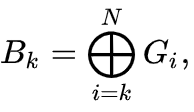
Однако, необходимо помнить, что данный алгоритм будет работать правильно, если компилятор реализует *нециклический* логический сдвиг (например, стандарт языка C не уточняет тип сдвига для знаковых чисел, но для unsigned типов поддержка гарантируется).

### Преобразование кода Грея в двоичный код

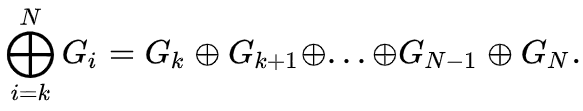
Обратный алгоритм — преобразование кода Грея в двоичный код — можно выразить рекуррентной формулой

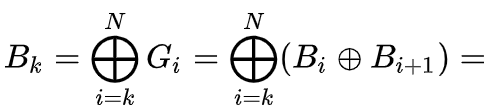
причём преобразование осуществляется побитно, начиная со старших разрядов, и значение Bi+1, используемое в формуле, вычисляется на предыдущем шаге алгоритма. Действительно, если подставить в эту формулу вышеприведённое выражение для *i*-го бита кода Грея, получим

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

где *N* — число битов в коде Грея (для увеличения быстродействия алгоритма в качестве *N* можно взять номер старшего ненулевого бита кода Грея); знак  означает суммирование при помощи операции «исключающее ИЛИ», то есть

Действительно, подставив в формулу выражение для *i*-го бита кода Грея, получим





Здесь предполагается, что бит, выходящий за рамки разрядной сетки B N+1, равен нулю.

Ниже приведена функция на языке С, реализующая данный алгоритм. Она осуществляет последовательный сдвиг вправо и суммирование исходного двоичного числа, до тех пор, пока очередной сдвиг не обнулит слагаемое.

unsigned int graydecode(unsigned int gray)

{

unsigned int bin;

**for** (bin = 0; gray; gray >>= 1) {

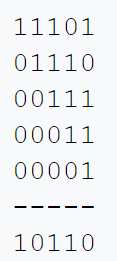
bin ^= gray;

}

**return** bin;

}

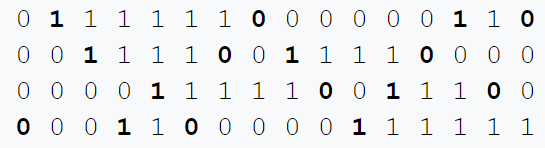
Пример: преобразовать код Грея 11101 в двоичный код.



## **Необычные вариации кода Грея**

### *Сбалансированный код Грея*

Если датчики имеют ограниченный [ресурс](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%81%D1%83%D1%80%D1%81_(%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0)) по количеству переключений, хотелось бы, чтобы они изнашивались равномерно. В сбалансированном коде Грея в разных разрядах количество переключений настолько близко, насколько можно.



Здесь в 4-битном коде каждый разряд переключается четырежды. В 5-битном коде такое невозможно, приходится переключать один бит 8 раз, остальные — по 6.

### *Однодорожечный код Грея*

Код Грея является однодорожечным, если все столбцы матрицы являются кольцевыми сдвигами друг друга. Это позволяет сделать угловой датчик с одной дорожкой.

Двухбитный код Грея является однодорожечным, это можно увидеть в [компьютерной мыши](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%D0%B0%D1%8F_%D0%BC%D1%8B%D1%88%D1%8C) — как в шариковом механизме старых мышей, так и в колесе прокрутки новых. Два датчика стоят в разных точках одной дорожки. Если довести эту систему до крайности — половина диска «чёрная», половина «белая», и датчики стоят на 90° друг относительно друга — то можно узнать абсолютное положение диска с дискретностью в 90°.

Для кодов Грея более высокой разрядности это не так, приходится увеличивать количество дорожек, это делает датчик громоздким и дорогим. Поэтому, если возможно, обходятся двумя дорожками — одна для двухбитного кода Грея, и одна — позиция нуля. Однако существуют коды, где дорожка именно одна, правда, все 2n позиций так закодировать невозможно. Для 5 бит рекорд — 30 позиций, для 9 — 360.

### *Двухмерный код Грея*

Используется в [квадратурной модуляции](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F) сигналов. Соседние точки «[созвездия](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%81%D0%BE%D0%B7%D0%B2%D0%B5%D0%B7%D0%B4%D0%B8%D0%B5)» отличаются одним битом, диагональные — двумя.

Приведем еще один (на этот раз нерекуррентный) алгоритм генерации кодов Грея. Будем рассматривать бинарные коды Грея порядка n. Итак, на вход алгоритма подается единственное число n, которое указывает порядок кода Грея. По ходу выполнения алгоритма мы получим последовательность всех подмножеств n-элементного множества, в которой каждое последующее подмножество получается из предыдущего добавлением или удалением единственного элемента (наименьшим возможным изменением) — код Грея. При этом каждое подмножество будет представляться бинарной последовательностью B[1], …, B[n].

Gray-Generation(n)

1 **for** i := 1 **to** n do B[i] := 0;

2 i := 0;

3 **repeat**

4 **write** (B[i], …, B[n]);

5 i := i + 1; p := 1; j := i;

6 **while** j **mod** 2 = 0 **do**

7 **begin**

8 j := j/2; p := p + 1;

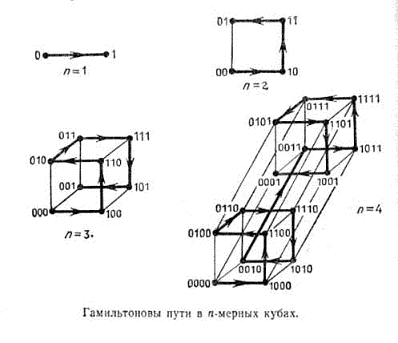
9 **end**;

10 **if** p ≤ n **then** B[p] := 1 − B[p];

11 **until** p > n

Доказательство корректности работы алгоритма можно найти в [1].

Последовательность полученных подмножеств можно проиллюстрировать на графе (n-мерном кубе), вершины которого соответствуют бинарным последовательностям длины n и две вершины которого соединены ребром, если соответствующие последовательности отличаются лишь в одной позиции. Тогда эта последовательность соответствует **гамильтонову пути** в этом графе, т. е. пути, содержащему каждую вершину графа только один раз.



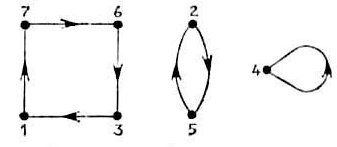
## Перестановки

**Перестановка** n-элементного множества X — это взаимно однозначная функция f: X→X. Если для простоты принять X = {1, …, n}, то перестановкой можно назвать упорядоченный набор из n различных чисел, лежащих в промежутке от 1 до n (далее будет рассматриваться именно этот случай).

Обозначим множество всех перестановок множества X через Sn. Понятно, что |Sn| = n!, т. к. на первое место набора можно поставить любое число из n возможных, на второе — любое из n−1 оставшихся и т. д. до последней позиции, в которую мы помещаем последний оставшийся элемент: в результате, мы получим произведение, которое и будет являться факториалом n.

Рассмотрим множество Mi из предыдущей части. Пусть каждое Mi будет представлять собой множество чисел от 0 до i−1. Обозначим через Tn произведение n таких множеств. Тогда если будет существовать взаимнооднозначное соответствие между Sn и Tn, то мы сможем, в частности, перенумеровать все перестановки Sn. Предъявим механизм перехода от перестановки r1, …, rn ∈ Sn к элементу t1, …, tn ∈ Tn. Для этого для любого i ∈ 1:n найдем номер позиции s значения i в перестановке (rs = i) и примем в качестве ti количество элементов меньших i среди r1, …, rs−1. А при обратном переходе от элемента к перестановке, можно использовать то, что место любого элемента i (начиная с самого большого) на единицу больше, чем число элементов, предшествующих i, а само это число равно сумме ti и числа элементов, бОльших i.

Кроме того, каждую перестановку f ∈ Sn можно представить с помощью ориентированного графа с множеством вершин X = {1, …, n}, в котором ребро идет от x к у, когда f(x) = y. Легко можно показать, что такой граф состоит из некоторого числа элементарных циклов с различными множествами вершин, которые в сумме дают множество X. Т. е. каждую перестановку мы можем разложить на циклы. Например, рассмотрим перестановку f(1234567) = 7514236 и разобьем ее на циклы:



### Генерирование перестановок

Введем некоторые обозначения для записи алгоритмов. Элементы перестановки будем запоминать в виде элементов массива P[1], …, P[n]. Обмен значениями переменных будем обозначать через P[i] :=: P[j]. Напомним, что мы рассматриваем только перестановки, заданные на множестве X = {1, …, n}.

На множестве Sn зададим **лексикографический порядок**:

(x1, …, xn) < (y1, …, yn) ⇔ существует k ≥ 1, такое что xk ≤ yk и xl = yl для каждого l < k.

Аналогично задается **антилексикографический порядок**:

(x1, …, xn) < (y1, …, yn) ⇔ существует k ≤ n, такое что xk > yk и xl = yl для каждого l > k.

Теперь представим алгоритм генерации всех перестановок в антилексикографическом порядке. Заметим, что этот алгоритм является рекурсивным.

1 **procedure** REVERSE(m);

2 **begin** i := 1; j := m;

3 **while** i < j **do**

4 **begin** P[i] :=: P[j]; i := i + 1; j := j - 1;

5 **end**

6 **end**

7 **procedure** ANTYLEX(m);

8 **begin**

9 **if** m = 1 **then**

10 write(P[1], …, P[n])

11 **else**

12 **for** i := 1 **to** m **do**

13 **begin** ANTYLEX(m − 1);

14 **if** i < m **m** **then**

15 **begin** P[i] :=: P[m]; REVERSE(m − 1)

16 **end**

17 **end**

18 **end**;

19 **begin**

20 **for** i := 1 **to** n **do** P[i] :=: i;

21 ANTYLEX(n)

22 **end**

Среднее число транспозиций, приходящихся на каждую перестановку, приблизительно равно 1.543, т. е. для порождения n! перестановок используется приблизительно 3.077n! сравнений.

Можно предложить более быстрый алгоритм, в котором каждая следующая перестановка образуется из предыдущей с помощью только одной транспозиции. Приведем алгоритм генерации перестановок за минимальное число транспозиций:

1 **procedure** B(m, i);

2 **begin**

3 **if** (m **mod** 2 = 0) **and** (m > 2) **then**

4 **if** i < m − 1 **then** B := i

5 **else** B := m − 2

6 **else** B := m − 1

7 **end**;

8 **procedure** PERM(m);

9 begin

10 **if** m = 1 **then**

11 write(P[1], …, P[n])

12 **else**

13 **for** i := 1 **to** m **do**

14 **begin** PERM(m − 1);

15 **if** i < m **then** P[B(m, i)] :=: P[m]

16 **end**

17 **end**;

18 **begin**

19 **for** i := 1 **to** n **do** P[i] :=: i;

20 PERM(n)

21 **end**

Доказательство корректности алгоритма можно найти в [1].

Однако, существует еще более быстрый алгоритм генерирования перестановок, он строит последовательность перестановок, в которой каждая следующая образуется из предыдущей с помощью однократной транспозиции соседних элементов. Обозначим через PR[i] булеву переменную, содержащую информацию о том, переносится ли элемент i вперед (PR[i] = true) или назад. Переменнная C[i] будет показывать, какую из возможных n−i+1 позиций элемент i занимает относительно элементов i+1, …, n на своем пути вперед и назад. Вот его код:

1 **begin**

2 **for** i := 1 **to** n **do**

3 **begin** P[i] := i; C[i] := 1; PR[i] := **true**;

4 **end**;

5 C[n] := 0;

6 write(P[1], …, P[n]);

7 i := 1;

8 **while** i < n **do**

9 **begin** i := 1; x := 0;

10 **while** C[i] = n − i + 1 **do**

11 **begin** PR[i] := **not** PR[i]; C[i] := 1;

12 **if** PR[i] **then** x := x + 1;

13 i := i + 1;

14 **end**;

15 **if** i < n **then**

16 **begin**

17 **if** PR[i] **then** k := C[i] + x;

18 **else** k := n − i + 1 − C[i] + x;

19 P[k] :=: P[k + 1];

20 write(P[1], …, P[n]);

21 C[i] := C[i] + 1

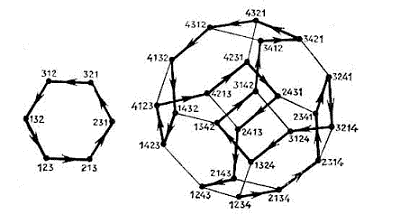
22 **end**

23 **end**

24 **end**

На эту тему можно посмотреть визуализаторы [3] и [4].

Последовательность перестановок, полученная с помощью данного алгоритма имеет интересную интерпретацию. Так, если рассмотреть граф, вершины которого соответствуют всем перестановкам и в котором две вершины, соответствующие перестановкам f и g, соединены ребром, если g образуется из f однократной транспозицией соседних элементов, то полученная последовательность является гамильтоновым путем в этом графе. На рисунке изображен граф последовательности для n = 3, 4.



***Обратная польская запись***

Обратная польская нотация была разработана австралийским философом и специалистом в области теории вычислительных машин Чарльзом Хэмблином в середине 1950-х на основе польской нотации, которая была предложена в 1920 году польским математиком Яном Лукасевичем. Отличительной особенностью обратной польской нотации является то, что все аргументы (или операнды) 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.

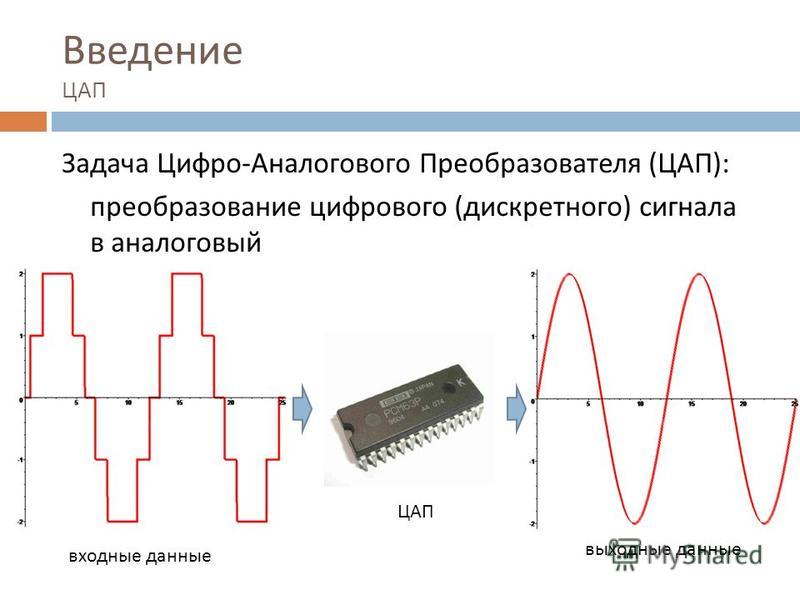
• Выражение читается слева направо. Когда в выражении встречается знак операции, выполняется соответствующая

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.

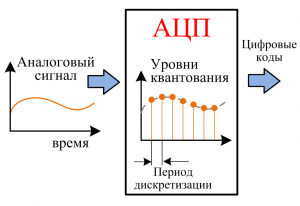
• Результатом вычисления выражения становится результат последней вычисленной операции.

# 34.АЦП и ЦАП

**Цифро-аналоговый преобразователь** (**ЦАП**) — устройство для преобразования цифрового (обычно двоичного) кода в [аналоговый сигнал](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D0%BE%D0%B3%D0%BE%D0%B2%D1%8B%D0%B9_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB) ([ток](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_%D1%82%D0%BE%D0%BA), [напряжение](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%BE%D0%B5_%D0%BD%D0%B0%D0%BF%D1%80%D1%8F%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) или [заряд](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)). Цифро-аналоговые преобразователи являются [интерфейсом](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) между дискретным цифровым миром и аналоговыми сигналами.



**Аналого-цифровой преобразователь**   (*АЦП*, [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) *Analog-to-digital converter, ADC*) — устройство, преобразующее входной [аналоговый сигнал](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D0%BE%D0%B3%D0%BE%D0%B2%D1%8B%D0%B9_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB) в дискретный код ([цифровой сигнал](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB)).



Обратное преобразование осуществляется при помощи [цифро-аналогового преобразователя](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE-%D0%B0%D0%BD%D0%B0%D0%BB%D0%BE%D0%B3%D0%BE%D0%B2%D1%8B%D0%B9_%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C)(ЦАП, DAC).

Как правило, АЦП — [электронное](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%B0) устройство, преобразующее [напряжение](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%BE%D0%B5_%D0%BD%D0%B0%D0%BF%D1%80%D1%8F%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) в двоичный цифровой код. Тем не менее, некоторые неэлектронные устройства с цифровым выходом следует также относить к АЦП, например, некоторые типы [преобразователей угол-код](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D1%82%D1%87%D0%B8%D0%BA_%D1%83%D0%B3%D0%BB%D0%B0_%D0%BF%D0%BE%D0%B2%D0%BE%D1%80%D0%BE%D1%82%D0%B0). Простейшим одноразрядным двоичным АЦП является [компаратор](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BE%D1%80).

Разрешение АЦП — минимальное изменение величины аналогового сигнала, которое может быть преобразовано данным АЦП — связано с его разрядностью. В случае единичного измерения без учёта шумов разрешение напрямую определяется *разрядностью*АЦП.

Разрядность АЦП характеризует количество дискретных значений, которые преобразователь может выдать на выходе. В двоичных АЦП измеряется в [битах](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82), в троичных АЦП измеряется в [тритах](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B8%D1%82). Например, двоичный 8-разрядный АЦП способен выдать 256 дискретных значений (0…255), поскольку {\displaystyle 2^{8}=256}, троичный 8-разрядный АЦП способен выдать 6561 дискретное значение, поскольку {\displaystyle 3^{8}=6561}.

Разрешение по напряжению равно разности напряжений, соответствующих максимальному и минимальному выходному коду, делённой на количество выходных дискретных значений. Например:

* Пример 1
  + Диапазон входных значений = от 0 до 10 вольт
  + Разрядность двоичного АЦП 12 бит: 212 = 4096 уровней квантования
  + Разрешение двоичного АЦП по напряжению: (10-0)/4096 = 0,00244 вольт = 2,44 мВ
  + Разрядность троичного АЦП 12 трит: 312 = 531 441 уровень квантования
  + Разрешение троичного АЦП по напряжению: (10-0)/531441 = 0,0188 мВ = 18,8 мкВ
* Пример 2
  + Диапазон входных значений = от −10 до +10 вольт
  + Разрядность двоичного АЦП 14 бит: 214 = 16384 уровня квантования
  + Разрешение двоичного АЦП по напряжению: (10-(-10))/16384 = 20/16384 = 0,00122 вольт = 1,22 мВ
  + Разрядность троичного АЦП 14 трит: 314 = 4 782 969 уровней квантования
  + Разрешение троичного АЦП по напряжению: (10-(-10))/4782969 = 0,00418 мВ = 4,18 мкВ

На практике разрешение АЦП ограничено [отношением сигнал/шум](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB/%D1%88%D1%83%D0%BC) входного сигнала. При большой интенсивности шумов на входе АЦП различение соседних уровней входного сигнала становится невозможным, то есть ухудшается разрешение. При этом реально достижимое разрешение описывается ***эффективной разрядностью*** ([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) *effective number of bits, ENOB*), которая меньше, чем реальная разрядность АЦП. При преобразовании сильно зашумлённого сигнала младшие разряды выходного кода практически бесполезны, так как содержат шум. Для достижения заявленной разрядности отношение сигнал/шум входного сигнала должно быть примерно 6 дБ на каждый бит разрядности (6 дБ соответствует двукратному изменению уровня сигнала).

# **35. Искусство управления сложностью. цифровая абстракция**

1.2 ИСКУССТВО УПРАВЛЕНИЯ СЛОЖНОСТЬЮ Одной из характеристик, отличающих профессионального инженераэлектронщика 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.

1.2.1 Абстракция

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 самом низком уровне абстракции находится физика, изучающая движение электронов. Поведение электронов описывается квантовой механикой и системой уравнений Максвелла. Рассматриваемая нами современная электронная система состоит из полупроводниковых устройств (devices), таких как транзисторы (а когдато это были электронные лампы). Каждое такое устройство имеет четко определенные точки соединения с другими подобными устройствами. Эти точки мы будем называть контактами (в англоязычной литературе используется термин terminal). Любое электронное устройство может быть представлено абстрактной математической моделью, описывающей изменяющуюся во времени взаимозависимость тока и напряжения. Такие же изменения тока и напряжения можно наблюдать на экране осциллографа, если подключить осциллограф к контактам реального устройства. Данный подход означает, что, если рассматривать систему на уровне устройств, функции которых однозначно определены, то можно не учитывать поведение электронов внутри отдельных устройств этой системы. Следующий уровень абстракции – это аналоговые схемы (analog circuits), в которых полупроводниковые устройства соединены таким образом, чтобы они образовывали функциональные компоненты, такие как усилители, например. Напряжение на входе и на выходе аналоговой цепи изменяется в непрерывном диапазоне. В отличие от аналоговых цепей, цифровые схемы (digital circuits), такие как логические вентили, используют два строго ограниченных дискретных уровня напряжения. Один из этих дискретных уровней – это логический нуль, другой – логическая единица. В разделах этой книги, посвященных разработке цифровых схем и устройств, мы будем использовать простейшие цифровые схемы для построения сложных цифровых модулей, таких как сумматоры и блоки памяти. Микроархитектурный уровень абстракции, или просто микроархитектура (microarchitecture), связывает логический и архитектурный уровни абстракции. Архитектурный уровень абстракции, или архитектура (architecture), описывает компьютер с точки зрения программиста. Например, архитектура Intel x86, используемая микропроцессорами большинства персональных компьютеров (ПК), определяется набором инструкций и регистров (памяти для временного хранения переменных), доступным для использования программистом. Микроархитектура – это соединение простейших цифровых элементов в логические блоки, предназначенные для выполнения команд, определенных какой-то конкретной архитектурой. Отдельно взятая архитектура может быть реализована с использованием различных вариантов микроархитектур с разным соотношением цены, производительности и потребляемой энергии, и такое соотношение зачастую выбирается как баланс между этими тремя факторами. Процессоры Intel Core i7, Intel 80486 и AMD Athlon, например, используют одну и ту же архитектуру x86, но реализованную с использованием трех разных микроархитектурных решений. Теперь мы перемещаемся в область программного обеспечения. Операционная система (operating system) управляет операциями нижнего уровня, такими как доступ к жесткому диску или управление памятью. И, наконец, программное обеспечение использует ресурсы операционной системы для решения конкретных задач пользователя. Именно принцип абстрагирования от маловажных деталей позволяет вашей бабушке общаться с внуками в Интернете, не задумываясь о квантовых колебаниях электронов или организации памяти компьютера.

**1.2.2 Конструкторская дисциплина** 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 контексте данной книги соблюдение конструкторской дисциплины в виде максимального использования цифровых схем играет очень важную роль. В цифровых схемах используются дискретные значения напряжения, в то время как в аналоговых схемах напряжение изменяется непрерывно. Таким образом, цифровые схемы, которые можно рассматривать как подмножество аналоговых цепей, в некотором смысле уступают по своим характеристикам более широкому классу аналоговых цепей. Однако цифровые цепи гораздо проще проектировать. Ограничивая использование аналоговых схем и по возможности заменяя их цифровыми, мы можем легко объединять отдельные компоненты в сложные системы, которые, в конечном итоге, для большинства приложений превзойдут по своим параметрам системы, построенные на аналоговых цепях. Примером тому могут служить цифровые телевизоры, компакт-диски (CD) и мобильные телефоны, которые уже практически полностью вытеснили своих аналоговых предшественников.

**1.2.3 Три базовых принципа** 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 применительно 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.

**1.3 ЦИФРОВАЯ АБСТРАКЦИЯ**

Большинство физических величин изменяется непрерывно. Например, напряжение в электрическом проводе, частота колебаний или распределение массы – все это параметры, изменяющиеся непрерывно. Цифровые системы, с другой стороны, представляют информацию в виде дискретно меняющихся переменных с конечным числом строго определённых значений. Одной из наиболее ранних цифровых систем стала Аналитическая Машина Чарльза Бэббиджа, которая использовала переменные с десятью дискретными значениями. Начиная с 1834 года и до 1871 года 1 Бэббидж разрабатывал и пытался построить этот механический компьютер. Шестеренки Аналитической Машины могли находится в одном из десяти фиксированных положений, а каждое такое положение было промаркировано от 0 до 9 подобно механическому счетчику пробега автомобиля. Рис. 1.3 показывает, как выглядел прототип Аналитической Машины. Каждый ряд шестеренок такой машины обрабатывал одну цифру. В своем механическоммкомпьютере Бэббидж использовал 25 рядов шестеренок таким образом, чтобы машина обеспечивала вычисления с точностью до 25-го знака. В отличие от машины Бэббиджа большинство электронных компьютеров использует двоичный (бинарный) код. В случае двоичного кода высокое напряжение – это единица, а низкое напряжение – нуль, поскольку гораздо легче оперировать двумя уровнями напряжения, чем десятью. Объем информации D, передаваемый одной дискретной переменной, которая может находиться в N различных состояниях, измеряется в единицах, называемых битами, и вычисляется по следующей формуле:

D = log2N bits

Двоичная переменная передает log22 = 1 – один бит информации. Теперь вам, вероятно, понятно, почему единица информации называется битом. Bit (бит) – это сокращение от английского binary digit, что дословно переводится как двоичный разряд. Каждая шестеренка в машине Бэббиджа содержит log210 = 3,322 бит информации, поскольку она может находиться в одном из 23,322 = 10 уникальных положений. Теоретически непрерывный сигнал может передавать бесконечное количество информации, поскольку может принимать неограниченное число значений. На практике, однако, шум и ошибки измерения ограничивают информацию, передаваемую большинством непрерывных сигналов, диапазоном от 10 бит до 16 бит. Если же измерение уровня сигнала должно быть произведено очень быстро, то объём передаваемой информации будет еще ниже (в случае 10 бит, например, это будет только 8 бит). Предмет этой книги – цифровые схемы, использующие двоичные переменные нуль и единицу. Джордж Буль разработал систему логики, использующую двоичные переменные, и эту систему сегодня называют его именем – Булева логика. Булевы переменные могут принимать значения ИСТИНА (TRUE) или ЛОЖЬ (FALSE). В электронных компьютерах положительное напряжение обычно представляет единицу, а нулевое напряжение представляет нуль. В этой книге мы будем использовать понятия единица (1), ИСТИНА (TRUE) и ВЫСОКОЕ (HIGH) как синонимы. Аналогичным образом мы будем использовать нуль (0), ЛОЖЬ (FALSE), и НИЗКОЕ (LOW) как взаимозаменяемые термины.

**Преимущества цифровой абстракции** заключаются в том, что разработчик цифровой системы может сосредоточиться исключительно на единицах и нулях, полностью игнорируя, каким образом булевы переменные представлены на физическом уровне. Разработчика не волнует, представлены ли нули и единицы определенными значениями напряжения, вращающимися шестернями или уровнем гидравлической жидкости. Программист может продуктивно работать, не располагая детальной информацией об аппаратном обеспечении компьютера. Однако, понимание того, как работает это аппаратное обеспечение, позволяет программисту гораздо лучше оптимизировать программу для конкретного компьютера.

Как вы могли видеть выше, один-единственный бит не может передать большого количества информации. Поэтому в следующем разделе мы рассмотрим вопрос о том, каким образом набор битов можно использовать для представления десятичных чисел. В последующих главах мы также покажем, как группы битов могут представлять буквы и даже целую программу.

**36.Логические элементы. Таблицы истинности. Обозначения в разных представления.**

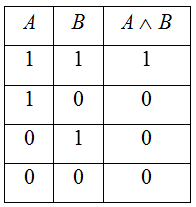
Элементы ЭВМ, являющиеся представителями низшего иерархического уровня архитектуры компьютера – цифрового логического уровня (схемотехнического), реализованные на радиотехнических деталях, представляют собой мельчайшие компоненты, на основе которых строятся более крупные составляющие вычислительной машины.

- 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)

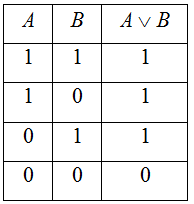
**Таблицы истинности для основных двоичных логических функций**

**Обозначение:**



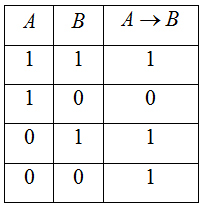
1. Конъюнкция (логическое умножение) – сложное логическое выражение, которое является истинным только в том случае, когда истинны оба входящих в него простых выражения

**Обозначение:**



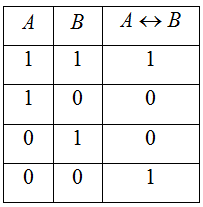
2. Дизъюнкция (логическое сложение) – это сложное логическое выражение, которое истинно, если хотя бы одно из простых логических выражений истинно и ложно, если оба простых логических выражения ложны.

**Обозначение:**



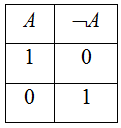
3. Импликация (логическое следствие) – это сложное логическое выражение, которое является ложным тогда и только тогда, когда условие истинно, а следствие ложно.

**Обозначение:**



4. Эквиваленция – это сложное логическое высказывание, которое является истинным только при одинаковых значениях истинности простых выражений, входящих в него.

**Обозначение:**



# 5. Логическое отрицание (инверсия) делает истинное высказывание ложным и, наоборот, ложное – истинным. **37.** **За пределами цифровой абстракции. Напряжение питания. Логические уровни. Допускаемые Уровни Шумов. Передаточная Характеристика. Статическая Дисциплина.**

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 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)*,составляет0В.Самое высокое напряжениев 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**

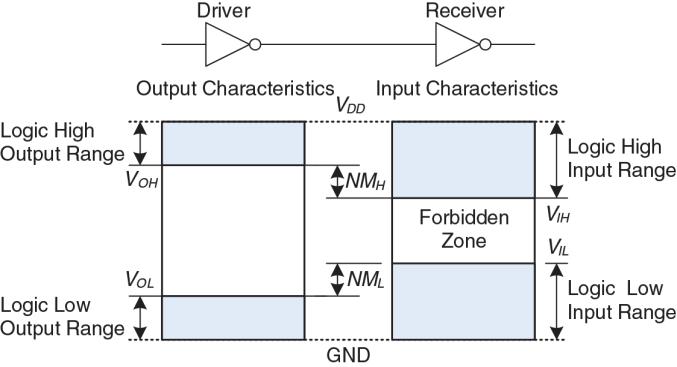
Отображение непрерывно-меняющейся переменной на различные значения дискретной двоичной переменной выполняется путем определения *логических уровней,* как показано на [**Рис.** **1.23**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page121) Первый логический элемент в рассматриваемой схеме называется *источник* *(driver)*,а второй– *приемник (receiver)*.Выходной сигнал источникаподключается 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 *VOL* < *VIL* и *VOH* > *VIH*.В этом случае,даже если выходной сигнал источника будетзагрязнен 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**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page121) 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 = VOL – VIH* | (1.3) |

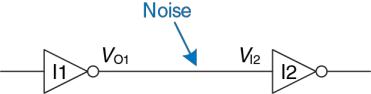


**Рис. 1.23 Логические уровни и уровни шума**

*VDD* обозначает напряжение стока(*drain*)в транзисторах,построенныхна структуре металл-оксид-полупроводник (МОП). Такие транзисторы используются сегодня для создания самых современных микросхем. Напряжение источника питания иногда также обозначают *VCC*, как напряжение коллектора (*collector*) в биполярных транзисторах более ранних микросхем. Напряжение земли (ground voltage или просто ground) иногда обозначают как *VSS* потому, что это напряжение на истоке (*source*) МОП-транзистора. Для более подробной информации о том, как функционирует транзистор, смотри **раздел** [**1.7**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page128)

**Пример 1.18** РАСЧЕТ УРОВНЕЙ ШУМА

Рассмотрим схему с инверторами на [**Рис.** **1.24**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page122) *VO1* – это напряжение на выходе инвертора I1, а *VI2* – напряжение на входе инвертора I2. Оба инвертора имеют следующие характеристики: *VDD* = 5 В, *VIL* = 1,35 В, *VIH* = 3,15 В, *VOL* = 0,33 В, и *VOH* = 3,84В.Каковы нижний и верхний уровни шума?Может ли схема корректнообработать уровень шума в 1 В между *VO1* и *VI2*?



**Рис. 1.24 Схема с инверторами**

**Решение:** Границы уровня шума инвертора следующие:NM*L*=*VIL*–*VOL*=(1,35 В − 0,33 В) = 1,02 В, NM*H* = *VOH* – *VIH* = (3,84 В − 3,15 В) = 0,69 В. Схема может корректно обработать шум в 1В когда на выходе НИЗКИЙ уровень (NML = 1,02 В), но не когда на выходе ВЫСОКИЙ уровень (NM*H* = 0,69 *V*). Например, предположим, что инвертор I1 имеет на выходе в наихудшем случае ВЫСОКОЕ значение, *VO1* = *VOH* = 3,84 В. Если наличие шума вызовет падение напряжения на 1 В на входе инвертора I2, тогда *VI2 =* (3,84 В − 1 В) = 2,84 В. Это меньше, чем допустимое входное значение ВЫСОКОГО уровня, *VIH* = 3,15 В, поэтому инвертор I2 может не принять правильное входное значение ВЫСОКОГО уровня.

**1.6.4 Передаточная Характеристика**

Для понимания предела цифровой абстракции мы должны рассмотреть поведение логических вентилей с аналоговой точки зрения. *Передаточная характеристика (DС transfer characteristics)* какого-либологического 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,как показано на[**Рис.****1.25**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page125) *(a)*.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 электроннойсистемы 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 меняетсябыстро. Переходные процессы рассматриваются в **разделе** [**2.9**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page288)

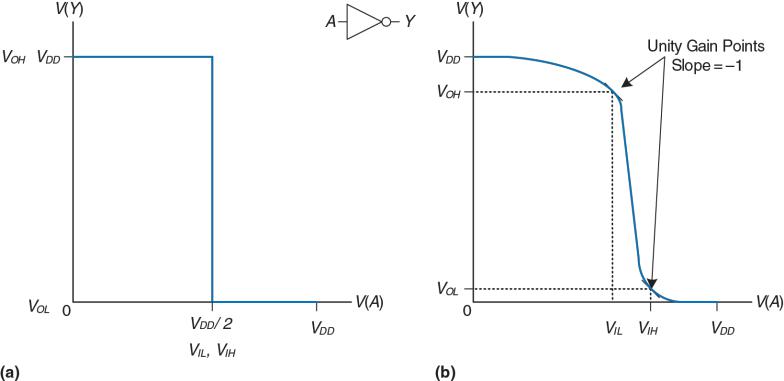
Напряжение при переключении реального инвертора изменяется постепенно между граничными значениями – так, как показано на [**Рис. 1.25**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page125) **(b)**.Если входное напряжение*V*(*A*)равно0,то 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 d*V*(*Y*)/d*V*(*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.

**1.6.5 Статическая Дисциплина**

Для того, чтобы избежать попадания входных сигналов в запретные зоны, логические вентили должны разрабатываться в соответствии с *принципом статической дисциплины (static discipline)*.Принципстатической дисциплины требует, чтобы при условии наличия логически корректных сигналов на входе каждый элемент системы выдавал логически корректные сигналы на выходе.



**Рис. 1.25 Передаточные характеристики и уровни шума**

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 семействапри 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 этихсемейств представлены в [**Табл.** **1.4**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page127) 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**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page1500) наиболее распространённые семейства логических вентилей рассматриваются детально.

**Табл. 1.4 Семейства логики с уровнями напряжения 5 В и 3,3 В**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Семейство логики | *VDD* | |  | *VIL* | *VIH* | *VOL* | *VOH* |
|  | TTL | 5 (4,75 | − | 5,25) | 0,8 | 2,0 | 0,4 | 2,4 |
|  |  |  | | |  |  |  |  |
|  | CMOS | 5 (4,5 − 6) | | | 1,35 | 3,15 | 0,33 | 3,84 |
|  | LVTTL | 3.3 (3 − | | 3,6) | 0,8 | 2,0 | 0,4 | 2,4 |
|  |  |  | |  |  |  |  |  |
|  | LVCMOS | 3.3 (3 − | | 3,6) | 0,9 | 1,8 | 0,36 | 2,7 |
|  |  |  |  |  |  |  |  |  |

**Пример 1.19** СОВМЕСТИМОСТЬ ЛОГИЧЕСКИХ СЕМЕЙСТВ

Какие из логических семейств из [**Табл.** **1.4**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page127) могут надежно взаимодействовать между собой?

**Решение:** в[**Табл. 1.5**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page128) перечисленылогические семейства,которые имеютсовместимые логические уровни. Заметим, что пятивольтовые логические семейства, такие как TTL и CMOS, могут выдавать на выход ВЫСОКИЙ уровень 5 В. Если этот пятивольтовый сигнал подается на вход семейству с уровнем 3,3 В, такому как LVTTL или LVCMOS, это может повредить приемник, если в спецификации последнего не указана прямо, что он «5 В-совместимый».

**Табл. 1.5 Совместимость логических семейств**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  |  |  |  |  | 72 | |
|  | | | |  |  |  |  |  |  | |
|  |  | | | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | Приемник | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | TTL | | CMOS |  | LVTTL | LVCMOS |  |
| Источник | TTL | | | ДА | | НЕТ: VOH < VIH |  | ВОЗМОЖНОa | ВОЗМОЖНОa |  |
|  | CMOS |  |  | ДА |  | ДА |  | ВОЗМОЖНОa | ВОЗМОЖНОa |  |
|  | LVTTL | | | ДА | | НЕТ: VOH < VIH |  | ДА | ДА | |
|  | LVCMOS |  |  | ДА |  | НЕТ: VOH < VIH |  | ДА | ДА |  |

1. если сигнал в 5 В ВЫСОКОГО уровня не может повредить вход приемника.

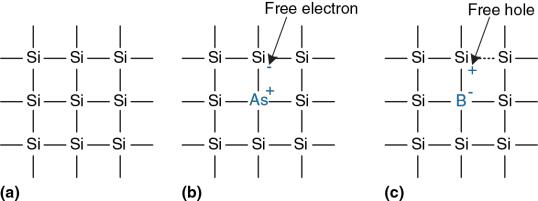
# **38. Биполярные и КМОП транзисторы. Полупроводники. Конденсаторы. n-МОП и p-МОП-транзисторы**

Существуют два основных типа транзисторов – биполярные транзисторы (bipolar junction transistors) и МОП-транзисторы – металл-оксид-полупроводник- транзисторы (иногда говорят полевые транзисторы – metal-oxide-semiconductor field effect transistors, или MOSFET).

**1.7.1 Полупроводники**

МОП-транзисторы изготовляются из кремния – элемента, преобладающего в скальной породе и песке. Кремний (Si) – это элемент IV атомной группы, то есть он имеет четыре валентных электрона, может образовывать связи с четырьмя соседними атомами и, таким образом, формировать кристаллическую *решетку* *(lattice)*. На [**Рис. 1.26**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page133) **(a)**,для простоты,кристаллическая решетка показана вдвумерной системе координат, однако полезно помнить, что реальная кристаллическая решетка имеет форму куба. Линия на [**Рис.** **1.26**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page133) (a) изображает ковалентную связь. По своей природе, кремний – плохой проводник, потому что все электроны заняты в ковалентных связях. Однако проводимость кремния улучшается, если добавить в него небольшое количество атомов другого вещества, называемого *примесью (dopant)*.Если в качестве примеси используется элементV атомной группы, например, мышьяк (As), то в каждом атоме примеси окажется дополнительный электрон, не участвующий в образовании ковалентных связей. Этот свободный электрон может легко перемещаться внутри кристаллической решетки. При этом атом мышьяка, потерявший электрон, превращается в положительный ион (As+), как показано на [**Рис.** **1.26**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page133) (b). Электрон имеет *отрицательный заряд (negative charge)*,поэтому мышьяк принятоназывать примесью *n-типа* *(n-type dopant)*. Если же в качестве примеси используется элемент III атомной группы, например, бор (B), то в каждом из атомов примеси будет не хватать одного электрона, как показано на [**Рис.** **1.26**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page133) (c). Отсутствующий электрон называют *дыркой* *(hole)*.Электрон из соседнего атома кремния может перейти к атомубора и заполнить недостающую связь. При этом, атом бора, получивший дополнительный электрон, превращается в отрицательный ион (B−), а в атоме кремния возникает дырка. Таким образом, дырка может мигрировать в кристаллической решетке подобно электрону. Дырка – это всего лишь отсутствие отрицательного заряда, но она ведет себя в полупроводнике как положительно заряженная частица.

Именно поэтому бор называют примесью *p-типа* *(p-type* *dopant)*.Поскольку проводимость кремния может меняться на порядки в зависимости от концентрации примеси, кремний называют *полупроводником (semiconductor)*.



**Рис. 1.26 Кремниевая решетка и атомы примесей**

**1.7.3 Конденсаторы**

*Конденсатор (capacitor)* состоит из двух проводников,отделенных друг от друга изолятором. Если к одному из проводников приложить напряжение V, то через некоторое время этот проводник накопит электрический заряд Q, а другой проводник накопит противоположный электрический заряд −Q. *Емкостью* *(capacitance)* C

конденсатора называется отношение заряда к приложенному напряжению C = Q/V. Емкость прямо пропорциональна размеру проводников и обратно пропорциональна расстоянию между ними.

Емкость – это очень важный параметр электрической схемы, поскольку зарядка или разрядка любого проводника требует времени и энергии. Более высокая емкость означает, что электрическая схема будет работать медленнее и потребует для своего функционирования больше энергии. К понятиям скорости и энергии мы будем постоянно возвращаться на протяжении всей этой книги.

**1.7.4 n-МОП и p-МОП-транзисторы**

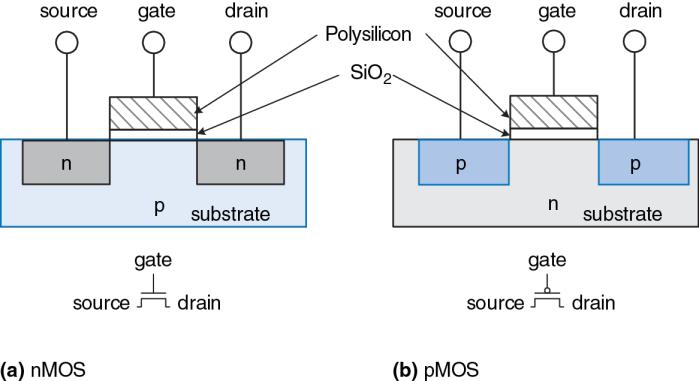
Полевой МОП-транзистор представляет собой «сэндвич» из нескольких слоёв проводящих и изолирующих материалов. «Фундамент», с которого начинается построение полевых МОП-транзисторов, – это тонкая круглая кремневая пластина *(wafer)* приблизительно от 15 см до 30 см в диаметре, в русскоязычной литературе называемая подложкой, вафлей или вэйфером. Производственный процесс начинается с пустой подложки. Этот процесс включает заранее определенную последовательность операций, в ходе которой примеси имплантируются в кремний, на подложке выращиваются тонкие пленки кремния и диоксида кремния и наносится слой металла. После каждой операции на подложку в качестве маски наносится определенный *рисунок (pattern)*,чтобы наносимый в ходе следующей операцииматериал оставался лишь в тех местах, где он необходим. Поскольку размеры одного транзистора – это доли микрона 2 , а вся подложка обрабатывается в ходе одного производственного процесса, когда одновременно производятся миллиарды транзисторов, себестоимость одного транзистора существенно снижается. После того, как все операции завершены, подложка нарезается на прямоугольные кристаллы, называемые в англоязычной литературе *chip* или *dice*,

1. 1 μm = 1 мкм = 10–6 м.
2. причем на каждом из этих прямоугольников размещаются тысячи, миллионы или даже миллиарды транзисторов. Каждый такой кристалл тестируется, а затем помещается в пластиковый или керамический *корпус-упаковку (package)* с металлическими *контактами (pins)* длятого, чтобы его можно было установить на монтажной плате.

Сэндвич полевого МОП-транзистора состоит из слоя проводника, называемого затвором *(gate),* наложенного на слой изолятора – диоксида кремния (SiO2), в свою очередь, наложенного на кремневую пластину, называемую подложкой. Изначально для изготовления затвора использовался тонкий слой металла, отсюда и название этого типа транзисторов – металл-оксид-полупроводник. В современных же технологических процессах в качестве материала затвора используется поликристаллический кремний, поскольку кремний не плавится в ходе последующей высокотемпературной обработки кристалла. Диоксид кремния – это хорошо известное всем нам стекло, и в полупроводниковой промышленности этот материал часто называют просто оксидом. Слои металл-оксид-полупроводника образуют конденсатор, в котором тонкий слой оксида (или окисла), называемого диэлектриком, изолирует металлическую пластину от полупроводниковой.

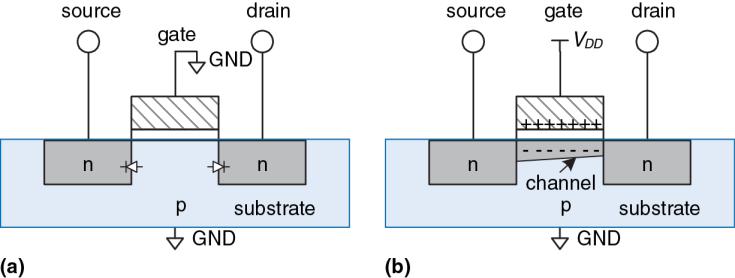
Существуют два вида полевых МОП-транзисторов: n-МОП и p-МОП (по английски n-MOS и p-MOS, что произносится как н-мосс и пи-мосс). На [**Рис.** **1.29**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page140) схематически показано сечение каждого из этих двух типов транзисторов так, как будто мы распилили кристалл и теперь смотрим на транзистор сбоку. В транзисторах n-типа, называемых n-МОП, области, где расположены полупроводниковые примеси n-типа – в свою очередь называемые истоком *(source)* и *стоком* *(drain)* –находятся рядом с затвором *(gate)*,причем вся эта структураразмещается на подложке p-типа. В транзисторах же p-МОП и исток, и сток – это области p-типа, размещенные на подложке n-типа.

Полевой МОП-транзистор ведет себя как переключатель, управляемый приложенным к нему напряжением. В таком транзисторе напряжение перехода создает электрическое поле, включающее или выключающее линию связи между источником и стоком. Термин *полевой транзистор* *(field effect transistor)* является прямым отражением принципа работытакого устройства. Знакомство с работой полупроводниковых устройств мы начнем с изучения n-МОП-транзистора.



**Рис. 1.29 n-МОП и p-МОП-транзисторы**

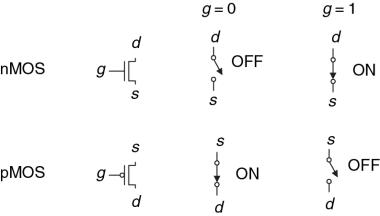
Подложка n-МОП транзистора обычно находится под напряжением земли GND, которое является минимальным напряжением в системе. Для начала рассмотрим случай, когда, как показано на [**Рис.** **1.30**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page143) (a), напряжение на затворе также равно 0 В. Диоды между истоком или стоком и подложкой находятся в состоянии, называемым обратным смещением *(reverse bias)*, поскольку напряжение на истоке и стоке не является отрицательным. В результате этого канал для движения тока между истоком и стоком остается закрытым, а транзистор выключенным. Теперь рассмотрим ситуацию, когда напряжение на затворе повышается до *VDD* – так, как показано на [**Рис.** **1.30**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page143) (b). Если приложить положительное напряжение к затвору (верхней пластине конденсатора), то это создает электрическое поле между затвором и подложкой, в результате в зону между истоком и стоком под слоем оксисла формируется избыток электронов. При достаточно высоком напряжении на нижней границе затвора накапливается настолько много электронов, что область с полупроводником p-типа превращается в область с полупроводником n-типа. Такая инвертированная область называется *каналом* *(channel)*. В этот момент в транзисторе образуется область проводимости от источника n-типа, через каналы n-типа к стоку n-типа, и через этот канал электроны могут беспрепятственно перемещаться от истока к стоку. Транзистор включен. Напряжение перехода, которое требуется для включения транзистора, называется пороговым значением напряжения (*threshold voltage*) *VT* и обычно составляет от 0,3 В до 0,7 В.



**Рис. 1.30 Работа n-МОП-транзистора**

Транзистор p-МОП работает с точностью до наоборот, как вы, возможно, уже догадались по наличию точки в обозначении этого типа транзистора на [**Рис.** **1.31**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page145) Подложка p-МОП-транзистора находится под напряжением *VDD*. Если затвор также находится под напряжением *VDD*, то p-МОП-транзистор выключен. Если же на затвор подается напряжение земли GND, проводимость канала инвертируется, превращаясь в проводимость p-типа, и транзистор включается.

К сожалению, полевые МОП-транзисторы в роли переключателя работают далеко не идеально. В частности, n-МОП-транзисторы хорошо передают 0, но плохо передают 1. Если переход n-МОП-транзистора находится под напряжением *VDD*, то напряжение на стоке будет колебаться между 0 и *VDD* − *VT*. Аналогичным же образом, p-МОП-транзисторы хорошо передают 1, но плохо передают 0. Однако, как мы увидим в дальнейшем, возможно построить хорошо работающий логический вентиль, используя только те режимы n-МОП- и p-МОП-транзисторов, в которых их работа близка к идеальной.

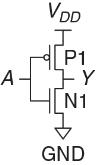
1. 

**Рис. 1.31 Модели переключения полевых МОП-транзисторов**

Для изготовления n-МОП-транзистора требуется подложка с проводимостью p-типа, а для изготовления p-МОП-транзисторов необходима подложка n-типа. Для того, чтобы разместить оба типа транзисторов на одном кристалле, производственный процесс, как правило, начинается с подложки p-типа, в который затем имплантируют области для размещения p-МОП-транзисторов n-типа, называемые *колодцами (wells)*.Такой процесс называется *Комплементарным МОП* или *КМОП* *(Complementary MOS* или *CMOS)*. В настоящее время КМОП-процесс используется для изготовления подавляющего большинства транзисторов и микросхем.

Подведем итог. КМОП-процесс позволяет разместить МОП-транзисторы n-типа и p-типа, показанные на [**Рис.** **1.31**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page145) на одном кристалле. Напряжение на затворе (g) управляет током между истоком (*s*) и стоком (d). Транзисторы n-МОП выключены, когда значение напряжения на переходе соответствует логическому 0, и включены, когда значение напряжения на переходе соответствует логическому 1. Транзисторы p-МОП, напротив, включены, когда значение напряжения на переходе соответствует логическому 0, и выключены, когда значение напряжения на переходе соответствует логическому 1.

# **39. Логический вентиль НЕ и другие на КМОП-транзисторах. Псевдо n-МОП-Логика Потребляемая мощность**

Схема на [**Рис.** **1.32**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page146) демонстрирует, как можно построить логический элемент НЕ, используя КМОП-транзисторы.

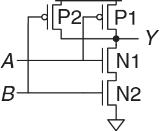
На этой схеме треугольник обозначает напряжение земли GND, а горизонтальная линия обозначает напряжение питания *VDD*.

На всех последующих схемах в этой книге мы не будем использовать буквенные обозначения V*DD* и GND.

**Рис. 1.32 Схема вентиля НЕ**

n-МОП-транзистор N1 включен между землей GND и выходным контактом *Y*. В свою очередь, p-МОП-транзистор P1 включен между напряжением питания *VDD* и выходным контактом *Y*. Напряжение на входном контакте *А* управляет переходами обоих транзисторов.

Если напряжение на *А* равно 0, то транзистор N1 выключен, транзистор P1 включен. При этом, напряжение на контакте Y равно напряжению питания *VDD*, а не земли, что соответствует логической единице. В этом случае говорят, что Y «подтянут» к единице (англ.: *pulled up)*.Включенный транзисторP1хорошо передает логическуюединицу (равную напряжению питания), то есть напряжение на контакте *Y* очень близко к *VDD*.Если же напряжение на контакте А равнологической единице, то транзистор N1 включен, а транзистор P1 выключен, и напряжение на контакте *Y* равно напряжению земли, что соответствует логическому нулю. В этом случае говорят, что *Y* «подтянут» к нулю (англ.: *pulled down)*. Включенный транзистор N1 хорошо передает логический ноль, то есть напряжение на контакте *Y* очень близко к GND. Проверка в таблице истинности на [**Рис.** **1.12**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page107) подтверждает, что мы действительно имеем дело с логическим вентилем НЕ.

**1.7.6 Другие логические вентили на КМОП-транзисторах**

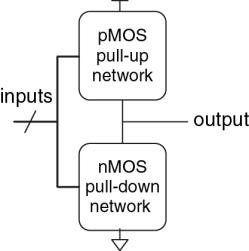
1. **Рис. 1.33 Схема вентиля И-НЕ с двумя входами**

На [**Рис.** **1.33**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page148) показана схема для построения с помощью МОП-транзисторов логического элемента И-НЕ с двумя входными контактами. На электронных схемах принято, что если нет никаких дополнительных замечаний или обозначений, то подразумевается, что две линии соединяются друг с другом в том случае, если одна из линий заканчивается в точке пересечения (пересечение в форме буквы Т). Если же обе линии продолжаются точкой пересечения, то для обозначения контакта этих двух линий в точке пересечения ставится точка. Если точка отсутствует, то это означает, что линии не пересекаются, и одна из линий проходит над другой. На [**Рис.** **1.33**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page148) n-МОП-транзисторы N1 и N2 соединены последовательно. Причем, чтобы замкнуть выходной контакт на землю GND – то есть понизить логический уровень (*pull down*), оба этих транзистора должны быть включены. В то время как p-МОП-транзисторы P1 и P2 соединены параллельно, и только один из них должен быть включен, чтобы соединить выходной контакт с напряжением питания *VDD* – то есть повысить логический уровень (*pull up*). В [**Табл.** **1.6**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page149) перечислены все возможные состояния для части схемы, понижающей логический уровень (*pull-down network*), для части схемы, повышающей логический уровень (*pull-up network*), и для выхода. Из [**Табл.** **1.6**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page149) видно, что электрическая схема, показанная на [**Рис.** **1.33**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page148) действительно работает как логический вентиль И-НЕ. Например, если A равно 1 и B равно 0, то транзистор N1 включен, однако транзистор N2 выключен и блокирует связь контакта *Y* с напряжением земли GND. При этом транзистор P1 выключен, а транзистор P2 включен и соединяет напряжение питания *VDD* с контактом *Y*. То есть, на контакте *Y* мы имеем 1.

**Табл. 1.6 Работа вентиля И-НЕ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *A* | *B* | Схема понижения | Схема повышения | *Y* |
| логического уровня | логического уровня |
|  |  |  |
| 0 | 0 | ВЫКЛ. | ВКЛ. | 1 |
|  |  |  |  |  |
| 0 | 1 | ВЫКЛ. | ВКЛ. | 1 |
| 1 | 0 | ВЫКЛ. | ВКЛ. | 1 |
|  |  |  |  |  |
| 1 | 1 | ВКЛ. | ВЫКЛ. | 0 |
|  |  |  |  |  |

[**Рис. 1.34**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page150) вобобщенном виде показывает блоки,необходимые дляпостроения любого инвертированного логического вентиля, такого как НЕ, И-НЕ, ИЛИ-НЕ.



**Рис. 1.34 Общая форма инвертирующего логического вентиля**

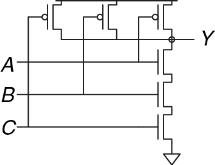
Транзисторы n-МОП хорошо передают 0, поэтому схема, понижающая логический уровень (*pull-down network*), составленная из таких транзисторов, помещается между выходным контактом и землей GND для передачи 0 на выход. Транзисторы p-МОП хорошо передают 1, поэтому схема, повышающая логический уровень (*pull-up network*), составленная из таких транзисторов, помещается между выходным контактом и напряжением питания *VDD* для передачи 1 на выход. Понижающая и повышающая схемы могут состоять из транзисторов, соединенных как параллельно, так и последовательно. Причем при параллельном соединении транзисторов вся схема включена, если включен хотя бы один из транзисторов. При последовательном соединении схема включена, только если оба транзистора включены.

Косая черта на входной линии указывает на то, что этот логический элемент имеет несколько входов.

Если и понижающую, и повышающую части схемы включить одновременно, то во всей схеме возникнет короткое замыкание между напряжением питания *VDD* и землей GND. Сигнал на выходном контакте может оказаться в запретной зоне, а транзисторы, потребляющие при этом большое количество энергии, могут перегореть. С другой стороны, если и понижающую и повышающую части схемы одновременно выключить, то выходной сигнал будет отключен и от *VDD,* и от GND. В этом случае говорят, что выходной сигнал *плавает* *(floats)*. Его значение, так же как и в случае одновременно включенных схем, не определено. Наличие плавающего сигнала на выходе системы обычно нежелательно, но в **разделе** [**2.6**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page254) мы рассмотрим, как разработчик может использовать такие сигналы.

1. правильно функционирующем логическом вентиле в любой момент времени одна из схем должна быть включена, а другая выключена, и напряжение на выходе должно быть или высоким (*VDD*), или низким (GND). Ни короткое замыкание, ни плавающее значение сигнала не допускается. Чтобы гарантировать это условие, пользуются правилом

*дополнения проводимости (conduction complements)*.Еслиn-МОП-транзисторы в какой-либо цепи соединены последовательно, то p-МОП-транзисторы в этой же цепи должно быть соединены параллельно. Если же n-МОП-транзисторы соединены параллельно, то p-МОП-транзисторы должны соединяться последовательно.



|  |
| --- |
|  |

**Схема вентиля И-НЕ с 3 входами**

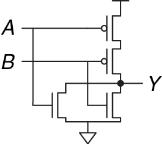


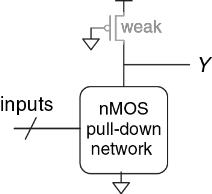
Схема вентиля ИЛИ-НЕ с 2 входами



Схема вентиля И с 2 входами

* + 1. **Псевдо n-МОП-Логика**

Построенный по технологии КМОП логический вентиль ИЛИ-НЕ, 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.

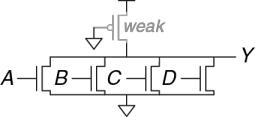


**Рис. 1.39 Обобщенный псевдо n-МОП-вентиль**

Как показано на [**Рис.** **1.39**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page156) при использовании *псевдо* *n-МОП-логики* *(pseudo-nMOS logic)*,или просто *псевдо-логики*,медленный стек изp-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**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page158) 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.



**Рис. 1.40 Псевдо n-МОП-вентиль ИЛИ-НЕ с четырьмя входами**

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.

**1.8** **ПОТРЕБЛЯЕМАЯ МОЩНОСТЬ**

*Потребляемая* мощность–это количество энергии,потребляемойсистемой 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.

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.В процессе разрядки конденсатор не потребляет энергию от источника питания.

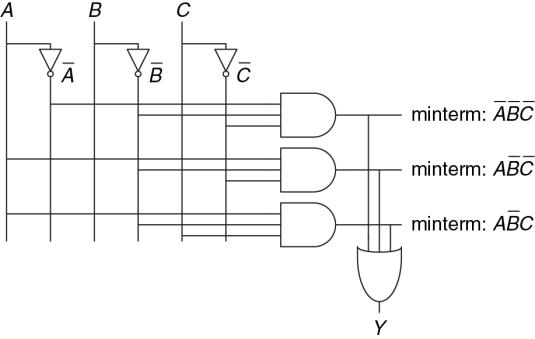
Утечка тока в системе происходит, даже если система находится в состоянии покоя. У некоторых типов электронных схем, таких как псевдо n-МОП-логика, рассмотренная в **разделе** [**1.7.8**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page156) 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

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

**ОТ ЛОГИКИ К ЛОГИЧЕСКИМ ЭЛЕМЕНТАМ**

Принципиальная схема – это изображение цифровой схемы, показывающее элементы и соединяющие их проводники. Например, схема на [**Рис. 2.23**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page236) показывает возможную аппаратную реализацию нашей любимой логической функции (**уравнение** [**(2.3)**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page215)):

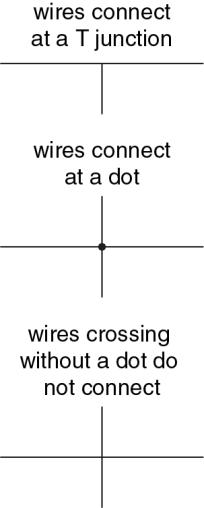
*Y* = *A¯B¯C¯* + *AB¯C¯* + *AB¯C*



**Рис. 2.23 Схема *Y* =*A¯ B¯ C¯* + *AB¯ C¯* + *AB¯* *C***

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

1. Входы изображаются на левой (или верхней) части схемы;
2. Выходы изображаются на правой (или нижней) части схемы;
3. Всегда, когда это возможно, элементы необходимо изображать слева направо;
4. 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);
5. Проводники всегда должны соединяться в виде буквы «Т»;



1. Точка в месте пересечения проводников обозначает их соединение;
2. Проводники, пересекающиеся без точки, не имеют соединения друг с другом.

Три последних правила показаны на [**Рис.** **2.24**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page238)

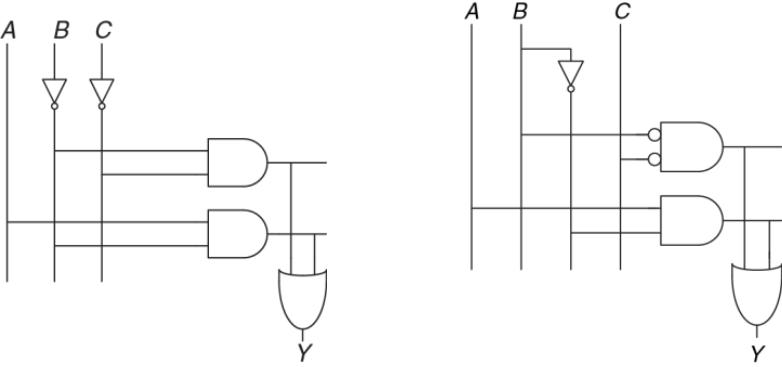
Любое булево уравнение в дизъюнктивной форме может быть изображено в виде принципиальной схемы с использованием систематического подхода, как показано на [**Рис. 2.23**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page236) Сначала нарисуйте вертикальные проводники для входов. Поместите инверторы на соседних вертикальных линиях для получения комплементарных входов, если это необходимо. Нарисуйте горизонтальные линии, ведущие к элементам И, для каждого минтерма. Затем для каждого выхода нарисуйте элемент ИЛИ, соединенный с минтермом, соответствующим этому выходу. Такой стиль изображения называется программируемой логической матрицей (ПЛМ, PLA), потому что инверторы, элементы И и элементы ИЛИ систематически объединены в массивы. Программируемые логические матрицы будут рассмотрены в **разделе** [**5.6**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page747)

Мы даже можем ещё уменьшить количество элементов (пусть хотя бы

на один инвертор), если воспользуемся преимуществом инвертирующих логических элементов. Заметьте, что *B¯C¯* – это элемент с инвертированными входами. На [**Рис.** **2.26**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page240) показана схема, которая использует эту оптимизацию для исключения инвертора на входе *С*. Вспомните, что согласно теореме де Моргана логический элемент И с

инвертированными входами эквивалентен элементу ИЛИ-НЕ.

1. В зависимости от технологии реализации, использование наименьшего числа элементов или использование элементов определенного типа взамен других может быть выгоднее. Например, в технологии КМОП элементы И-НЕ и ИЛИ-НЕ более предпочтительны, чем И или ИЛИ. многих схем имеется несколько выходов, каждый из которых вычисляет независимые булевы функции для входов. Мы можем записать отдельные таблицы истинности для каждого выхода, но часто удобно записать все выходы в одну таблицу истинности и начертить одну схему для всех выходов.



**Рис. 2.25 Схема реализации функции *Y* = *B¯ C¯* + *AB¯* Рис. 2.26 Схема, использующая меньше элементов**

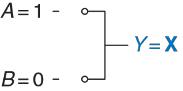
Символ «X» используется не только для обозначения переменных, чье состояние нам безразлично, но и для обозначения недопустимых состояний сигналов при симуляции логических схем (см. **Раздел** [**2.6.1**)](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page254). Старайтесь понять из контекста, о каком варианте использования идет речь. Чтобы избежать такой двусмысленности, некоторые авторы используют символы «D» или «?» для обозначения сигналов, состояние которых нам безразлично.

**2.6** **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.

**2.6.1 Недопустимое значение: Х**

Символ X обозначает неизвестное логическое значение или недопустимое значение физического напряжения в соединении, не соответствующее уровням логических 0 и 1. Это обычно происходит, если к соединению подключены выходы других элементов схемы, выдающие значения 0 и 1 одновременно. На [**Рис.** **2.39**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page254) показан такой случай, когда выход *Y* подключен к элементам, имеющим на выходе ВЫСОКИЙ и НИЗКИЙ уровни.



**Рис. 2.39 Схема с недопустимым значением на выходе**

Эта ситуация, называемая состязанием или конфликтом (contention), считается ошибкой, и её необходимо избегать. Реальное (физическое) напряжение на выходе с конфликтом может быть где-то между нулем и напряжением питания, в зависимости от соотношения мощностей элементов, выдающих в цепь ВЫСОКОЕ и НИЗКОЕ напряжения. Часто, но не всегда, значение напряжения оказывается в «запрещенной» зоне. Состязание также может стать причиной повышенного потребления энергии конфликтующими элементами, в результате чего схема нагревается и может быть повреждена.

Значение X также иногда используется программами моделирования для обозначения неинициализированного значения. Например, если вы забыли определить входное значение, симулятор присвоит ему значение X для того, чтобы предупредить вас о проблеме.

Как уже упоминалось в **разделе** [**2.4**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page236) разработчики цифровых схем также используют символ X для обозначения в таблицах истинности безразличных переменных, от которых не зависит состояние выходов. Не путайте эти два смысла. Когда X появляется в таблицах истинности, он показывает, что значение этой переменной может быть и нулем, и единицей. Когда X появляется в схеме, это означает, что цепь имеет неизвестное или запрещенное значение.

**2.6.2 Третье состояние: Z**

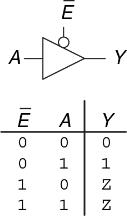
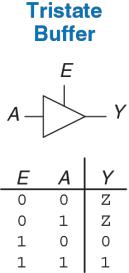
Символ Z указывает, что напряжение в цепи не определяется ни источником ВЫСОКОГО, ни источником НИЗКОГО напряжения. Говорят, что такая цепь отключена, находится в состоянии высокого импеданса или в третьем состоянии. Типично неправильное представление – это что неподключенная, или плавающая цепь имеет значение логического

1. В реальности логическое состояние неподключенной цепи может быть как 0, так и 1, а напряжение на ней может принять некое промежуточное значение в зависимости от истории изменения состояния системы. Неподключенная цепь не обязательно означает наличие ошибки в схеме. Например, какой-нибудь другой элемент схемы может задать цепи допустимый логический уровень именно в тот момент, когда эта цепь влияет на работу схемы.

Один из распространенных способов получить неопределенное значение – это забыть подключить вход схемы к источнику напряжения логического уровня или предположить, что неподключенный вход – то же самое, что вход со значением 0. Эта ошибка может привести к тому, что поведение цепи будет хаотичным, так как неопределенные значения на входе могут случайно меняться из 0 в 1. Действительно, касания схемы может быть достаточно, чтобы привести к изменению из-за слабого статического электричества тела. Мы видели схему, которая корректно работала, только до тех пор, пока студент держал палец на микросхеме.

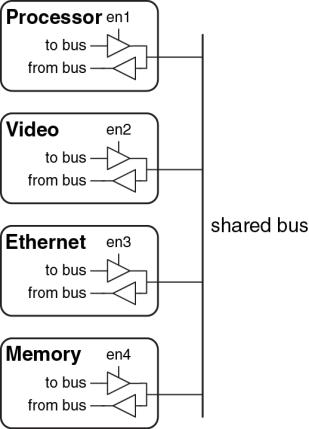
Буфер с тремя состояниями, показанный на [**Рис.** **2.40**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page258) имеет три возможных выходных значения: ВЫСОКОЕ (1), НИЗКОЕ (0) и отключенное или плавающее (Z) состояние (прим. переводчика: именно поэтому плавающее состояние называют третьим). Буфер с тремя состояниями имеет вход A, выход Y и сигнал управления E. Когда сигнал разрешения (управления) имеет значение ИСТИНА, буфер с тремя состояниями работает как простой буфер, передавая входное значение на выход. Когда сигнал управления имеет значение ЛОЖЬ, выход буфера переключается в третье состояние и становится плавающим (Z). Буфер с тремя состояниями на [**Рис.** **2.40**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page258) имеет активный высокий уровень. Это значит, что когда сигнал разрешения ВЫСОКИЙ (1), передача разрешена.

На [**Рис.** **2.41**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page258) показан Буфер с тремя состояниями с активным низким уровнем. Когда сигнал управления НИЗКИЙ (0), передача разрешена. Мы видим, что сигнал имеет активный низкий уровень из-за отрицания, поставленного на его входной цепи. Мы часто обозначаем вход с активным низким уровнем, рисуя черточку (символ отрицания) над его именем (*E¯*), или добавляя букву "b" или "bar" после имени, *Eb* или *Ebar*.



**Рис. 2.40 Буфер с тремя состояниями Рис. 2.41 Буфер с тремя состояниями с активным низким уровнем**

Буферы с третьим состоянием обычно используются в шинах, соединяющих несколько микросхем. Например, микропроцессор, видеоконтроллер и Ethernet-контроллер могут нуждаться во взаимодействии с подсистемой памяти в персональном компьютере. Каждая микросхема может подключаться к общей шине памяти, используя буферы с третьим состоянием, как показано на [**Рис.** **2.42**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page259) При этом только одна микросхема имеет право выставить свой сигнал разрешения, чтобы выдать значение на шину. Выходы других микросхем должны находиться в третьем состоянии, чтобы не стать причиной коллизии с микросхемой, осуществляющей обмен данными с памятью. Однако, любая микросхема может читать информацию с общей шины в любое время. Такие шины на основе буферов с тремя состояниями когда-то были очень распространенными. Однако, в современных компьютерах высочайшие скорости возможны только при соединении микросхем друг с другом напрямую (point-to-point), а не с помощью общей шины.



**Рис. 2.42 Шина с третьим состоянием, соединяющая несколько микросхем**

# **42. Базовые комбинационные блоки. Мультиплексоры. Логика на мультиплексорах. Дешифраторы**

**2.8** **БАЗОВЫЕ КОМБИНАЦИОННЫЕ БЛОКИ**

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». Мы уже изучили три таких блока: полный сумматор (см. **раздел** [**2.1**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page201)), схемы приоритета (см. **раздел** [**2.4**)](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page236) и дешифратор семисегментного индикатора (см. **раздел** [**2.7**)](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page260). Этот раздел представляет два типа блоков, еще более часто используемых при проектировании: мультиплексоры и дешифраторы. В **главе** [**5**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page655) будет рассказано и о других комбинационных «строительных блоках».

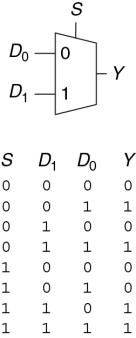
**2.8.1 Мультиплексоры**

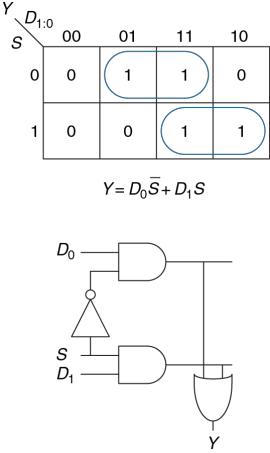
Мультиплексоры являются одними из наиболее часто используемых комбинационных схем. Они позволяют выбрать одно выходное значение из нескольких входных в зависимости от значения сигнала выбора.

**Двухвходовой мультиплексор (2:1)**

На [**Рис.** **2.54**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page278) показано условное графическое обозначение и таблица истинности для двухвходового мультиплексора (2:1) с двумя входами данных *D*0 и *D*1, входом выбора *S* и одним выходом *Y*. Мультиплексор передает на выход один из двух входных сигналов данных, основываясь на сигнале выбора: если *S* = 0, выход *Y* = *D*0, и если *S* = 1, то выход *Y* = *D*1. *S* также называют управляющим сигналом, так как он управляет поведением мультиплексора.

Двухвходовой мультиплексор может быть построен с использованием дизъюнкции конъюнкций (суммы произведений), как показано на [**Рис. 2.55**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page278)Логическоевыражение для него может быть получено спомощью карт Карно или составлено на основе описания (*Y* = 1 если *S* = 0И *D*0= 1ИЛИ если *S* =1И *D*1= 1).

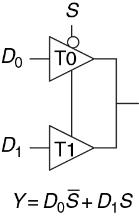
Мультиплексор также может быть построен на буферах с третьим состоянием, как показано на [**Рис.** **2.56**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page279) Сигналы разрешения буферов с третьим состоянием организованы так, что все время активен только один буфер. Когда *S* = 0, то включен только элемент T0, позволяющий сигналу *D*0 передаваться на выход *Y*. Когда *S* = 1, то активен только элемент T1, передавая на выход сигнал *D*1.



**Рис. 2.54 Условное обозначение и** **Рис. 2.55 Реализация двухвходового**

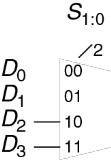
**таблица истинности** **мультиплексора**

**двухвходового мультиплексора** **с использованием**

**двухуровневой логики**

**Рис. 2.56 Мультиплексор на буферах с тремя состояниями**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Многовходовые мультиплексоры** | | |  |  |  |
|  | Четырехвходовой мультиплексор | | | | (4:1) имеет |
|  | четыре входа данных и один выход, как показано на | | | | |
|  | [**Рис. 2.57**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page280)Длявыбора одного из четырех входов | | | | |
|  | данных | требуется | двухразрядный | | управляющий |
|  | сигнал. Четырехвходовой мультиплексор может | | | | |
|  | быть построен с использованием дизъюнкции | | | | |
| **Рис. 2.57** | конъюнкций (суммы | | произведений), буферов с | | |
| тремя | состояниями | | или | двухвходовых |
| **Четырехвходовой** |  |  |  |  |  |



|  |  |
| --- | --- |
| **мультиплексор** | мультиплексоров, как показано на [**Рис.** **2.58**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page281) |
|  |

Конъюнкции, подключенные к сигналам разрешения работы буферов с тремя состояниями, могут быть построены с использованием элементов И и инверторов. Они также могут быть сформированы дешифратором.

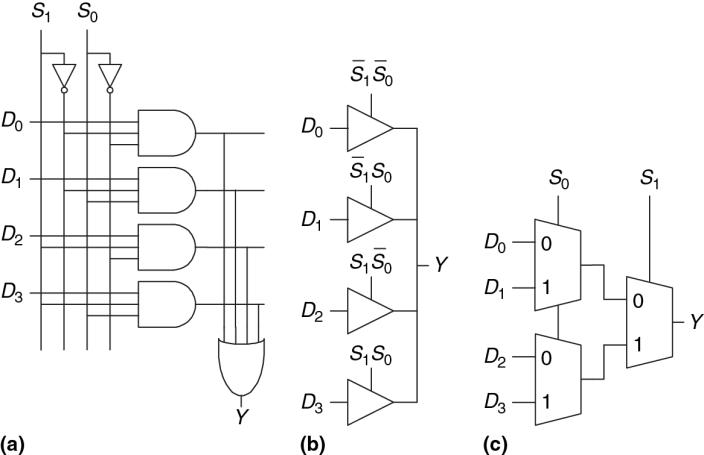
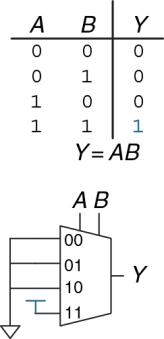
Мультиплексоры с большим числом входов, например восьмивходовые или шестнадцативходовые, могут быть построены простым масштабированием методов, показанных на [**Рис.** **2.58**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page281) В общем случае, мультиплексор *N*:1 требует log2*N* управляющих сигналов. Выбор наилучшей реализации, как и прежде, зависит от используемой технологии.

Рис. 2.58 Реализация четырехвходового мультиплексора: двухуровневая логика (a), буфера с тремя состояниями (b), иерархическая (c)

**Логика на мультиплексорах**

Мультиплексоры могут использоваться как таблицы преобразования (lookup tables) для выполнения логических функций. На [**Рис.** **2.59**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page282) показан четырехвходовой мультиплексор, используемый для реализации двухвходового элемента И.

 **Рис. 2.59 Получение двухвходового элемента И из четырехвходового мультиплексора**

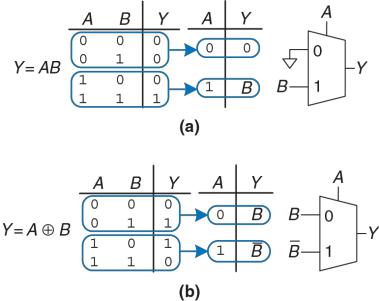
Входы *A* и *B* служат управляющими линиями. Входы данных мультиплексора подключены к 0 и 1 согласно соответствующей строке таблицы истинности. Вообще, 2*N*-входовой мультиплексор можно запрограммировать для выполнения любой *N*-входовой логической функции,используя0и1

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 2*N*-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.

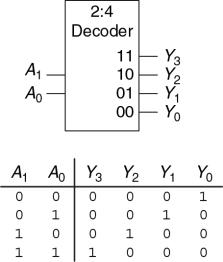
Для иллюстрации этого принципа на [**Рис.** **2.60**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page283) показаны функции двухвходовых элементов И и ИСКЛЮЧАЮЩЕЕ ИЛИ, реализованных на двухвходовых мультиплексорах. Мы начали с обычной 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 таблицуподстановки согласно этой новой уменьшенной таблице истинности.



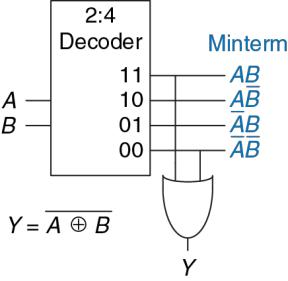
**Рис. 2.60 Реализация логических функций на мультиплексорах**

**2.8.2 Дешифраторы**

1. В общем случае у дешифратора имеется *N* входов и 2*N* выходов. Он выдает единицу строго на один из выходов в зависимости от набора входных значений. На [**Рис.** **2.63**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page286) показан дешифратор 2:4. Когда *A*[1:0] = 00, *Y*0 = 1. Когда *A*[1:0] = 01, *Y*1 = 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.



**Рис. 2.63 Дешифратор 2:4**

**Логика на дешифраторах**

**Рис. 2.65 Реализация логической функции на дешифраторе**

Дешифратор может комбинироваться с элементами ИЛИ для построения логических

функций. На [**Рис.** **2.65**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page288) показана двухвходовая функция ИСКЛЮЧАЮЩЕЕ

ИЛИ-НЕ (XNOR), использующая дешифратор 2:4 и один элемент ИЛИ. Поскольку каждый выход дешифратора представляет одиночный минтерм, функция построена как логическое ИЛИ всех минтермов в этой функции. На [**Рис.** **2.65**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page288) показана функция.

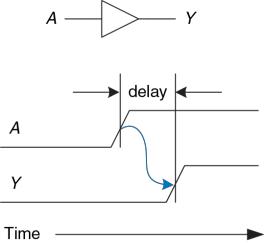
При использовании дешифраторов для реализации логических функций, проще всего выразить функцию таблицей истинности или записать ее в дизъюнктивной нормальной форме. *N*-входовая функция,имеющая *M* единиц в таблице истинности,можетбыть построена с использованием *N*:2*N* дешифратора и *M*-входового элемента ИЛИ, подключенным ко всем минтермам, содержащим единицу в таблице истинности.

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

**2.9** **ВРЕМЕННЫЕ ХАРАКТЕРИСТИКИ**

1. В предыдущих разделах мы концентрировались в первую очередь на работе схемы, в идеале использующей наименьшее число элементов. Однако, 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**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page290) 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.

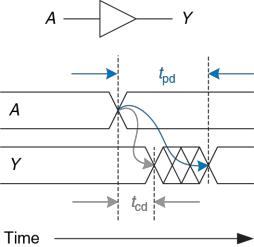


**Рис. 2.66 Задержка схемы**

**2.9.1 Задержка распространения и задержка реакции**

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**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page291) 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*.



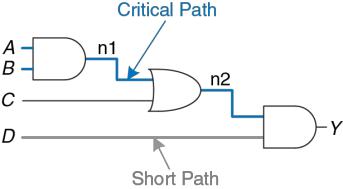
**Рис. 2.67 Задержка распространения и задержка реакции**

Основные причины задержек в схемах заключаются во времени, требуемом для перезарядки емкостей цепи, а так же в конечной скорости распространения электромагнитных волн в среде. Величины *tpd* и *tcd* могут различаться по многим причинам,включающим в себя:

1. Разные задержки нарастания и спада сигнала;
2. Несколько входов и выходов, одни из которых быстрее чем другие;
3. Замедление работы схемы при повышении температуры и ускорение при охлаждении.

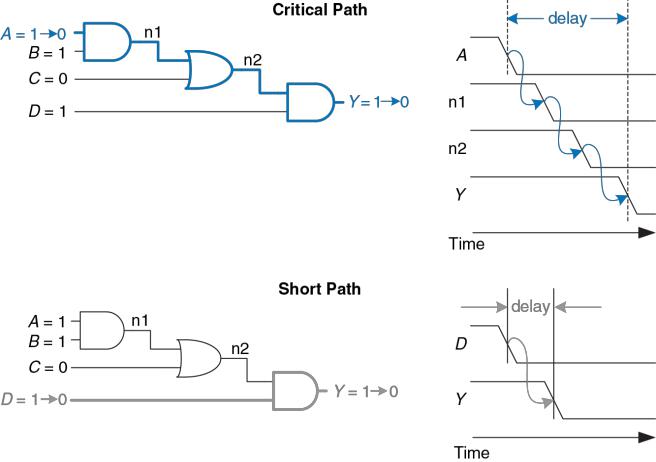
Вычисление *tpd* и *tcd* требует вникания в нижние уровни абстракций, что выходит за рамки этой книги. Однако, производители обычно предоставляют документацию со спецификацией этих задержек для каждого элемента.

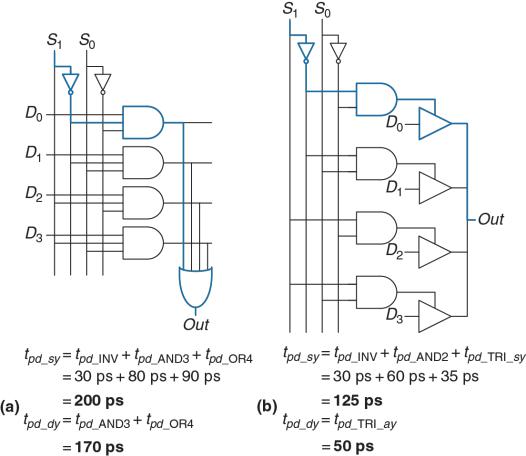
Наряду с уже перечисленными факторами, задержки распространения реакции также определяются *путем*, который проходит сигнал от входа до выхода. На [**Рис.** **2.68**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page293) показана четырехвходовая схема. *Критический путь* (critical path),выделенный синим–это путь от входа *A* или *B* до выхода *Y*.Он соответствует цепи с наибольшей задержкой и является самым медленным, поскольку входному сигналу нужно пройти три элемента до выхода. Этот путь критический потому, что он ограничивает скорость, с которой работает схема. Самый короткий путь схеме, показанный серым – путь от входа *D* до выхода *Y*. Это кратчайший и, следовательно, самый быстрый путь в схеме, т.к. входному сигналу до выхода нужно пройти только через один элемент.



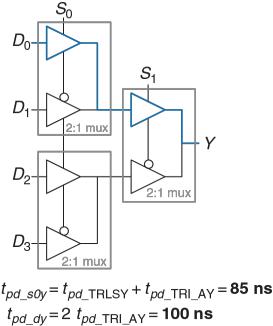
**Рис. 2.68 Кратчайшая цепь и цепь с наибольшей задержкой**

Задержка распространения комбинационной схемы – это сумма задержек распространения всех элементов в критическом пути. Задержка реакции – сумма задержек реакции всех элементов в кратчайшем пути. Эти задержки показаны на [**Рис.** **2.69**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page294)

1. 

**Рис. 2.69 Временные диаграммы для кратчайшей цепи и цепи с наибольшей задержкой**

**Рис. 2.73 Задержки распространения в четырехвходовом мультиплексоре:**

**двухуровневая логика (a), буфера с тремя состояниями (b)**

**Рис. 2.74 Задержки распространения в четырехвходовом мультиплексоре, построенном из двухвходовых**

**Табл. 2.7 Временные характеристики элементов в схемах мультиплексоров**

|  |  |  |
| --- | --- | --- |
|  | **Элемент** | ***tpd* (пс)** |
|  | НЕ | 30 |
|  |  |  |
|  | Двухвходовой И | 60 |
|  | Трехвходовой И | 80 |
|  |  |  |
|  | Четырехвходовой ИЛИ | 90 |
|  | Буфер с тремя состояниями (от *A* до *Y*) | 50 |
|  |  |  |
|  | Буфер с тремя состояниями (от *E* до *Y*) | 35 |

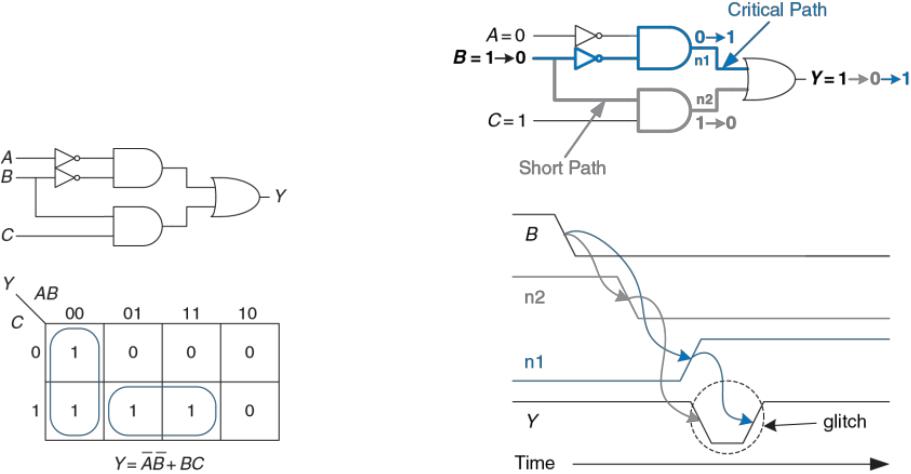
**2.9.2 Импульсные помехи**

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 невызывает проблем, важно понимать, что он есть, и уметь распознавать его на временных диаграммах. На [**Рис.** **2.75**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page301) 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**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page301) 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.

Как показано на временных диаграммах на [**Рис.** **2.76**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page301) *Y* начинается с 1 и заканчивается 1, но на короткое время переключается в 0.



**Рис. 2.75 Схема, подверженная импульсным помехам Рис. 2.76 Временная диаграмма импульсной помехи**

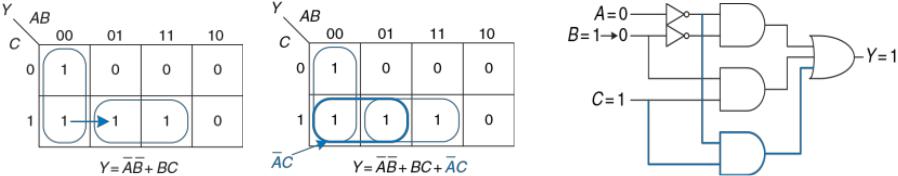
До тех пор, пока мы выдерживаем интервал равный времени задержки распространения, прежде чем использовать значение с выхода, импульсная помеха не представляет проблемы, потому что выход в конце концов установится в правильное значение.

При желании мы можем избежать этого импульса добавлением дополнительного элемента в схему. Это проще понять в термах карты Карно.

На [**Рис.** **2.77**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page303) показано, как изменение входа *B* при переходе из *ABC* =001в *ABC* = 011приводит к переходу от одной первичнойимпликанты к другой. Переход через границу двух первичных импликант в карте Карно свидетельствует о возможном появлении импульсной помехи.

Как мы видели на временных диаграммах на [**Рис.** **2.76**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page301) если схема реализации одной первичной импликанты выключается до того, как может включиться схема другой первичной импликанты, возникнет импульсная помеха. Чтобы исправить это, мы добавили другую цепь, которая охватывает границу первичных импликант, как показано на [**Рис.** **2.78**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page303) Вы могли бы узнать в этом теорему согласованности, где добавленный терм *A¯C* — это согласованный или избыточный терм.

На [**Рис.** **2.79**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page303) показана схема, устойчивая к паразитным импульсам. Добавленный элемент И выделен синим. Сейчас переключение *B*, когда *A* = 0 и *C* = 1, не вызывает паразитного импульса на выходе, поскольку синий элемент И формирует на выходе 1 во время этого перехода.



**Рис. 2.77 Переход от одной импликанты другой**

**Рис. 2.78 Карта Карно без импульсных помех**

**Рис. 2.79 Схема без импульсных помех**

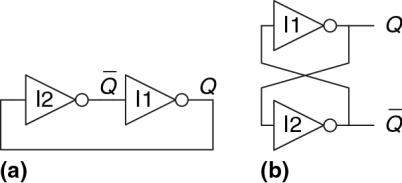
1. В общем случае, паразитный импульс может возникать, когда одна переменная пересекает границу между двумя первичными импликантами в карте Карно. Мы можем устранить эти импульсы добавлением избыточных импликант в карту Карно, чтобы покрыть эти границы. Естественно, это будет сделано ценой дополнительных аппаратных затрат.

Однако одновременное переключение нескольких входов также может стать причиной паразитных импульсов. Эти импульсы не могут быть исправлены дополнительными элементами в схеме. Поскольку подавляющее большинство интересующих нас систем имеют одновременные (или почти одновременные) переключения множества входов, возникновение паразитных импульсов в них неизбежно. Хотя мы показали, как устранить один вид импульсных помех, смысл дискуссии о паразитных импульсах не в том, чтобы устранять их, а в том чтобы знать, что они есть. Это особенно важно, при анализе временных диаграмм в симуляторе или на экране осциллографа.

**43. Проектирование последовательностной логики. ЗАЩЕЛКИ И ТРИГГЕРЫ. RS-триггер. D-защелка. D-Триггер. Регистр.**

**3.2** **ЗАЩЕЛКИ И ТРИГГЕРЫ**

Основным блоком для построения памяти является бистабильная ячейка – элемент с двумя устойчивыми состояниями. На [**Рис.** **3.1**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page330) **(а)** показана простая бистабильная ячейка, состоящая из пары инверторов, замкнутых в кольцо. Эту схему можно перерисовать так, чтобы рисунок выглядел симметрично [(**Рис.** **3.1**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page330) (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*.



**Рис. 3.1 Перекрестно соединенные инверторы**



Выход последовательностной схемы принято обозначать буквой *Q* аналогично тому, что выход комбинационной схемы принято обозначать буквой *Y*.

Рассмотрим два случая: *Q*=0 и *Q*=1

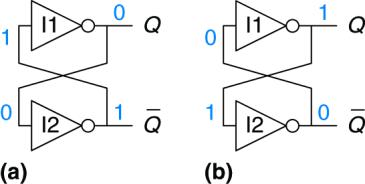
1. Случай I: *Q*=0

Как показано на [**Рис.** **3.2**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page331) (a), на вход I2 поступает сигнал *Q* = 0. I2 инвертирует сигнал и подает на вход I1 сигнал *Q¯* = 1. Соответственно, на выходе I1 – логический 0. В рассмотренном случае схема находится в *стабильном состоянии*.

1. Случай II: *Q*=1

Как показано на [**Рис.** **3.2**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page331) (b), на вход I2 поступает 1 (*Q*). I2 инвертирует сигнал и подает на вход I1 0 *Q¯* . Соответственно, на выходе I1 – логическая 1. В этом случае схема также находится в стабильном состоянии.

Так как инверторы, включенные перекрестно, имеют два стабильных состояния *Q* = 0 и *Q* = 1, то говорят, что схема бистабильна. У схемы есть и третье состояние, когда оба выхода находятся в состоянии между 0 и 1. Такое состояние называется *метастабильным*, и оно будет рассмотрено в **разделе** [**3.5.4**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page429)



**Рис. 3.2 Бистабильный режим перекрестно соединенных инверторов**

Элемент с *N* стабильными состояниями хранит log2*N* бит информации. Таким образом, бистабильная ячейка хранит 1 бит. Состояние перекрестно включенных инверторов содержится в одной переменной состояния *Q*. Значение *Q* сообщает нам всю информацию о прошлом, необходимую для определения будущего поведения схемы.

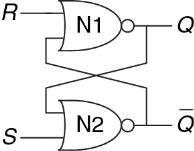
1. В частности, если *Q* =1, то оно и будет 0 всегда, а если *Q* = 1, то оно и останется 1. У схемы есть еще один выход – *Q¯*. Но *Q¯* не содержит никакой дополнительной информации.

При включении питания исходное состояние последовательностной схемы неизвестно и обычно непредсказуемо. Оно может быть различным всякий раз, когда схему включают.

Несмотря на то, что перекрестно включенные инверторы могут хранить бит информации, они не используются на практике, так как у схемы нет входов, с помощью которых пользователь мог бы контролировать ее состояние. Однако, другие элементы, такие как защелки и триггеры, имеют входы, которые позволяют управлять переменной состояния. Эти схемы рассматриваются в оставшейся части раздела.

**3.2.1 RS-триггер**

Одной из простейших последовательностных схем является *RS-триггер*, (от англ.ResetиSet),состоящий,как показано на[**Рис.****3.3**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page333)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*.



**Рис. 3.3** **RS-триггер (защелка)**

Для того чтобы понять, как работает неизвестная цепь, обычно строят таблицу истинности. Вспомним, что на выходе элемента ИЛИ-НЕ появляется логический нуль, если на какой-либо из его входов подана логическая единица. Рассмотрим четыре возможных комбинации *R* и *S*:

1. *Случай I:* *R*=1, *S*=0

На входе N1 как минимум одна единица – вход *R*, следовательно, выход *Q*=0. Оба входа N2 – в состоянии логического нуля (*Q*=0 и *S*=0),поэтому выход *Q¯* =1.

1. *Случай II*: *R*=0, *S*=1

На вход N1 поступает 0 и *Q¯* . Так как мы еще не знаем значения *Q¯* , мы не можем определить значение *Q*. На вход N2 поступает как минимум одна единица *S*, поэтому на выходе *Q¯* нуль. Теперь можно вернуться к определению состояния выхода элемента N1. Мы знаем, что на обоих его входах 0, следовательно, *Q*=1.

1. *Случай III*: *R*=1, *S*=1

Как на входе N1, так и на входе N2 как минимум по одной единице (*R* и *S*), поэтому на выходе каждой защелки – логический 0. Следовательно, *Q*=0 и *Q¯* =0.

1. *Случай IV*: *R*=0, *S*=0

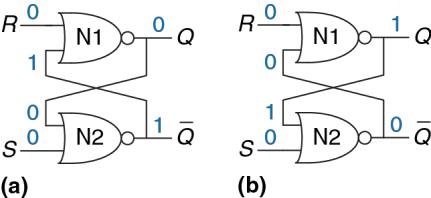
На вход N1 поступает 0 и *Q¯* . Так как мы еще не знаем значения *Q¯* , мы не можем определить значение на выходе элемента N1. На вход N2 поступает 0 и *Q*. Так как мы еще не знаем значения *Q*, мы не можем определить значение на выходе элемента N2. Кажется, мы зашли в тупик. Этот случай аналогичен случаю с двумя перекрестно включенными инверторами. Мы знаем, что *Q* должен быть равен либо 0, либо 1. Итак, мы сможем решить проблему, если рассмотрим каждый из этих двух случаев.

1. *Случай IVa*: *Q*=0

Так как *S* и *Q* равны 0, то на выходе N2 будет логическая 1, *Q¯* =1, как показано на [**Рис.** **3.4**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page335) (a). Теперь на входе N1 есть одна единица – *Q¯*, поэтому на его выходе *Q*=0, как мы и предполагали.

1. *Случай IVb*: *Q*=1

Так как *Q* = 1, то на выходе N2 будет 0, *Q¯* =0, как показано на [**Рис. 3.4**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page335) **(b)**.Теперь на обоих входахN1нули(*R*и*Q¯*),поэтому на еговыходе логическая 1, *Q*=1, как мы и предполагали.



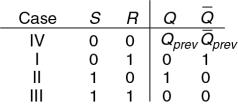
**Рис. 3.4 Бистабильные состояния RS-триггера**

Исходя из сказанного выше, предположим, что у *Q* есть какое-то определенное значение, установленное до наступления случая IV, которое мы назовем *Qпред*, *Qпред* может быть либо 0, либо 1. *Qпред* отражает состояние системы. Когда *R* и *S* равны 0, на выходе *Q* будет сохраняться старое значение *Qпред*, а *Q¯* будет его булевым дополнением.

Таблица истинности, приведенная на [**Рис.** **3.5**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page336) иллюстрирует эти четыре случая. Входы *R* и *S* отвечают за сброс и установку значений соответственно.

Установить бит означает перевести его в логическую единицу, 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, *Q¯* –нулем.Если ни на один из входов не поступает логическаяединица, 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**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page336) Условные обозначения используются при модульном проектировании схемы с целью абстрагирования от внутренней структуры элемента.

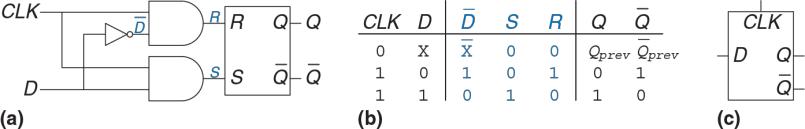


**Рис. 3.5 Таблица истинности RS-триггера** **Рис. 3.6 Обозначение RS-триггера**

Существует несколько способов построения RS-триггера, таких как использование логических элементов или транзисторов. Тем не менее, любой элемент схемы, специфицированный таблицей истинности на [**Рис. 3.5**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page336)обозначаетсясимволом на[**Рис. 3.6**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page336) иназываетсяRS-триггером.

Так же как и перекрестно включенные инверторы, RS-триггер является бистабильным элементом с одним битом состояния, хранящимся в *Q*. Состоянием можно управлять при помощи входов *R* и *S*. Когда на *R* поступает высокий уровень, выход сбрасывается в 0. Когда высокий уровень приходит на *S*, выход устанавливается в 1. Если ни на один вход не пришла логическая единица, триггер сохраняет свое предыдущее состояние, значение выходов не изменяется. Отметим, что вся история сигналов, поданных на вход, может быть сосредоточена в одной переменной состояния *Q*. Не имеет значения, что происходило в прошлом. Все, что нужно, чтобы предсказать будущее поведение RS-триггера, – это знать, было ли последнее изменение состояния триггера сбросом или установкой.

**3.2.2 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**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page338) **(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.

**Рис. 3.7** **D-триггер-защелка: (a) схема, (b) таблица истинности, (c) обозначение**

Для анализа защелки снова составим таблицу истинности [(**Рис.** **3.7**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page338) (b)). Сначала рассмотрим внутренние линии *D*¯, *R* и *S*. Если *CLK*=0, то оба сигнала *R* и *S* нулевые, независимо от значения *D*. Если *CLK*=1, на выходе одного элемента И будет единица, а на выходе другого – нуль. Элемент И, на выходе которого будет 1, определяется входом *D*. Значения *Q* и *Q¯* определяются *R* и *S* по таблице на [**Рис.** **3.5**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page336) Заметим, что пока *CLK*=0, *Q* сохраняет предыдущее значение *Qпред*. Если *CLK*=1, *Q*=*D*.Очевидно,что *Q¯* всегда является булевым дополнением *Q*.

В D-защелке исключен случай необычного поведения при одновременно поданных сигналах сброса и установки (*R*=1 и *S*=1).

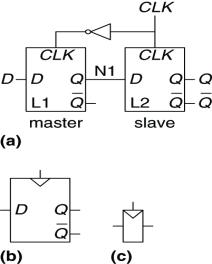
Таким образом, мы видим, что тактовый сигнал контролирует, КОГДА данные проходят через триггер-защелку. Когда *CLK*=1, защелка «*прозрачна*», т.е. она пропускает данные D на выход *Q*, как если бы он являлся обычным буфером. Когда *CLK*=0, защелка «*непрозрачна*», она не пропускает новые данные с входа D на выход *Q*, а *Q* сохраняет свое значение. D-защелку иногда называют *прозрачным триггером* или *триггером, синхронизируемым уровнем*.Условное обозначениеD-защелки представлено на [**Рис.** **3.7(c)**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page338).

Состояние D-триггера-защелки изменяется непрерывно, пока *CLK*=1. Позже в этой главе мы увидим, что часто удобнее изменять состояние схемы только в определенный момент времени. Следующий раздел – как раз об этом. В нем описывается D-триггер, синхронизируемый фронтом.

Иногда состояние защелки называют «открытым» или «закрытым», а не «прозрачным» или «непрозрачным».

**3.2.3 D-Триггер**

D-триггер, триггер синхронизируемый фронтом (далее – триггер), может быть построен из двух включенных последовательно D-защелок. Как показано на [**Рис.** **3.8**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page340) (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**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page340) **(b)**.Когда выход*Q¯*не используется,обозначение может бытьупрощено до представленного на [**Рис.** **3.8**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page340) (c).



**Рис. 3.8** **D-триггер: (a) схема, (b) обозначение, (c) упрощенное обозначение**

Когда *CLK*=0, master-защелка открыта, а slave – закрыта. Следовательно, значение со входа D проходит до линии N1. Когда *CLK*=1, master-защелка закрывается,аslave-защелка открывается.Значение с N1 проходит на выход *Q*, но N1 становится отрезанным от *D*. Следовательно, то значение, которое было на входе *D* непосредственно перед переходом *CLK* из 0 в 1, сразу же попадает на выход *Q* после того как тактовый сигнал устанавливается в 1. Во все остальное время *Q* сохраняет свое прежнее значение, так как закрытый триггер постоянно блокирует путь между *D* и *Q*.

Другими словами, *D-триггер копирует значение с* *D* *на* *Q* *по* *переднему фронту тактового импульса и помнит это состояние все остальное время*.Перечитайте это определение до тех пор,пока выего не запомните. Одна из самых распространенных ошибок начинающих разработчиков цифровых схем – они забывают, что такое синхронизация фронтом. Часто передний фронт тактового импульса называют просто фронтом. Вход D определяет новое, будущее состояние триггера. Фронт определяет момент времени, когда состояние будет обновлено.

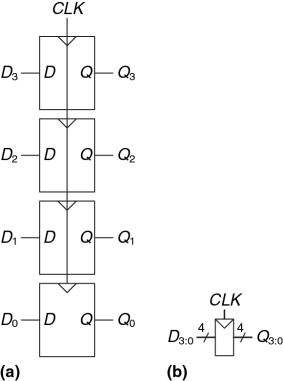
D-триггер также известен как *MS-триггер*, *master-slave-триггер* и как *триггер, синхронизируемый фронтом*.Треугольник в обозначенииуказывает на то, что вход синхронизируется фронтом. У многих триггеров выход *Q¯* отсутствует, и их обычно используют, когда *Q¯* не нужен.

Различие между триггером и защелкой весьма расплывчатое, оно изменялось с течением времени. В производственных кругах под триггером обычно понимают триггер, синхронизируемый фронтом, или, другими словами, это бистабильный элемент с тактовым входом. Состояние триггера изменяется только по переднему фронту тактового сигнала, то есть когда тактовый сигнал переходит из 0 в 1. Бистабильные элементы, в которых отсутствует синхронизация по фронту, обычно называют защелками.

Употребляя термины «триггер» или «защелка», обычно имеют в виду D-триггер или D-защелку соответственно, потому что именно эти триггеры чаще всего используются на практике.

**Регистр**

*N*-разрядный регистр–набор из *N* триггеров с общим тактовымсигналом. Таким образом, все биты регистра обновляются одновременно. Регистр является ключевым блоком при построении большинства последовательностных схем. На [**Рис.** **3.9**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016%20—%20копия.doc#page343) показана схематобозначение 4-разрядного регистра со входами *D*3:0 и выходами *Q*3:0. *D*3:0и *Q*3:0являются4-разрядными шинами.

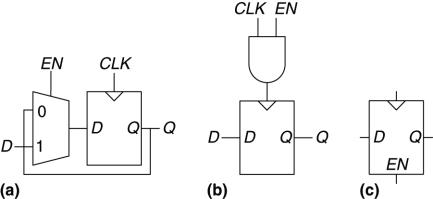


**Рис. 3.9** **4-разрядный регистр: (a) схема, (b) обозначение**

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

**3.2.5 Триггер с функцией разрешения**

1. В некоторых триггеров имеется еще один вход, называемый *EN*, или *ENABLE* (разрешить).Этот вход определяет,будут ли данныезагружены 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.



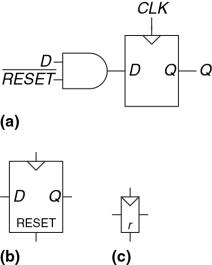
**Рис. 3.10 Триггер с функцией разрешения (a,b) схемы, (c) обозначение**

На [**Рис.** **3.10**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page344) показаны два способа добавления входа разрешения к обычному D-триггеру. На [**Рис.** **3.10**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page344) **(а)** входной мультиплексор выбирает, подавать ли данные на вход *D*, если на *EN* логическая единица, или подавать на вход *D* старое значение с выхода *Q*, если на *EN* подается логический нуль. На [**Рис.** **3.10**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page344) (b) тактовый сигнал проходит, если *EN* равен единице; импульсы на вход тактового сигнала подаются в обычном режиме. Если на *EN* – логический нуль, то и на *CLK* –также нуль,и триггер сохраняет свое предыдущее состояние.Заметим, что сигнал *EN* не должен изменяться, пока *CLK*=1, во избежание сбоя (выброса) тактового сигнала (переключение в неверное время). Вообще говоря, добавление логических элементов в тракт тактирования – плохая идея. Управление тактированием вносит задержку в тактовый сигнал и может привести к временным ошибкам, о чем будет сказано в **разделе** [**3.5.3**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page421) то есть делайте так только в том случае, если вы уверены в том, что вы делаете. Обозначение триггера с функцией разрешения представлено на [**Рис.** **3.10**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page344) **(с)**.

**3.2.6 Триггер с функцией сброса**

1. *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.



**Рис. 3.11 Синхронно сбрасываемый триггер: (a) схема, (b,c) обозначения**

На [**Рис.** **3.11**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page346) **(а)** показано, как построить синхронно сбрасываемый триггер из обычного D-триггера и элемента И. Когда на *RESET* поступает логический нуль, элемент И подает 0 на вход триггера. Когда на *RESET* поступает логическая единица, элемент И пропускает сигнал *D* на вход триггера. В этом примере *RESET* – сигнал с активным низким уровнем (инверсная логика). Это означает, что сброс происходит, когда на этот вход поступает 0, а не 1. Добавив инвертор, мы могли бы получить схему с активным высоким уровнем (прямая логика). На [**Рис.** **3.11**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page346) (b) и [**Рис.** **3.11**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page346) **(с)** показаны обозначения сбрасываемого триггера с прямым сбросом.

Асинхронно сбрасываемые триггеры требуют изменения своей внутренней структуры и оставлены для самостоятельного разбора (**упражнение** [**3.13**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page461)), однако, и они зачастую доступны разработчикам как стандартный компонент.

Как вы могли бы легко догадаться, иногда используются и триггеры с функцией установки. Когда установлен сигнал SET, в такой триггер загружается логическая 1, и они происходят в синхронном и асинхронном исполнениях. У сбрасываемых и устанавливаемых триггеров также может быть вход ENABLE, и они могут быть сгруппированы в N-разрядные регистры.

**3.3** **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. Этот раздел начнется с разбора нескольких таких схем. Затем мы введем понятия синхронных последовательностных схем и динамической дисциплины. Ограничив себя рассмотрением только синхронных последовательностных схем, мы сможем сформулировать простые систематические подходы к анализу и проектированию таких схем.

**3.3.1 Некоторые проблемные схемы**

**1.** НЕУСТОЙЧИВЫЕ СХЕМЫ

**2.** ГОНКИ В ПОСЛЕДОВАТЕЛЬНОСТНЫХ СХЕМАХ

**3.3.2 Синхронные последовательностные схемы**

1. В предыдущих двух примерах присутствовали циклические пути, в которых выходы напрямую соединены обратной связью со входами.

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.

Напомним, что схема (цепь) определяется набором входов и выходов и функциональными и временными параметрами. У последовательностной схемы существует конечный набор дискретных *состояний* {S0, S1, …, Sk-1}. У *синхронной* *последовательностной схемы* есть вход тактового сигнала,передниефронты тактовых импульсов определяют последовательность точек на временной оси, в которых происходят изменения состояния. Мы часто будем использовать термины «*текущее состояние*» и «*следующее* *состояние*»для того,чтобы различать состояние системы в настоящемот состояния системы, в которое она перейдет по фронту следующего тактового импульса. Функциональное описание определяет следующее состояние и значение каждого выхода для каждой возможной комбинации текущих состояний и входных сигналов. Временная спецификация состоит из верхней границы *tpcq* и нижней границы *tccq* длительности временного промежутка от переднего фронта тактового импульса до момента изменения *выходного* сигнала, а также из времен *предустановки* и *удержания tsetup* и *thold*,которые определяетпромежуток времени до и после поступления фронта тактового импульса, в течение которого значения на входах не должны изменяться.

Правила *построения синхронных последовательностных схем* гласят, что схема является синхронной последовательностной схемой, если ее элементы удовлетворяют следующим условиям:

Каждый элемент схемы является либо регистром, либо комбинационной схемой.

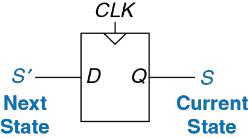
Как минимум один элемент схемы является регистром.

Все регистры тактируются единственным тактовым сигналом.

В каждом циклическом пути присутствует как минимум один регистр

Последовательностные схемы, не являющиеся синхронными, называют асинхронными.

Триггер является самой простой синхронной последовательностной схемой с двумя состояниями {0,1}. У него есть один вход данных *D*, один вход тактового сигнала *CLK*, один выход *Q*.



**Рис. 3.20 Текущее и следующее состояние триггера**

Функциональное описание D-триггера заключается в том, что его следующим состоянием является значение входа *D*, а значение выхода *Q* является текущим состоянием, (См.[**Рис.****3.20**)](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page360)*.*

Мы часто будем обозначать текущее состояние переменной *S*, следующее состояние – переменной *S*’, то есть *S*’ (штрих) обозначает следующее состояние, а не инверсию. Временные диаграммы последовательностных схем будут рассмотрены в **разделе** [**3.5**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page404)

Два других вида синхронных последовательностных схем – конечные автоматы и конвейеры. Они будут рассмотрены позже в этой главе.

**3.3.3 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

55. Конечные автоматы. Пример проектирования конечного автомата

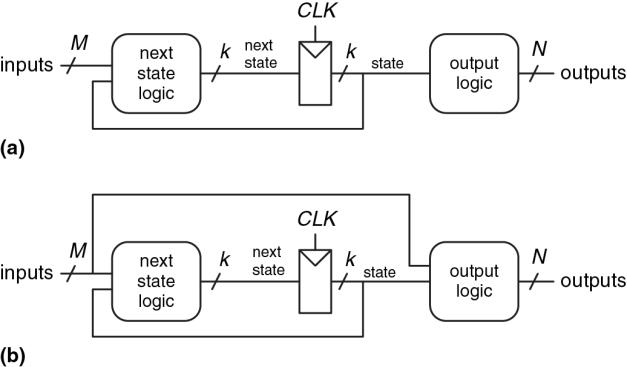
**3.4** **КОНЕЧНЫЕ АВТОМАТЫ**

Последовательностные логические схемы могут быть изображены в форме, представленной на [**Рис.** **3.22**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page366)

Такие представления *называются конечными автоматами* (*КА*). Они получили свое название из-за того, что схема с *k*-регистрами может находиться в одном из 2*k*, 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 текущеесостояние. 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.

1. *В автомате Мура* выходные значения зависят лишь от текущегосостояния, 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.
2. **3.4.1 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 Конечные автоматы: (a) автомат Мура (b) автомат Мили**

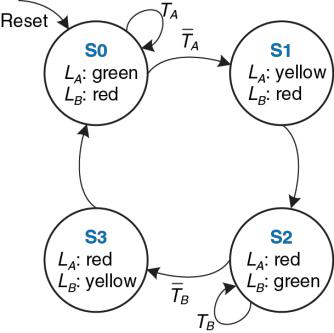
Бен решил справиться с проблемой с помощью конечного автомата. Он установил два датчика движения, *TА* и *TБ*, на Академической и Беговой улицах соответственно. Каждый датчик выдает единицу, если студенты присутствуют на улице и нуль, если никого нет. Он также установил два светофора для управления движением, *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**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page367) 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**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page367) автомат изображен в виде «черного ящика».



**Рис. 3.24 Конечный автомат как «черный ящик»**

**Рис. 3.23 Карта кампуса**

Следующий шаг первокурсника – сделать набросок *диаграммы* *переходов* (или графа),показанный на[**Рис.****3.25**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page368)накотором приведенывсе возможные состояния системы и переходы между ними.

**Рис. 3.25 Таблица переходов**

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 ее светофоре 5 секунд горит желтый, затем 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.

На основе этой диаграммы переходов Бен Битдидл записал таблицу переходов [(**Табл. 3.1**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page371)), которая отражает, каким должно быть следующее состояние *S*’, соответствующее текущему состоянию и входным сигналам. Заметим, что в таблице используются символы Х, означающие, что следующее состояние не зависит от конкретного входа. Также заметим, что сигнал сброс исключен из этой таблицы. Вместо этого мы использовали сбрасываемые триггеры, которые переходят в состояние S0 сразу после сброса, независимо от данных на входе.

Диаграмма переходов абстрактна в том смысле, что она использует состояния, обозначенные как {S0, S1, S2, S3}, и выходы, обозначенные как {красный, желтый, зеленый}.

Для построения реальной схемы состояниям и выходам должны быть поставлены в соответствие двоичные коды.

Бен выбрал простое кодирование, см. [**Табл. 3.2**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page371) и [**Табл. 3.3**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page371) Каждое состояние и каждое выходное значение закодировано двумя битами:

*S*1:0, *L*A1:0и *L*B1:0.

\*S0, S1 – обозначение состояний с индексами – биты двоичного числа, соотв. некоторому состоянию.

Бен переписывает таблицу переходов, используя двоичное кодирование, как показано в [**Табл.** **3.4**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page372) Эта таблица является таблицейистинности, определяющей логику следующего состояния. Она определяет следующее состояние *S*’ как функцию входов и текущего состояния.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Табл. 3.1 Таблица переходов** | | | |  | **Табл. 3.2 Кодирование** | |  |  | **Табл. 3.3 Кодирование** | |
|  |  |  |  |  |  | **состояний** | |  |  | **выходов** | |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | **Current** | **Inputs** | | **Next State** |  | **State** | **Encoding *S*1:0** |  |  | **Output** | **Encoding *L*1:0** |
|  | **State *S*** | ***TA*** | ***TB*** | ***S′*** |  | S0 | 00 |  |  | green | 00 |
|  | S0 | 0 | X | S1 |  | S1 | 01 |  |  | yellow | 01 |
|  | S0 | 1 | X | S0 |  | S2 | 10 |  |  | red | 10 |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | S1 | X | X | S2 |  | S3 | 11 |  |  |  |  |
|  | S2 | X | 0 | S3 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | S2 | X | 1 | S2 |  |  |  |  |  |  |  |
|  | S3 | X | X | S0 |  |  |  |  |  |  |  |

**Табл. 3.4 Таблица переходов с двоичным кодированием**

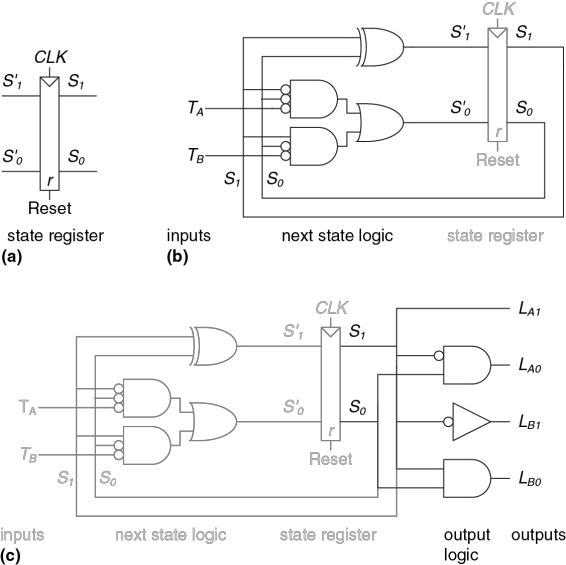
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Current State** | |  | **Inputs** | | **Next State** | | |
|  | ***S*1** | ***S*0** | ***TA*** |  | ***TB*** | ***S′*** |  | ***S′*** |
|  | 0 | 0 | 0 |  | X | 0 |  | 1 |
|  | 0 | 0 | 1 |  | X | 0 | 0 | |
|  |  |  |  |  |  |  |  | |
|  | 0 | 1 | X |  | X | 1 |  | 0 |
|  | 1 | 0 | X | 0 | | 1 | 1 | |
|  |  |  |  |  | |  |  | |
|  | 1 | 0 | X |  | 1 | 1 |  | 0 |
|  | 1 | 1 | X |  | X | 0 | 0 | |
|  |  |  |  |  |  |  |  |  |

(3.2)

Подобным образом Бен записывает таблицу выходов [(**Табл.** **3.5**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page374)), определяя, каким должен быть выход для каждого состояния. Затем он снова составляет и упрощает булевы выражения для выходов. Например, *L*A1 =1 в строках, где истинно *S*1=1.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | **Табл. 3.5 Таблица выходов** | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | **Current State** | |  |  | **Outputs** | |  |  |
|  |  | ***S*1** | ***S*0** |  | ***LA*1** | ***LA*0** | ***LB*1** | ***LB*0** |  |
|  |  | 0 | 0 |  | 0 | 0 | 1 | 0 |  |
|  | | 0 | 1 |  | 0 | 1 | 1 | 0 |  |
|  |  | |  |  |  |  |  |  |  |
|  |  | 1 | 0 |  | 1 | 0 | 0 | 0 |  |
|  | | 1 | 1 |  | 1 | 0 | 0 | 1 |  |
|  |  | |  |  |  |  |  |  |  |

Сначала он рисует 2-разрядный регистр состояний, как показано на [**Рис.** **3.26**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page375) (a). По каждому переднему фронту тактового сигнала регистр состояний фиксирует следующее состояние *S*’1:0, и, таким образом, оно становится текущим состоянием *S*1:0 Регистр состояний получает сигнал синхронного или асинхронного сброса для инициализации КА после подачи питания. Затем, основываясь на **уравнениях** [**(3.2)**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page373)Бенрисует схему определения следующегосостояния, которые вычисляют следующее состояние по значению на входах и по текущему состоянию. Эта схема показана на [**Рис.** **3.26**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page375) (b). Наконец, он по **уравнениям** [**(3.3)**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page373) рисует схему (см. [**Рис.** **3.26**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page375) (**с)**), которая вычисляет значения на выходах автомата по текущему состоянию.



**Рис. 3.26 Схема конечного автомата контроллера светофора**

# **46.Конечные автоматы. Кодирование состояний. Автоматы Мура и Мили. Декомпозиция конечных автоматов. Восстановление конечных автоматов по электрической схеме (320-339)**

**3.4.2 Кодирование состояний**

1. 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 сконтроллером светофора, каждому состоянию ставится в соответствие двоичное число (номер этого состояния). Так как K двоичных чисел можно записать в log2*K* разрядах, системе с *K* состояниями нужно всегоlog2*K* битовсостояния.

*В прямом кодировании* для каждого состоянияиспользуется 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 .

* + 1. **Автоматы Мура и Мили**

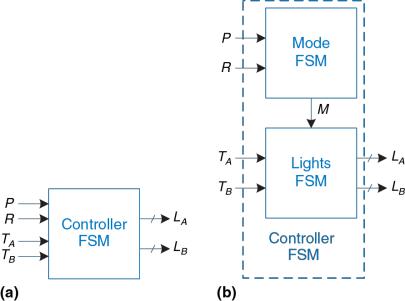
До сих пор мы рассматривали примеры автоматов Мура, выход в которых зависит лишь от состояния системы. Поэтому на диаграммах переходов для автоматов Мура значения выходов пишутся внутри кружков. Вспомним, что автоматы Мили очень похожи на автоматы Мура, но значения на их выходах могут зависеть он значений 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**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page392) (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.

**3.4.4 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.

**Пример 3.8** МОДУЛЬНЫЕ И НЕМОДУЛЬНЫЕ КОНЕЧНЫЕ АВТОМАТЫ



**Рис. 3.33 (a) Немодульная и (b) модульная модель КА модифицированного контроллера светофора**

**3.4.5 Восстановление конечных автоматов по электрической схеме**

Восстановление конечных автоматов по электрической схеме практически является процессом, обратным проектированию КА. Этот процесс необходим, например, при рассмотрении проекта с неполной документацией или для реверсивного проектирования чьей-то системы.

1. 1,Проанализируйте схему, возможные состояния входов, выходов и регистра состояний.
2. 2,Составьте выражения для следующего состояния и для выходов.
3. 3,Составьте таблицу выходов и таблицу переходов.
4. 4,Вычеркните из таблицы переходов состояния, в которые система никогда не попадает.
5. 5,Присвойте имя каждому используемому набору бит-состояний.
6. 6,Перепишите таблицы выходов и переходов, используя эти обозначения.
7. 7,Нарисуйте диаграмму переходов.

8,Опишите словами то, что делает автомат

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

# **47.Синхронизация последовательностных схем. Временные характеристики системы. Расфазировка тактовых сигналов. Метастабильность. Синхронизаторы (346-379)**

**3.5** **СИНХРОНИЗАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТНЫХ 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 их характеристик фотокамеры является *апертурное время*, в течение которого фотографируемый объект должен быть неподвижен, чтобы на фотографии сформировалось его резкое изображение. Аналогично, последовательностный 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.

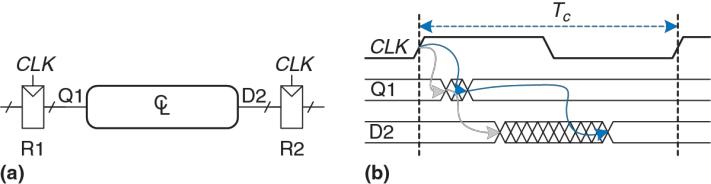
Часть апертурного времени последовательностного элемента до фронта тактового импульса называется временем *предустановки* (*setup time*), после фронта – *временем удержания* (*hold time*). Подобно статической дисциплине, которая разрешает использование логических уровней только за пределами запретной зоны, динамическая дисциплина позволяет использовать только те сигналы, которые изменяются вне апертурного времени. При выполнении требований динамической дисциплины мы можем оперировать дискретными единицами времени, которые называются тактовыми циклами, аналогично тому, как мы оперируем с дискретными логическими уровнями 1 и 0. Сигнал может изменяться и осциллировать в течение некоторого ограниченного промежутка времени. При выполнении требований динамической дисциплины важно лишь его значение в конце цикла тактового сигнала, когда он уже принял стабильное значение. Следовательно, для описания сигнала *A* можно использовать его величину *A*[*n*] в конце *n*-го цикла тактового импульса, где *n* – целое число, вместо его величины *A*(*t*) в произвольный времени *t*, где *t* – действительное число.

Период тактовых импульсов должен быть достаточно большим, чтобы переходные процессы всех сигналов успели завершиться. Это требование ограничивает быстродействие всей системы. В реальных системах тактовые импульсы поступают на входы триггеров неодновременно. Этот разброс по времени, который называется расфазировкой или разбросом фаз тактового сигнала (clock skew), заставляет разработчиков дополнительно увеличивать период тактовых сигналов.

Иногда невозможно удовлетворить требованиям динамической дисциплины, особенно в устройствах сопряжения цифровой системы с реальным миром. Например, рассмотрим схему, к входу которой подключена кнопка. Обезьяна может нажать на кнопку как раз во время фронта тактового импульса. Это может привести к возникновению явления, которое называется метастабильностью, при этом триггер оказывается в промежуточном состоянии между 0 и 1, причем переход корректное логическое состояние (0 или 1) может происходить бесконечно долго. Решением проблемы асинхронных входов является использование синхронизатора, на выходе которого некорректный логический уровень может появиться с очень малой (но не нулевой) вероятностью.

Эти идеи будут детально рассмотрены в оставшейся части раздела.

* + 1. **Временные характеристики системы**

*Периодом тактового сигнала* или *длительностью цикла синхронизации, Tc*,называется промежуток времени между переднимифронтами последовательных тактовых импульсов. Обратная величина, *fc* = 1/*Tc*,называется *тактовой частотой*.Увеличение тактовойчастоты 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, and 1 гигагерц (ГГц) = 109 Гц.

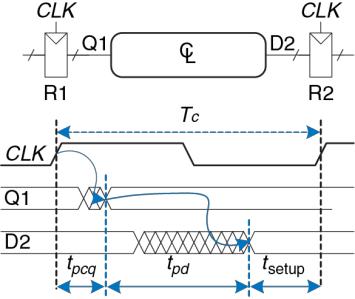
**Рис. 3.38 Тракт между регистрами и временная диаграмма**

На [**Рис.** **3.38**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page410) (a) показана характерная структура тракта обработки информации синхронной последовательностной схемы, для которой мы рассчитаем период тактового сигнала. По переднему фронту тактового импульса на выходе регистра R1 формируется выходной сигнал (или сигналы) *Q*1. Эти сигналы поступают на вход блока комбинационной логики, выходные сигналы этого блока поступают на вход (или входы) *D*2регистраR2.Как показано на[**Рис.****3.38**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page410) *(b)*,выходной сигнал блокаможет 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.

**Ограничение времени предустановки**

На [**Рис.** **3.39**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page411) на временной диаграмме приведена только максимальная задержка в тракте обработке информации, которая обозначена синими стрелками.



**Рис. 3.39 Максимальная задержка для ограничения времени предустановки**

Для выполнения ограничения по времени предустановки регистра R2, сигнал *D*2 должен установиться не позднее, чем за время предустановки до фронта следующего тактового импульса. Таким образом, мы можем получить выражение для минимальной длительности периода синхросигнала:

|  |  |
| --- | --- |
| *Tc ≥tpcq +tpd +t*setup | (3.13) |

При проектировании коммерческих продуктов период тактового сигнала будущего изделия часто задается из соображений конкурентоспособности руководителем отдела разработок или отделом маркетинга. Более того, задержка распространения сигнала триггером от фронта тактового сигнала до выхода (Clock-to-Q) и время предустановки *tpcq* и *t*setup определены производителем. Следовательно, неравенство [**(3.13)**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page412) следует преобразовать для определения максимальной задержки распространения комбинационной схемы, поскольку обычно именно это – единственный параметр, который может изменять проектировщик:

|  |  |
| --- | --- |
| *tpd ≤Tc−*(*tpcq +t*setup) | (3.14) |

Слагаемое в скобках, *tpcq* + *t*setup, называется *потерями на упорядочение* (*sequencing overhead*). В идеальном случае весь период тактового сигнала может быть затрачен на вычисления в комбинационной логике (время *tpd*). Однако, потери на упорядочение в триггерах уменьшают это время. Неравенство [**(3.14)**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page412) называется *ограничением времени* *предустановки* или *ограничением максимальной задержки*,поскольку оно зависит от времени предустановки и ограничивает максимальную задержку распространения в комбинационной логической схеме.

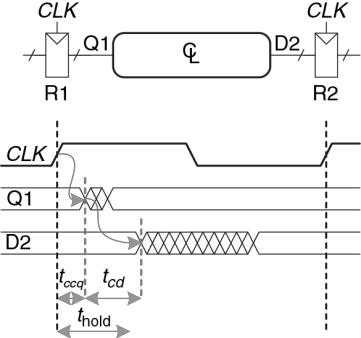
Если задержка распространения в комбинационной схеме слишком велика, то вход *D*2 может не успеть принять свое установившееся состояние ко времени, когда регистр R2 ожидает стабильности и фиксирует его. Таким образом, R2 может зафиксировать некорректный результат или даже логический уровень в запретной зоне. В таком случае схема будет работать некорректно. Проблему можно решить увеличением периода тактового сигнала или пересмотром комбинационной схемы с целью добиться меньшей задержки распространения.

**Ограничение времени удержания**

Регистр R2 на [**Рис.** **3.38**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page410) (a) имеет также *ограничение времени* *удержания*.Его вход, *D*2,не должен изменяться в течение некотороговремени *t*hold после переднего фронта тактового импульса.

1. В соответствии с [**Рис.** **3.40**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page414) *D*2 может измениться через *tccq* + *tcd* после переднего фронта тактового импульса. Следовательно, можно записать:

|  |  |
| --- | --- |
| *tccq +tcd ≥thold* | (3.15) |



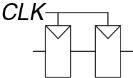
**Рис. 3.40 Минимальная задержка для ограничения времени удержания**

Как и ранее, характеристики используемого в схеме триггера *tccq* и *t*hold обычно находятся вне влияния разработчика схемы. После простых преобразований мы можем записать неравенство для минимальной задержки комбинационной логической схемы:

|  |  |
| --- | --- |
| *tcd ≥ t*hold *– tccq* | (3.16) |

Неравенство [**(3.16)**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page414) также называется *ограничением времени* *удержания* или *ограничением минимальной задержки*,потому что оноограничивает минимальную задержку комбинационной схемы.

Мы предполагаем, что при соединении логических элементов между собой временные проблемы синхронизации не возникают. В частности, мы считаем, что при непосредственном последовательном соединении двух триггеров, как показано на [**Рис.** **3.41**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page415) проблемы, обусловленные временем удержания, не возникают.



**Рис. 3.41 Непосредственное последовательное соединение триггеров**

1. В этом случае, вследствие отсутствия комбинационной логики между триггерами, *tcd* = 0. При такой подстановке неравенство [**(3.16)**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page414) сводится к требованию:

|  |  |
| --- | --- |
| *t*hold *≤tccq* | (3.17) |

Иными словами, время удержания надежного триггера должно быть меньше, чем его задержка реакции. Часто триггеры проектируются так, что *t*hold = 0, следовательно, неравенство [**(3.17)**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page415) всегда выполняется.

этой книге, если не указано обратное, мы будем считать такое предположение истинным- и игнорировать ограничение времени удержания.

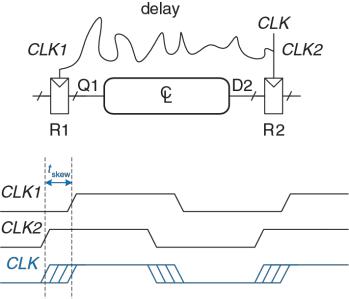
Тем не менее, ограничения времени удержания критически важны. Если они нарушаются, то единственным решением является увеличение задержки реакции комбинационной схемы, что требует ее перепроектирования. Такие нарушения, в отличие от нарушений ограничений времени предустановки, не могут быть исправлены изменением периода тактового сигнала. Перепроектирование интегральной микросхемы и производство ее исправленного варианта занимает несколько месяцев и требует затрат в несколько миллионов долларов при современных технологиях, поэтому к *нарушениям* *ограничения времени удержания* нужно относиться крайне серьезно.

**Заключение**

Последовательностные схемы имеют ограничения времен предустановки и удержания, которые устанавливают максимальную и минимальную задержки в комбинационной логической схеме между триггерами. Современные триггеры обычно спроектированы так, что минимальная задержка в комбинационной логике равна нулю, то есть триггеры могут быть размещены непосредственно друг за другом. Максимальная задержка ограничивает число последовательных логических элементов, включенных один за другим в критическом пути быстродействующей схемы.

**3.5.3 Расфазировка тактовых сигналов**

1. 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*.

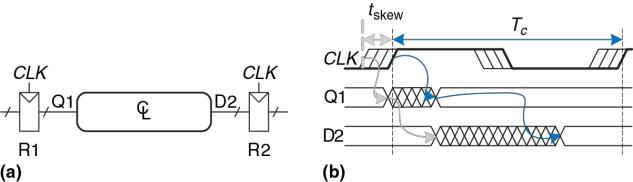


**Рис. 3.46 Расфазировка тактовых сигналов, обусловленная задержками в межсоединениях**

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**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page422)

Шум также приводит к различным задержкам. Стробирование тактовых сигналов, которое было описано в **разделе** [**3.2.5**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page344) приводит к их дополнительной задержке. Если в схеме используются стробированные нестробированные тактовые сигналы, то между ними будет существенное рассогласование. На [**Рис.** **3.46**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page422) сигнал *CLK*2 будет опережать по времени сигнал *CLK*1 из-за сложного пути тактового сигнала между регистрами. Если трассировка цепи тактового сигнала будет выполнена по-другому, ситуация может быть противоположной, *CLK*2будет отставать от сигнала *CLK*1.При выполнении временногоанализа мы рассматриваем наихудший случай, что позволяет гарантировать, что схема будет работать при всех условиях.

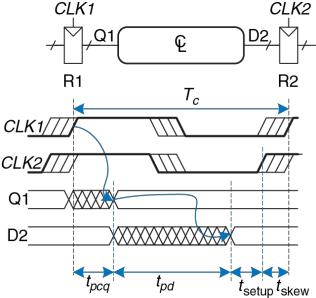
Учет расфазировки изменяет временную диаграмму, которая была показана на [**Рис.** **3.38**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page410) модифицированная диаграмма приведена на [**Рис.** **3.47**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page423)



**Рис. 3.47 Временная диаграмма с учетом расфазировки тактовых импульсов**

Жирной линией показана максимальная задержка тактового сигнала, тонкие линии показывают, что синхросигнал, может появиться на *t*skew.раньше.

Вначале рассмотрим ограничение времени предустановки, соответствующие диаграммы приведены на [**Рис.** **3.48**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page424)



**Рис. 3.48 Ограничение времени предустановки с учетом расфазировки тактовых импульсов**

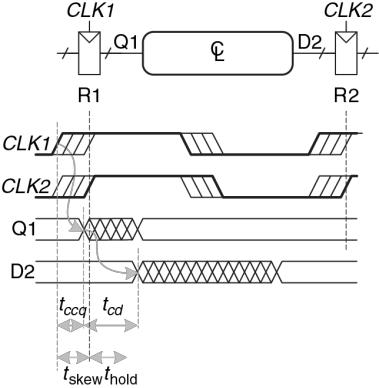
В худшем случае на регистр R1 поступает тактовый сигнал с наибольшей задержкой, а на R2 – с наименьшей, что оставляет минимальное время для прохождения данных через комбинационную схему между регистрами.

На вход регистра R2 данные поступают через регистр R1 и комбинационную логику, они должны прийти к стационарному состоянию перед началом их фиксации регистром R2. Следовательно, можно сделать вывод, что

|  |  |
| --- | --- |
| *Tc ≥ tpcq + tpd + t*setup *+ t*skew | (3.19) |
| *tcd ≥ t*hold *+ t*skew*− tccq* | (3.20) |

Далее мы рассмотрим ограничение времени удержания (см. [**Рис.** **3.49**)](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page426).

1. В худшем случае на регистр R1 поступает тактовый сигнал с наименьшей задержкой, а на R2 – с наибольшей. Данные могут быстро пройти через регистр R1 и комбинационную логику, но должны поступить на вход регистра R2 не ранее окончания времени удержания после переднего фронта тактового импульса.



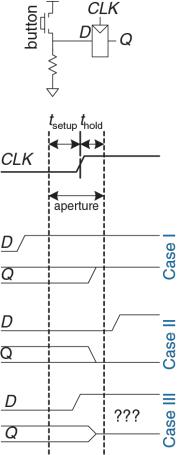
**Рис. 3.49 Ограничение времени удержания с учетом расфазировки тактовых импульсов**

Таким образом, можно записать:

|  |  |
| --- | --- |
| *tccq + tcd ≥ t*hold *+ t*skew | (3.21) |
| *tcd ≥ t*hold *+ t*skew *− tccq* | (3.22) |

В итоге, расфазировка тактовых импульсов приводит к эффективному увеличению как времени предустановки, так и времени удержания. Это, в свою очередь, приводит к росту потерь на упорядочение и уменьшает время, доступное для обработки данных комбинационной схемой. Даже если *t*hold = 0, пара последовательно соединенных триггеров будет нарушать неравенство [**(3.22)**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page426) если if *t*skew > *tccq* . Чтобы предотвратить такие серьезные нарушения ограничений времени удержания, проектировщик должен ограничивать расфазировку тактовых сигналов. Иногда триггеры специально проектируются медленными (время *tccq* велико), чтобы избежать проблем времени удержания, даже если расфазировка тактовых сигналов существенна.

**3.5.4 Метастабильность**

 Как было указано ранее, не всегда можно гарантировать, что вход последовательностной 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**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page429) Когда кнопка не нажата, *D* = 0.Когда кнопка нажата, *D* = 1.Обезьяна можетнажимать 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 случаеII,кнопка нажимается тольконамного позже фронта *CLK*, *Q* = 0. Но в случае III, когда кнопка нажимается в промежутке, который охватывает время предустановки перед фронтом тактового импульса и время удержания после него,

входной сигнал нарушает динамическую дисциплину и выход будет неопределенным.

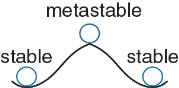
**Рис. 3.50 Входной сигнал, который изменяется до, после или в течение апертурного времени**

**Метастабильное состояние**

Когда состояние информационного входа триггера изменяется в течение апертурного времени, на его выходе *Q* может на некоторое время появиться напряжение в диапазоне от 0 до *VDD*, то есть в запретной зоне. Такое состояние называется *метастабильным*. Со временем выход триггера перейдет в *стабильное* *состояние* 0или1.Однако *время разрешения*,необходимое для достижения стабильного состояния, не ограничивается.

Метастабильное состояние триггера подобно состоянию шарика на вершине между двумя впадинами, как показано на [**Рис.** **3.51**.](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page431) Положения во впадинах являются стабильными, поскольку шарик будет находиться в них неограниченно долго при отсутствии внешнего возмущения.

Положение на вершине возвышенности называется метастабильным, потому что шарик будет находиться в нем только при условии идеальной балансировки. Но, поскольку в мире нет ничего совершенного, со временем шарик скатится в одну из впадин. Необходимое для этого время зависит от степени первоначальной балансировки шарика. Каждое бистабильное устройство имеет метастабильное состояние между двумя стабильными.



**Рис. 3.51 Стабильное и метастабильное состояния**

**Время разрешения**

Если вход триггера изменяется в произвольный момент цикла тактового сигнала, то время разрешения, *tres*, необходимое для перехода в стабильное состояние, также является случайной величиной. Если вход изменяется вне апертурного времени, то *tres* = *tpcq*. Но если произойдет изменение входа в апертурное время, *tres* может быть существенно больше.

Теоретическое и экспериментальное рассмотрение (см. **раздел** [**3.5.6**)](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page437) показывает, что вероятность того, что время разрешения превышает некоторое время *t*, экспоненциально падает с ростом *t*:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *T*0 | *e*- | *t* |  |
|  |  |  |
| *P*(*tres* > *t*)= | *t* | (3.24) |
|  |
|  | *Tc* | | |  |

где *Tc* – период тактового сигнала, *T*0 и t – характеристики триггера.

Выражение справедливо, только если *t* намного больше, чем *tpcq*.

Интуитивно понятно, что отношение *T*0/*Tc* описывает вероятность того, что вход изменится в неудачное время (то есть в апертурное время); эта вероятность уменьшается с ростом периода тактового сигнала *Tc*. – временная константа, которая показывает, насколько быстро триггер выходит из метастабильного состояния; она связана с задержкой в перекрестно соединенных вентилях триггера.

Таким образом, если вход бистабильного устройства, такого как триггер, изменяется в течении апертурного времени, его выход может некоторое время находиться в метастабильном состоянии, прежде чем перейти в стабильное состояние 0 или 1. Время перехода в стабильное состояние не ограничено, потому что для любого конечного времени *t* вероятность того, что триггер все еще находится в метастабильном состоянии, не равна нулю. Однако, эта вероятность экспоненциально падает с ростом *t*. Следовательно, если подождать достаточно долго, намного больше, чем *tpcq*, то с весьма высокой вероятностью можно ожидать того, что триггер достигнет корректного логического состояния.

**3.5.5 Синхронизаторы**

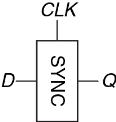
Наличие асинхронных входов цифровой системы, которые принимают информацию из внешнего мира, неизбежно. Например, сигналы, которые формирует человек, асинхронны. Такие асинхронные входы, если к ним относиться небрежно, могут привести к появлению метастабильных состояний в системе, что приведет к ее непредсказуемым 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**,](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page435) 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.

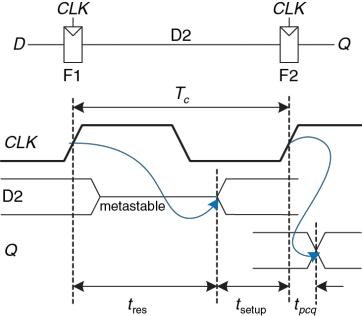
На [**Рис.** **3.1**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page330) показано, как из двух триггеров можно построить простой синхронизатор. Триггер F1 фиксирует значение входного сигнала *D* по переднему фронту тактового сигнала *CLK*. 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.

*Мы говорим о сбое* синхронизатора,если его выход *Q* станетметастабильным. Это может произойти, если *D*2 не успеет прийти к корректному состоянию до начала времени предустановки триггера F2,

то есть когда *tres* > *Tc* – *t*setup.



**Рис. 3.52 Символ синхронизатора**



**Рис. 3.53 Простой синхронизатор**

Вероятность сбоя, *P*(failure), есть вероятность того, что выход *Q* будет метастабильным после однократного изменения входа *D.* Если *D* изменяется один раз за секунду, то вероятность сбоя за одну секунду будет просто *P*(failure). Однако, если *D* изменяется *N* раз за секунду, то вероятность ошибки за секунду будет в *N* раз большей:

-*Tc* -*ts etup*

*P*(f ailure) / sec= *N T*0 *e* *t* (3.26)

*Tc*

Надежность системы обычно измеряют *средним временем наработки* *на отказ* (*mean time between failures*, *MTBF*).Как понятно из названия,MTBF – это среднее время между отказами системы. Эта величина обратна вероятности сбоя системы за любую заданную секунду:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | - | *Tc* -*ts etup* |  |  |
| *MTBF* = | 1 | = | *Tce* |  | *t* |  | (3.27) |
|  |  |  |
| *P*(f ailure) / sec |  | *NT*0 | |  |  |
|  |  |  |  |  |

Выражение [**(3.27)**](file:///C:\Users\Asus\AppData\Local\Temp\Цифровая_схемотехника_и_архитектура_компьютераJuly16_2016-1.doc#page436) показывает, что MTBF растет экспоненциально с ростом времени ожидания синхронизатора, *Tc*. Для большинства систем синхронизатор, который ожидает один период тактового сигнала, обеспечивает достаточную величину MTBF. В высокоскоростных системах может понадобиться ожидание на большее количество периодов тактового сигнала.

**48.Параллелизм**

**3.6** **ПАРАЛЛЕЛИЗМ**

Скорость обработки информации системой характеризуется задержкой и пропускной

способностью информации передачи информации через нее. Мы определим *токен*

(*token*) как группу входов, которая обрабатывается для того, чтобы получить группу выходов. Это название связано с методом визуализации передачи данных внутри системы путем размещения в схеме токенов или маркеров и их передвижением по схеме вместе обрабатываемыми данными. *Задержка,* или *латентность* (*latency*)системы–время,которое необходимо для прохождения одного токена через всю систему с ее входа на выход.

*Пропускнаяспособность*(*throughput*)*–* количество токенов, которое может быть обработано системой в единицу времени.

Достаточно легко понять, что пропускная способность может быть увеличена путем обработки нескольких токенов в одно и то же время. Это называется *параллелизмом* и используется в двух формах: пространственной и временной. В *пространственном параллелизме* используется несколько копий аппаратных блоков, так что в одно и то же время можно выполнять несколько задач. *Временной параллелизм* предполагает разбиение задачи на несколько стадий (или ступеней), как это происходит на сборочном конвейере. Несколько задач могут быть распределены по ступеням. Хотя все задачи должны пройти по всем ступеням, разные задачи в любой заданный момент времени будут находиться на своей ступени, так что несколько задач могут одновременно обрабатываться на разных ступенях. Временной параллелизм часто называется *конвейеризацией*. Пространственный параллелизм часто называют просто параллелизмом, но мы будем избегать этого названия из-за его неоднозначности.

**Пространственный параллелизм:** Бен просит Алису Хакер помочь ему.У нееесть собственная печь и противень.

**Временной параллелизм:** Бену дали второй противень.Как только он ставитодин противень в печь, он начинает сворачивать печенье на другом противне, а не ожидает окончания выпекания печенья на первом противне.

Пропускная способность – это количество противней с печеньем, которое производится за один час.

Конвейеризация (временной параллелизм) особенно привлекательна, поскольку она увеличивает скорость схемы без увеличения аппаратных затрат. Вместо этого, регистры, установленные между блоками комбинационной логики, разделяют ее на короткие ступени, которые могут работать на более высокой тактовой частоте. Регистры не позволяют токенам, находящимся в одной ступени, догонять и разрушать токены, которые находятся в следующей стадии обработки.

Использование параллелизма ограничивается *взаимозависимостями* (*dependencies*) реальных задач. Если текущая задача зависит от результатов предыдущей задачи, а не только от своих предыдущих шагов, то выполнение задачи не может быть начато до завершения предыдущей задачи. Например, если Бен Битдидл хочет проверить, достаточно ли вкусны печенья из первого противня, перед приготовлением второго, он имеет взаимозависимость, которая препятствует использованию конвейера или параллелизму. Параллелизм – один из самых важных методов проектирования высокопроизводительных цифровых систем.

**49.корректирующие коды. Код Хэ́мминга. Область применения. Презентация**

**Код Хэ́мминга** — вероятно, наиболее известный из первых самоконтролирующихся и самокорректирующихся [кодов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4). Построен применительно к [двоичной системе счисления](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F). Позволяет исправлять одиночную ошибку (ошибка в одном бите) и находить двойную.

В настоящее время наибольший интерес представляют [двоичные блочные корректирующие коды](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). При использовании таких кодов информация передаётся в виде блоков одинаковой длины и каждый блок кодируется и декодируется независимо друг от друга. Почти во всех блочных кодах символы можно разделить на информационные и проверочные. Таким образом, все комбинации кодов разделяются на разрешенные (для которых соотношение информационных и проверочных символов возможно) и запрещенные.

Основными характеристиками самокорректирующихся кодов являются:

1. Число разрешенных и запрещенных комбинаций. Если n — число символов в блоке, r — число проверочных символов в блоке, k — число информационных символов, то  2n — число возможных кодовых комбинаций, 2k — число разрешенных кодовых комбинаций,  2n **-** 2k — число запрещенных комбинаций.

2. Избыточность кода. Величину k/n называют избыточностью корректирующего кода.

3. Минимальное кодовое расстояние. Минимальным кодовым расстоянием d называется минимальное число искаженных символов, необходимое для перехода одной разрешенной комбинации в другую.

4. Число обнаруживаемых и исправляемых ошибок. Если g — количество ошибок, которое код способен исправить, то необходимо и достаточно, чтобы 

5. Корректирующие возможности кодов.

* [Граница Плоткина](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0_%D0%9F%D0%BB%D0%BE%D1%82%D0%BA%D0%B8%D0%BD%D0%B0) даёт верхнюю границу кодового расстояния
* Есть еще [Граница Хемминга](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0_%D0%A5%D1%8D%D0%BC%D0%BC%D0%B8%D0%BD%D0%B3%D0%B0) устанавливает максимально возможное число разрешенных кодовых комбинаций и [Граница Варшамова — Гилберта](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D1%80%D0%B0%D0%B2%D0%B5%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%93%D0%B8%D0%BB%D1%8C%D0%B1%D0%B5%D1%80%D1%82%D0%B0_%E2%80%94_%D0%92%D0%B0%D1%80%D1%88%D0%B0%D0%BC%D0%BE%D0%B2%D0%B0) для больших n определяет нижнюю границу числа проверочных символов.

Все вышеперечисленные оценки дают представление о **верхней границе** d при фиксированных n и k или **оценку снизу** числа проверочных символов

**Код Хэмминга**

Построение кодов Хэмминга основано на принципе проверки на четность числа единичных символов: к последовательности добавляется такой элемент, чтобы число единичных символов в получившейся последовательности было четным.

S=0 — ошибки нет, S=1 -  однократная ошибка.

Такой код называется (k+1, k)  или (n, n-1) . Первое число — количество элементов последовательности, второе — количество информационных символов.

Для каждого числа проверочных символов r = 3, 4, 5, …   существует классический код Хэмминга с маркировкой  (n,k) = (2r – 1, 2r – 1 - r) то есть — (7,4), (15,11), (31,26) . При иных значениях k получается так называемый усеченный код, например международный телеграфный код МТК-2, у которого k=5 . Для него необходим код Хэмминга (9,5) , который является усеченным от классического  (15,11), .

Для примера рассмотрим классический код Хемминга  (7,4). Сгруппируем проверочные символы следующим образом:

Получение кодового слова выглядит следующим образом:

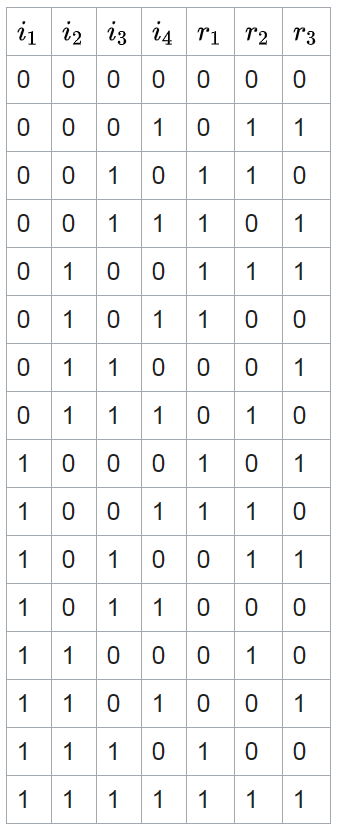
На вход декодера поступает кодовое слово , где штрихом помечены символы, которые могут исказиться в результате помехи. В декодере в режиме исправления ошибок строится последовательность синдромов:

 S=(S1, S2, S3)называется синдромом последовательности.

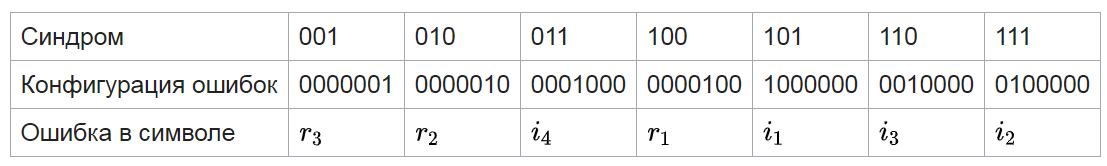
Получение синдрома выглядит следующим образом:

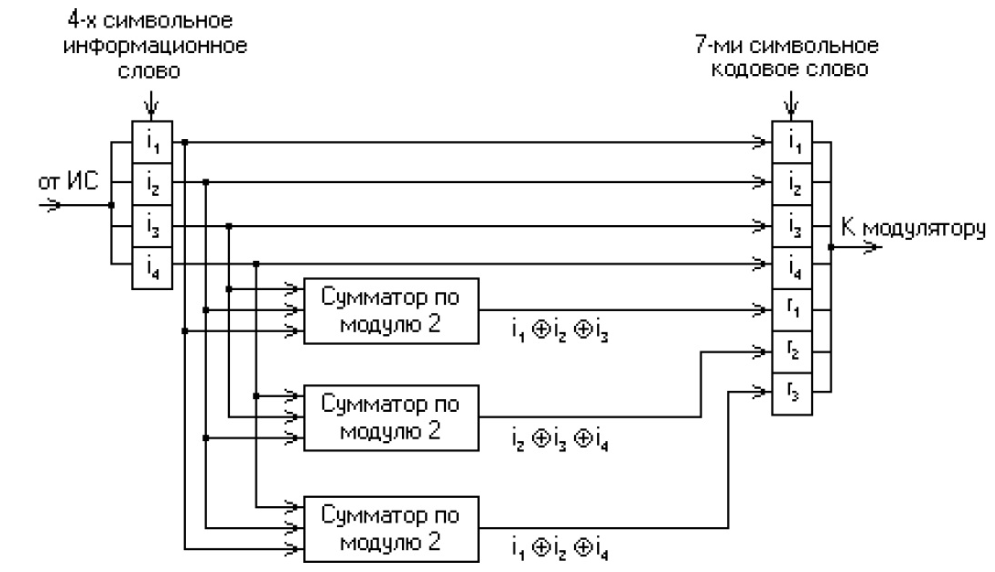
Кодовые слова   кода Хэмминга

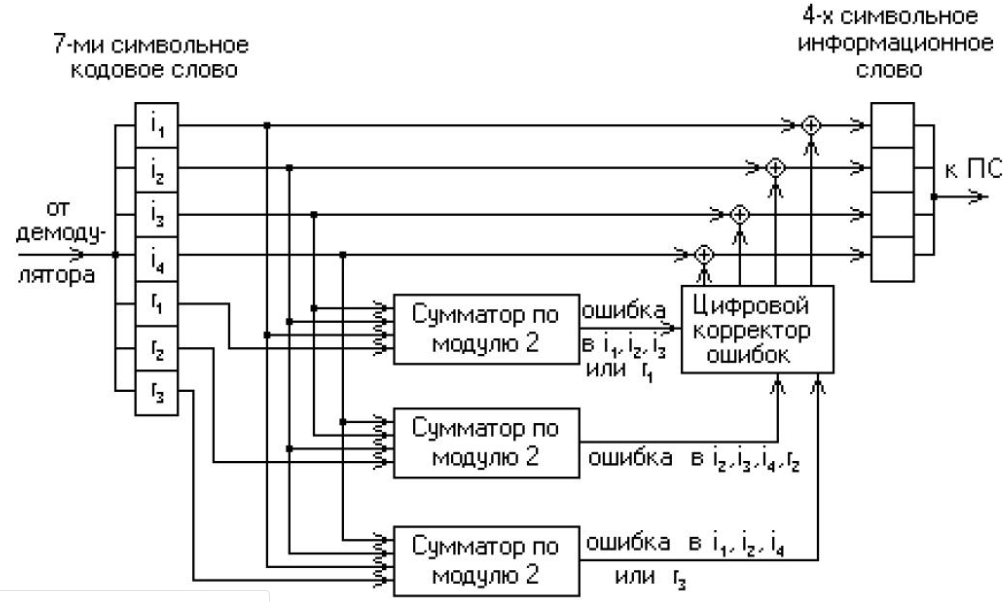
Синдром (0,0,0)  указывает на то, что в последовательности нет искажений. Каждому ненулевому синдрому соответствует определенная конфигурация ошибок, которая исправляется на этапе декодирования.



Для кода  (7,4)  в таблице указаны ненулевые синдромы и соответствующие им конфигурации ошибок (для вида: *i1,i2 ,i3 ,i4 ,r1 ,r2 ,r3 ,*).







**Алгоритм кодирования**

Предположим, что нужно сгенерировать код Хэмминга для некоторого информационного кодового слова. В качестве примера возьмём 15-битовое кодовое слово **x**1…**x**15, хотя алгоритм пригоден для кодовых слов любой длины. В приведённой ниже таблице в первой строке даны номера позиций в кодовом слове, во второй — условное обозначение битов, в третьей — значения битов.



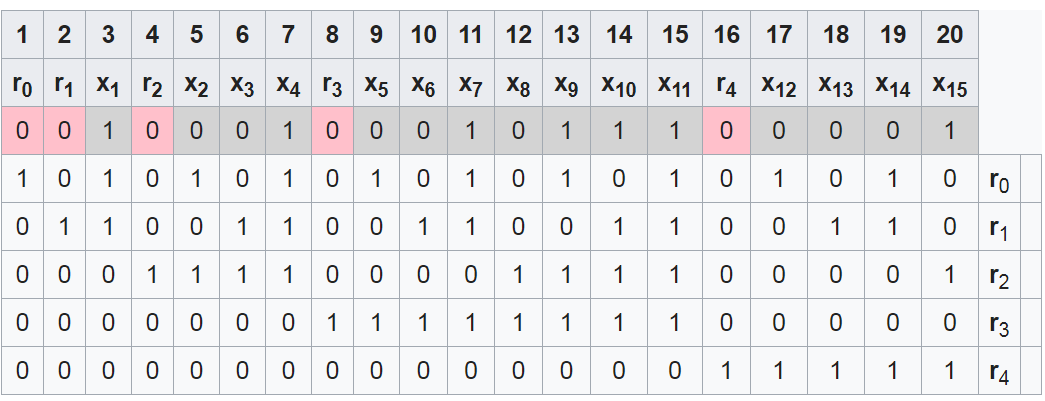
Вставим в информационное слово контрольные биты **r**0…**r**4 таким образом, чтобы номера их позиций представляли собой целые степени двойки: 1, 2, 4, 8, 16… Получим 20-разрядное слово с 15 информационными и 5 контрольными битами.

Первоначально контрольные биты устанавливаем равными нулю. На рисунке контрольные биты выделены розовым цветом.



В общем случае количество контрольных бит в кодовом слове равно двоичному логарифму числа, на единицу большего, чем количество бит кодового слова (включая контрольные биты); логарифм округляется в большую сторону. Например, информационное слово длиной 1 бит требует двух контрольных разрядов, 2-, 3- или 4-битовое информационное слово — трёх, 5…11-битовое — четырёх, 12…26-битовое — пяти и т. д.

Добавим к таблице 5 строк (по количеству контрольных битов), в которые поместим матрицу преобразования. Каждая строка будет соответствовать одному контрольному биту (нулевой контрольный бит — верхняя строка, четвёртый — нижняя), каждый столбец — одному биту кодируемого слова. В каждом столбце матрицы преобразования поместим двоичный номер этого столбца, причём порядок следования битов будет обратный — младший бит расположим в верхней строке, старший — в нижней. Например, в третьем столбце матрицы будут стоять числа 11000, что соответствует двоичной записи числа три: 00011.



В правой части таблицы мы оставили пустым один столбец, в который поместим результаты вычислений контрольных битов. Вычисление контрольных битов производим следующим образом. Берём одну из строк матрицы преобразования (например, r0) и находим её скалярное произведение с кодовым словом, то есть перемножаем соответствующие биты обеих строк и находим сумму произведений. Если сумма получилась больше единицы, находим остаток от его деления на 2. Иными словами, мы подсчитываем сколько раз в кодовом слове и соответствующей строке матрицы в одинаковых позициях стоят единицы и берём это число по модулю 2.

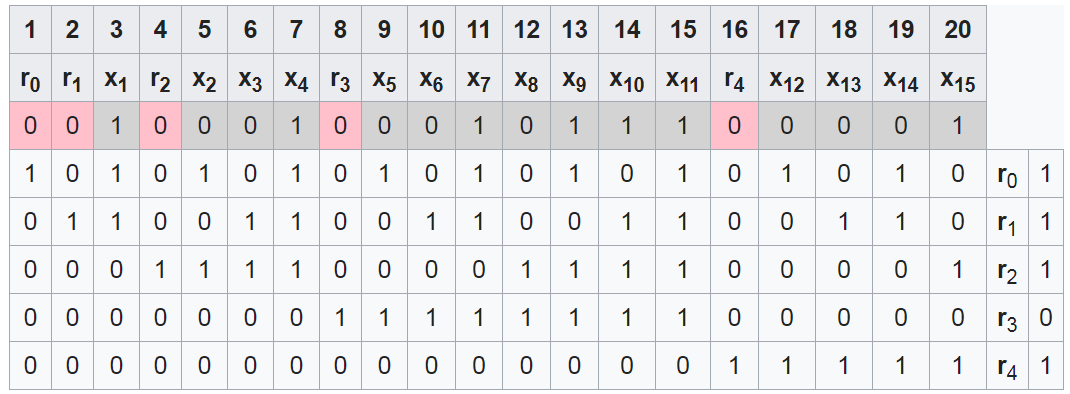
Если описывать этот процесс в терминах матричной алгебры, то операция представляет собой перемножение матрицы преобразования на матрицу-столбец кодового слова, в результате чего получается матрица-столбец контрольных разрядов, которые нужно взять по модулю 2.

Например, для строки r0:

r0 = (1·0+0·0+1·1+0·0+1·0+0·0+1·1+0·0+1·0+0·0+1·1+0·0+1·1+0·1+1·1+0·0+1·0+0·0+1·0+0·1)

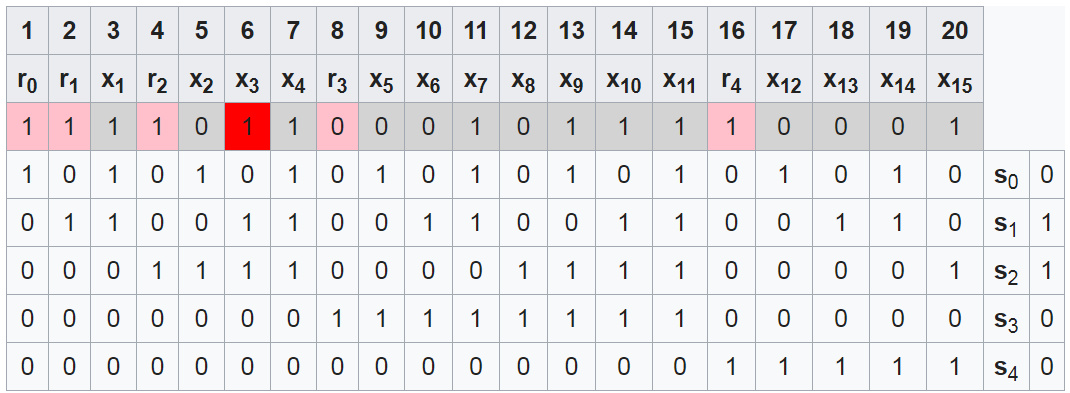
mod 2 = 5 mod 2 = 1.

Полученные контрольные биты вставляем в кодовое слово вместо стоявших там ранее нулей. По аналогии находим проверочные биты в остальных строках. Кодирование по Хэммингу завершено. Полученное кодовое слово — 11110010001011110001.



**Алгоритм декодирования**

Алгоритм декодирования по Хэммингу абсолютно идентичен алгоритму кодирования. Матрица преобразования соответствующей размерности умножается на матрицу-столбец кодового слова и каждый элемент полученной матрицы-столбца берётся по модулю 2. Полученная матрица-столбец получила название «матрица синдромов». Легко проверить, что кодовое слово, сформированное в соответствии с алгоритмом, описанным в предыдущем разделе, всегда даёт нулевую матрицу синдромов.



* Матрица синдромов становится ненулевой, если в результате ошибки (например, при передаче слова по линии связи с шумами) один из битов исходного слова изменил своё значение. Предположим для примера, что в кодовом слове, полученном в предыдущем разделе, шестой бит изменил своё значение с нуля на единицу (на рисунке обозначено красным цветом). Тогда получим следующую матрицу синдромов.

Заметим, что при однократной ошибке матрица синдромов всегда представляет собой двоичную запись (младший разряд в верхней строке) номера позиции, в которой произошла ошибка. В приведённом примере матрица синдромов (01100) соответствует двоичному числу 00110 или десятичному 6, откуда следует, что ошибка произошла в шестом бите.

**Применение**

Код Хэмминга используется в некоторых прикладных программах в области хранения данных, особенно в [RAID 2](https://ru.wikipedia.org/wiki/RAID); кроме того, метод Хэмминга давно применяется в памяти типа [ECC](https://ru.wikipedia.org/wiki/ECC) и позволяет «на лету» исправлять однократные и обнаруживать двукратные ошибки.

**RAID** ([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) *Redundant Array of Independent Disks* — *избыточный*[*массив*](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D0%BE%D0%B2%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2)*независимых*[*дисков*](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)*)* — технология виртуализации данных, которая объединяет несколько дисков в логический элемент для избыточности и повышения производительности.

**ECC-память** ([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) *error-correcting code memory*, память с коррекцией ошибок) — тип [компьютерной памяти](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%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C), которая автоматически распознаёт и исправляет спонтанно возникшие изменения (ошибки) [битов](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) памяти. Память не поддерживающая коррекцию ошибок, обозначается **non-ECC**.

# **50.Языки описания аппаратуры. ПЛИС (FPGA) модули**

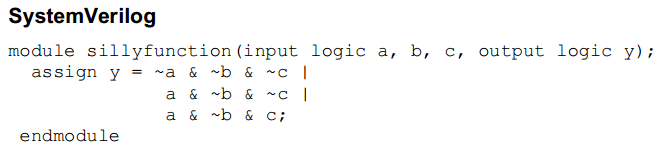
*Языки описания аппаратуры*

* До сих пор мы рассматривали разработку комбинационных и последовательностных цифровых схем на уровне схемотехники. Процесс поиска наилучшего набора логических элементов для выполнения данной логической функции трудоемок и чреват ошибками, так как требует упрощения логических таблиц или выражений и перевода конечных автоматов в вентили вручную. В 1990-е годы разработчики обнаружили, что их производительность труда резко возрастала, если они работали на более высоком уровне абстракции, определяя только логическую функцию и предоставляя создание оптимизированных логических элементов системе автоматического проектирования (САПР). Два основных языка описания аппаратуры (Hardware Description Language, HDL) – SystemVerilog и VHDL.
* SystemVerilog и VHDL построены на похожих принципах, но их синтаксис весьма различается. Их обсуждение в этой главе разделено на две колонки для сравнения, где SystemVerilog будет слева, а VHDL – справа. При первом чтении сосредоточьтесь на одном из языков. Как только вы разберетесь с одним, при необходимости вы сможете быстро усвоить другой. В последующих главах показана аппаратура и в схематическом виде и в форме HDL-модели. Если вы предпочтете пропустить эту главу и не изучать языки описания цифровой аппаратуры, вы тем не менее сможете постичь принципы архитектуры микропроцессоров на уровне схем. Однако, подавляющее большинство коммерческих систем сейчас строится с использованием языков описания цифровой аппаратуры, а не на уровне схемотехники. Если вы когда-либо в вашей карьере собираетесь заниматься разработкой цифровых схем, мы настоятельно рекомендуем вам выучить один из языков описания аппаратуры.

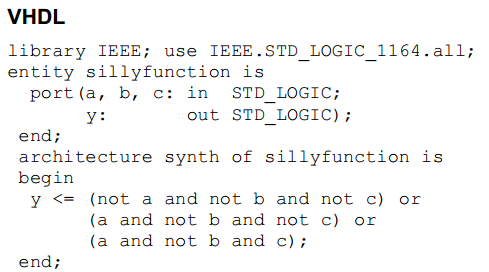
*Модули*

* Блок цифровой аппаратуры, имеющий входы и выходы, называется модулем. Логический элемент “И”, мультиплексор и схема приоритетов являются примерами модулей цифровой аппаратуры. Есть два общепринятых типа описания функциональности модуля – поведенческий и структурный. Поведенческая модель описывает, что модуль делает. Структурная модель описывает то, как построен модуль из простых элементов, с применением принципа иерархии. Код на SystemVerilog и VHDL из примера 4.1 показывает поведенческое описание модуля, который рассчитывает булеву функцию. На обоих языках модуль назван sillyfunction и имеет 3 входа, a, b и c и один выход y, и, как и следовало ожидать, следует принципу модульности. Он имеет полностью определенный интерфейс, состоящий из его входов и выходов, и выполняет определенную функцию. Конкретный способ, которым модуль был описан, неважен для тех, кто будет использовать модуль в будущем, поскольку модуль выполняет свою функцию.





* Модуль на SystemVerilog начинается с имени модуля и списка входов и выходов. Оператор assign описывает комбинационную логику. Тильда (~) означает НЕ, амперсанд (&) – И, а вертикальная черта (|) – ИЛИ. Сигналы типа logic, как входы и выходы в примере – логические переменные, принимающие значения 0 или 1. Они также могут принимать плавающее и неопределенное значения. Тип logic появился в SystemVerilog. Он введен для замены типа reg. Тип logic стоит использовать везде, кроме описания сигналов с несколькими источниками. Такие сигналы называются цепями (net)
* Код на VHDL состоит из трех частей: объявления используемых библиотек и внешних объектов (library, use), объявления интерфейса объекта (entity) и его внутренней структуры (architecture). В объявлении интерфейса указывается имя модуля и перечисляются его входы и выходы. Блок architecture определяет, что модуль делает. У сигналов в VHDL, в том числе входов и выходов, должен быть указан тип. Цифровые сигналы стоит объявлять как STD\_LOGIC. Сигналы этого типа принимают значения ‘0’ или ‘1’, а также плавающее и неопределенное значения. Тип STD\_LOGIC определен в библиотеке IEEE.STD\_LOGIC\_1164, поэтому библиотеку объявлять обязательно. VHDL не определяет соотношение приоритетов операций AND и OR, поэтому при записи логических выражений нужно всегда использовать скобки.



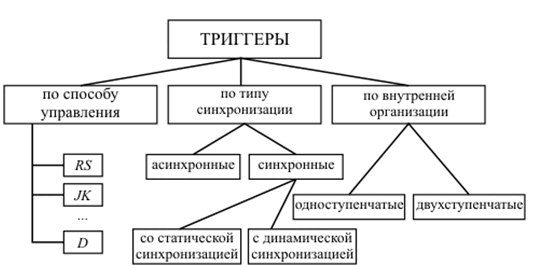


# **51. Типы триггеров. Классификация триггеров. RS-триггер на элементах И-НЕ и ИЛИ–НЕ. T-, JK-, D-триггеры**

**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:

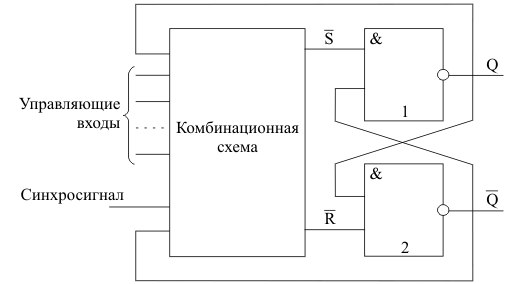
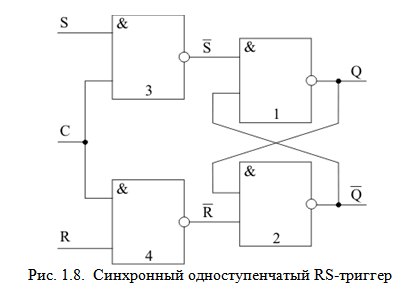
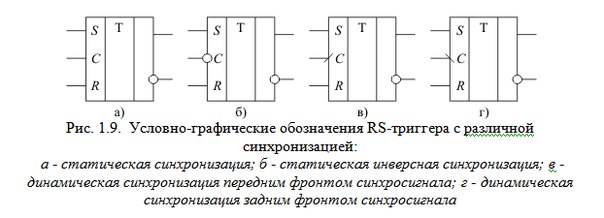


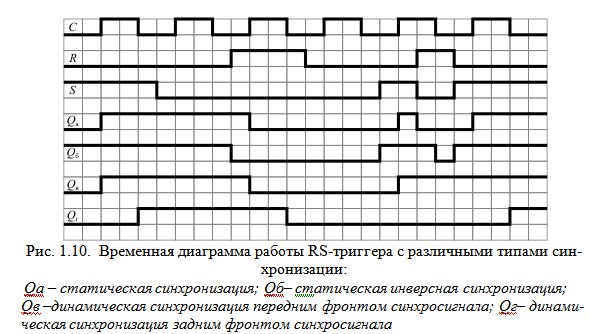
Рис. 1.7. Обобщенная схема синхронного одноступенчатого триггера

Схема синхронного одноступенчатого RS-триггера:





Идеализированная (без учета задержек) временная диаграмма работы RS-триггеров с различными типами синхронизации:



**2.1 RS-триггер**

Основным триггером, на котором базируются все остальные триггеры является RS-триггер.

RS-триггер имеет два логических входа:

R - установка 0 (от слова reset);

S - установка 1 (от слова set).

RS-триггер имеет два выхода:

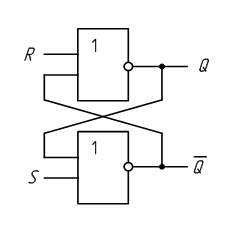
Q - прямой;

· Q(со штрихом) - обратный (инверсный).

Состояние триггера определяется состоянием прямого выхода. Простейший RS-триггер состоит из двух логических элементов, охваченных перекрёстной положительной обратной связью (рисунок 2.1).

Рисунок 2.1 - Схема простейшего RS- триггера

Рассмотрим работу триггера:



Пусть R=0, S=1. Нижний логический элемент выполняет логическую функцию ИЛИ-НЕ, т.е. 1 на любом его входе приводит к тому, что на его выходе будет логический ноль Q=0. На выходе Q будет 1 (Q=1), т.к. на оба входа верхнего элемента поданы нули (один ноль - со входа R, другой - с выхода ). Триггер находится в единичном состоянии. Если теперь убрать сигнал установки (R=0, S=0), на выходе ситуация не изменится, т.к. несмотря на то, что на нижний вход нижнего логического элемента будет поступать 0, на его верхний вход поступает 1 с выхода верхнего логического элемента. Триггер будет находиться в единичном состоянии, пока на вход R не поступит сигнал сброса. Пусть теперь R=1, S=0. Тогда Q=0, а Q(со штрихом)=1. Триггер переключился в "0". Если после этого убрать сигнал сброса (R=0, S=0), то все равно триггер не изменит своего состояния.

Для описания работы триггера используют таблицу состояний (переходов).

Обозначим:

Q(t) - состояние триггера до поступления управляющих сигналов (изменения на входах R и S);

· Q(t+1) - состояние триггера после изменения на входах R и S.

Таблица 2.1 - Таблица переходов RS триггера в базисе ИЛИ-НЕ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **R** | **S** | **Q(t)** | **Q(t+1)** | **Пояснения** |
| 0 | 0 | 0 | 0 | Режим хранения информации R=S=0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | Режим установки единицы S=1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | Режим установки нуля R=1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | \* | R=S=1 запрещённая комбинация |
| 1 | 1 | 1 | \* |

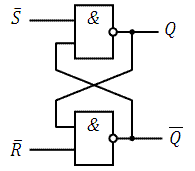
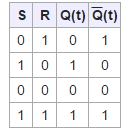
Триггер, построенный на базе элементов ИЛИ-НЕ, называют также дизьюнктивной бистабильной ячейкой. Бистабильные ячейки, помимо самостоятельного применения, входят в качестве составного узла в триггеры других типов.

**Одноступенчатый RS-триггер на элементах И–НЕ.**

**Триггер** (триггерная система) — класс электронных устройств, обладающих способностью длительно находиться в одном из двух устойчивых состояний и чередовать их под воздействием внешних сигналов. Каждое состояние триггера легко распознаётся по значению выходного напряжения. По характеру действия триггеры относятся к импульсным устройствам — их активные элементы (транзисторы, лампы) работают в ключевом режиме, а смена состояний длится очень короткое время.

ВИДЫ ОДНОСТУПЕНЧАТЫХ ТРИГГЕРОВ:

#### RS-триггер асинхронный



.**RS-триггер**, или **SR-триггер** — триггер, который сохраняет своё предыдущее состояние при нулевых входах и меняет своё выходное состояние при подаче на один из его входов единицы.

При подаче единицы на вход **S** (от англ. *Set* — установить) выходное состояние становится равным логической единице. А при подаче единицы на вход **R** (от англ. *Reset* — сбросить) выходное состояние становится равным логическому нулю. Состояние, при котором на оба входа **R** и **S** одновременно поданы логические единицы не определено и зависит от реализации, например в триггере на элементах «или-не» оба выхода переходят в состояние логического «0», которое является неустойчивым и переходит в одно из устойчивых состояний при снятии управляющего сигнала с одного из входов.

RS-триггер используется для создания сигнала с положительным и отрицательным фронтами, отдельно управляемыми посредством стробов, разнесенных во времени. Также RS-триггеры часто используются для исключения так называемого явления дребезга контактов.

RS-триггеры иногда называют RS-фиксаторами

#### RS-триггер синхронный

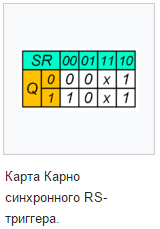
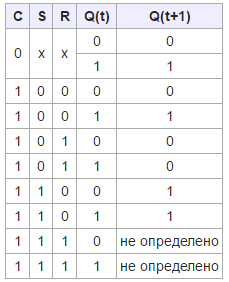
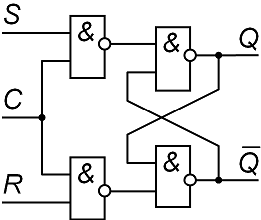


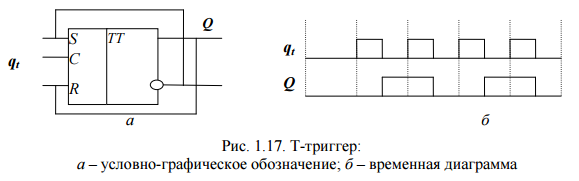
Схема синхронного RS-триггера совпадает со схемой одноступенчатого парафазного (двухфазного) D-триггера, но не наоборот, так как в парафазном (двухфазном) D-триггере не используются комбинации S=0, R=0 и S=1, R=1.

Алгоритм функционирования синхронного RS-триггера можно представить формулой

Безымянный.png где x — неопределённое состояние.

***T-, JK-, D-триггеры.***

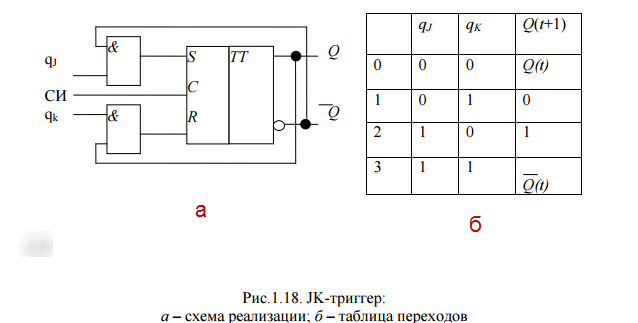
Т-триггер представляет собой триггер, имеющий один вход «Т», поступление единичного сигнала на который переводит Т- триггер в состояние, противоположное его исходному состоянию (фигурально говоря, по каждому входному сигналу триггер «кувыркается», меняя свое состояние на противоположное). На рис. 1.17 приведена реализация Т-триггера на базе двухтактного RS- триггера (а) и временная диаграмма его работы (б). Имеющиеся на схеме обратные связи создают ситуацию, при которой сигналы на входах R и S стремятся перевести триггер в состояние, противоположное текущему. Поэтому при приходе очередного сигнала qT триггер воспринимает сигналы, имеющиеся на его входах. Выходные сигналы триггера изменяются после снятия единичного сигнала на его входе qT, так как триггер двухтактный.



Т-триггер можно рассматривать как счетчик, считающий по модулю два количество импульсов, поступающих на его вход. Действительно, если в исходном состоянии триггер находится в «0», то при поступлении на его вход нечетного количества импульсов триггер будет находиться в «1», а при четном в– «0», что соответствует суммированию по модулю «2» количества поступающих импульсов.

JK-триггер. Реализация JK-триггера и соответствующая таблица истинности приведена на рис. 1.18. Вход «J» – это вход установки «1», вход «K» – вход установки «0».

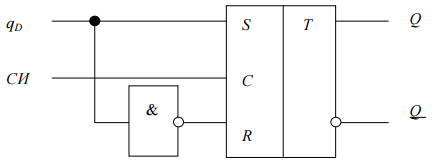
Из приведенной схемы видно, что сигналы qJили qK, стремящиеся установить триггер, соответственно, в «1» или «0», поступают на соответствующий вход «S» или «R» базового RS-триггера только тогда, когда его исходное состояние противоположно тому, в которое стремится перевести JK-триггер комбинация входных сигналов. В противном случае сигналы qJили qK на соответствующий вход S или R базового триггера не поступают. В связи с этим комбинация входных сигналов «1, 1» не является запретной, так как в этом случае на соответствующий вход базового RS-триггера поступит только тот сигнал, который стремится установить триггер в состояние, противоположное его исходному состоянию. Этот момент отражен в таблице истинности:при комбинации входных сигналов «1,1» триггер меняет исходное состояние, то есть работает как T-триггер



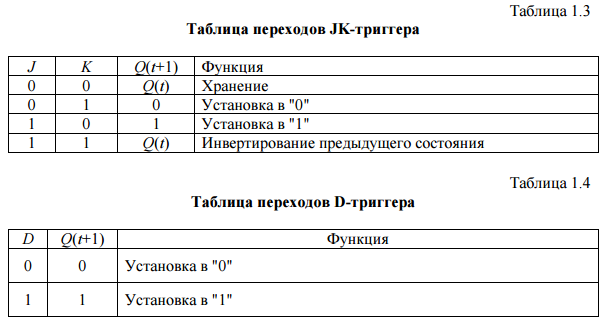
Таким образом, JK-триггер представляет собой универсальный триггер, объединяющий в себе свойства и RS-триггера и Т-триггера.

D-триггер. D-триггер по-другому называют элементом задержки.

Использование подачи сигнала установки «1» через логику НЕ на вход установки «0» приводит к тому, что на входы R и S базового RS-триггера подаются сигналы, имеющие противоположные значения.

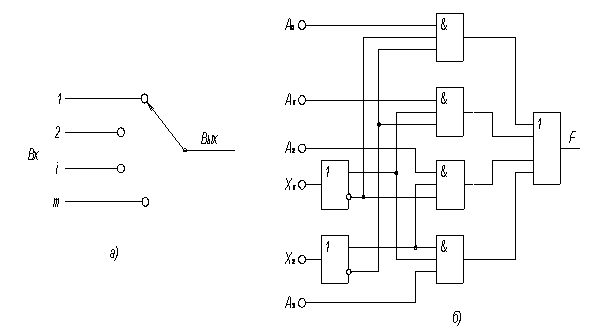


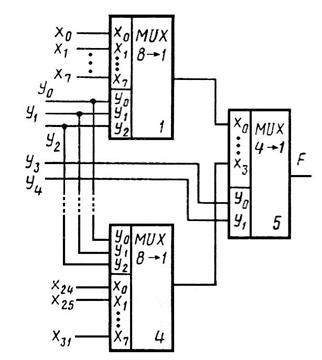
Это означает, что, когда есть «1» на входе qD, на входах базового триггера будут сигналы: «1»–на входе S и «0»–на входе «R». Поэтому по переднему фронту сигнала СИ в триггере устанавливается «1», если есть «1» на входе qD, в противном случае в триггере будет установлен «0». Состояние, которое имеется у триггера в момент заднего фронта сигнала СИ, будет сохраняться («задерживаться») до поступления очередного сигнала синхронизации СИ. Таблицы переходов JK- и D- триггеров приведены в таблиц.1.3 и 1.4 соответственно.



# **52. Параллельные и последовательные регистры. Мультиплексоры и демультиплексоры. Отличия в обозначения цифровых элементов в разных стандартах**

*Мультиплексор* – коммутатор логических сигналов, обеспечивающий передачу информации, поступающей по нескольким входным линиям связи, на одну выходную линию. Выбор вход­ной линии Аi осуществляется в соответствии с поступающим адресным кодом. При наличии m адресных входов можно реализовать M=2m комбинаций адресных сигналов, каждая из которых обеспечивает выбор одной из М вводных линий. Мультиплексор состоит из дешифратора адреса входной линии, схем И и схемы объединения ИЛИ. Функциональная схема мультиплексора приведена на рисунке б. Двоичный код, воздействующий на адресные входа, откроет одну из схем И, которая соединит с выходом соответствующую входную линию. При этом информация на выходе определяется состоянием выбранного входного канала и не зависит от состояния других каналов.

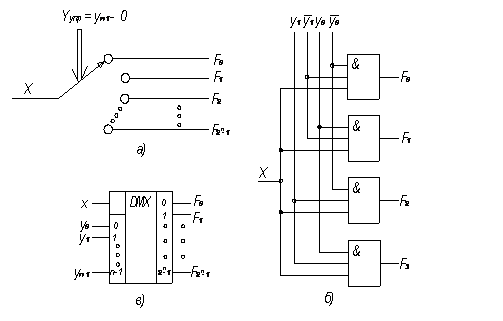


Наращивание размерности мультиплексора.

Мультиплексоры можно использовать для синтеза логических функций от нескольких переменных (x1, x2, …, xn). Если число адресных входов мультиплексора m(адр) , то из общего числа n переменных функции m(адр) можно подать на адресные входы. Тогда на информационные входы мультиплексора через дополнительную логическую схему подаются n-m(адр) переменных. Структуру такой логической схемы можно определить табличным методом или с помощью диаграмм Вейча.

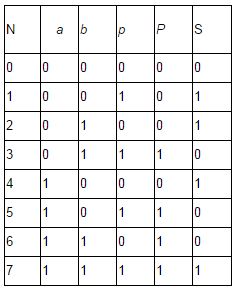
*Демультиплексор* – коммутатор логических сигналов, обеспечивающий передачу информации, поступающей по одному входу, и в соответствии с адресом направляющий в одну из выходных линий.  
При передачах данных по общему каналу с разделением во времени нужны не только мультиплексоры, но и устройства обратного назначения, распределяющие данные из одного канала между несколькими приемниками информации. Эта задача реализуется демультиплексорами, функционирование которых можно пояснить с помощью рисунка а. Демультиплексор имеет один информационный вход n адресующих (управляющих) входов и 2n выходов.  
Построение демультиплексора «1>4» на элементах И показано на рисунке б. Работа демультиплексора описывается логическими выражениями:

, где mi – минтермы n адресующих переменных.

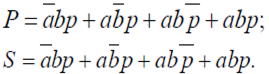
Принцип работы (а), схемная реализация (б) и условное обозначение для демультиплексора (в)  
В условных изображениях демультиплексор обозначается буквами ДМХ (рисунок в).  
Применительно к мультиплексорам и демультиплексорам пользуются также термином «селекторы данных».

**53.Сумматор. Многоразрядный сумматор.**

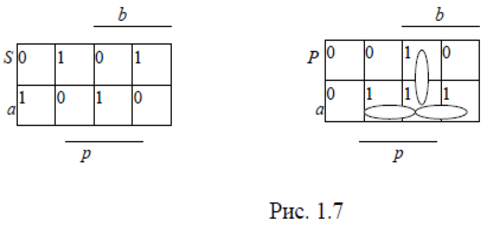
**Одноразрядный двоичный сумматор** обеспечивает сложение одноименных разрядов операндов с учетом переноса, поступающего от ближайшего младшего разряда. Сумматор вырабатывает значение соответствующего **разряда суммы (S) и перенос (P)**, который должен быть учтен в соседнем старшем разряде. Синтез схемы, реализующей функции одноразрядного сумматора, можно выполнить на основании таблицы истинности.



Исходя из реализуемой функции сумматор представляет собой логический узел с двумя выходами (выход суммы S и выход переноса Р) и тремя входами: а - разряд первого операнда; b - разряд второго операнда; р - перенос из младшего разряда. На основании таблицы истинности можно записать логические выражения для формируемых суммы и переноса, которые будут иметь следующий вид:



Полученные функции удобно минимизировать с помощью карты Карно, так как количество переменных невелико. Карты Карно с представленными в них функциями S и P приведены на рис. 1.7.



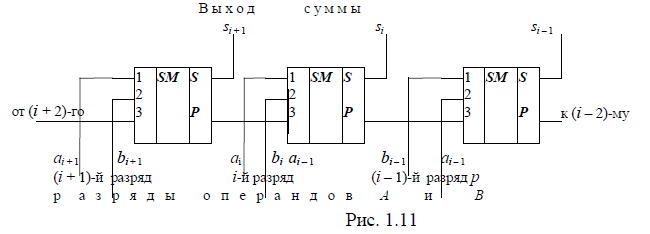
На основании представления функции S на карте можно заключить, что логическое выражение для этой функции не минимизируется.

Минимизированная функция переноса с учетов введенных контуров имеет вид:



Ввиду того что функции P и S формируются в одном и том же узле, при формировании S целесообразно использовать средства, примененные для реализации функции Р. С этой целью рассмотрим функцию Р как переменную для функции S. Тогда модифицированная функция S, зависящая теперь от четырех переменных a, b, p, P, будет записываться в карту Карно как функция четырех переменных.

**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.

**54. Устройства ЭВМ. Состав АЛУ. Типы памяти**.

**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;

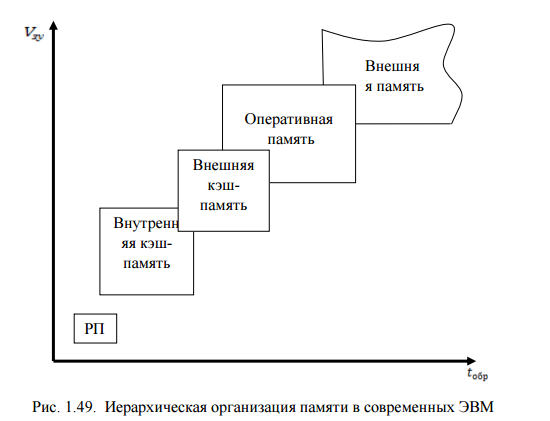
– генерирование поразрядного переноса;

– 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.

**Типы памяти** *Компьютерная память* обеспечивает поддержку одной из наиважнейших функций совр

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.

еменного компьютера, - способность длительного хранения информации

*Все персональные компьютеры используют три вида памяти: оперативную, постоянную и внешнюю (различные накопители).*

Внутренняя память компьютера - это место хранения информации, с которой он работает. Внешняя память (различные накопители) предназначена для долговременного хранения информации

Наиболее знакомы средства машинного хранения данных, используемые в персональных компьютерах: - это модули оперативной памяти, жесткие диски (винчестеры), дискеты (гибкие магнитные диски), CD или DVD диски, а также устройства флэш-памяти.

*Постоянное запоминающее устройство (ПЗУ*), в котором в частности хранится информация, необходимая для первоначальной загрузки компьютера в момент включения питания. Как очевидно из названия, информация в ПЗУ не зависит от состояния компьютера.

Внешняя память обычно располагается вне центральной части компьютера

К внешней памяти относятся различные магнитные носители (ленты, диски), оптические диски. Внешняя память дешевле внутренней, но ее недостаток в том, что она работает медленнее устройств внутренней памяти.

Существуют диски CD-ROM - диски с однократной записью, стереть или перезаписать их невозможно.

Внешняя память реализуется в виде довольно разнообразных устройств хранения информации и обычно конструктивно оформляется в виде самостоятельных блоков. Сюда, прежде всего, следует отнести накопители на гибких и жестких магнитных дисках (последние несколько жаргонно пользователи часто именуют винчестерами), а также оптические дисководы (устройства для работы с CD ROM).

**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.