**Экзамен ЦУиМП**

Оглавление

[1. Предметодье (как предисловие) 4](#_Toc137472459)

[2 Комбинационная логика 5](#_Toc137472460)

[2.1 Цифровые элементы (О, ТИ, АВ)+ 6](#_Toc137472461)

[2.2 Арифметические операции с числами 8](#_Toc137472462)

[2.2.1 Позиционный vs Непозиционный 8](#_Toc137472463)

[2.2.2 Коды двоичных чисел 8](#_Toc137472464)

[2.2.3 Умножение двоичных чисел 11](#_Toc137472465)

[2.3 Вещественные числа. Хранение вещественного числа в компьютере+ 12](#_Toc137472466)

[2.4 Перевод чисел 14](#_Toc137472467)

[2.5 Таблицы истинности и алгебраические выражения+ 15](#_Toc137472468)

[2.6 Карты Карно 21](#_Toc137472469)

[2.7 Метод Квайна+ 22](#_Toc137472470)

[2.8 Базис 2И-НЕ 24](#_Toc137472471)

[2.9 Базис 2ИЛИ-НЕ+ 25](#_Toc137472472)

[2.10 Шифратор (О, ТИ, АВ) 26](#_Toc137472473)

[2.11 Дешифратор+ 27](#_Toc137472474)

[2.12 ССИ 30](#_Toc137472475)

[2.13 Приоритетный шифратор+ 31](#_Toc137472476)

[2.14 Дешифратор матричного типа 34](#_Toc137472477)

[2.15 Каскадное включение дешифраторов+ 35](#_Toc137472478)

[2.16 Мультиплексор (О, ТИ, АВ) 37](#_Toc137472479)

[2.17 Демультиплексор (О, ТИ, АВ)+ 38](#_Toc137472480)

[2.18 Мажоритарный элемент (О?, ТИ, АВ) 41](#_Toc137472481)

[2.19 Сумматор и полусумматор+ 42](#_Toc137472482)

[2.20 Полувычитатель и вычитатель 48](#_Toc137472483)

[2.20.1 Полувычитатель 48](#_Toc137472484)

[2.20.2 Вычитатель 50](#_Toc137472485)

[2.21 Умножители. Умножитель 2х2, 3х3, 4х4+ 52](#_Toc137472486)

[2.22 Компаратор (О, ТИ, АВ) 54](#_Toc137472487)

[2.23 Быстрый сдвиговый регистр (Barrel shifter)+ 55](#_Toc137472488)

[2.24 АЛУ 57](#_Toc137472489)

[2.25 Статический сбой (static hazard)+ 58](#_Toc137472490)

[2.26 Вопросы на подумать 60](#_Toc137472491)

[2.26.1 ИСКЛ ИЛИ на 4 элементах (базис 2И-НЕ). ИСКЛ ИЛИ в базисе 2ИЛИ-НЕ 61](#_Toc137472492)

[2.26.2 Вычитатель на сумматоре 62](#_Toc137472493)

[2.26.3 Сложите 4-5 одноразрядных чисел. Есть восьмиразрядное число, определить количество единиц в нём 63](#_Toc137472494)

[2.26.4 Компаратор на сумматорах. 64](#_Toc137472495)

[2.26.5 Вариант 1 РК1 65](#_Toc137472496)

[2.26.6 Вариант 3 РК1 66](#_Toc137472497)

[2.26.7 Вариант 4 РК1 67](#_Toc137472498)

[2.26.8 Вариант 7 РК1 68](#_Toc137472499)

[3 Последовательстная логика (логика с памятью) 69](#_Toc137472500)

[3.1 RS-триггер, статический, асинхронный, одноступенчатый 70](#_Toc137472501)

[3.2 RS-триггер, синхронный, статический, одноступенчатый 71](#_Toc137472502)

[3.3 D-триггер (D flip-flop), синхронный, статический, одноступенчатый 72](#_Toc137472503)

[3.4 Динамический D-триггер, синхронный, двухступенчатый 73](#_Toc137472504)

[3.5 T-триггер (счётный) 74](#_Toc137472505)

[3.6 Асинхронный счётчик на D-триггерах. 75](#_Toc137472506)

[3.7 Динамический RS-триггер 76](#_Toc137472507)

[3.8 Вычитающий счётчик (асинхронный) 77](#_Toc137472508)

[3.9 Реверсивный счётчик 78](#_Toc137472509)

[3.10 Счётчик с предустановленными значениями 79](#_Toc137472510)

[3.11 Кольцевой счётчик (синхронный) 80](#_Toc137472511)

[3.12 Кольцевой счётчик Джонсона 81](#_Toc137472512)

[3.13 Кольцевой счётчик (ходит только одна единица) 82](#_Toc137472513)

[3.14 Синхронный счётчик вариант 1 83](#_Toc137472514)

[3.15 Синхронный счётчик вариант 2 84](#_Toc137472515)

[3.16 Регистры. Последовательно-параллельные 85](#_Toc137472516)

[3.17 Регистры. Параллельно-последовательные 86](#_Toc137472517)

[3.18 Регистры параллельно-параллельные 87](#_Toc137472518)

[3.19 Регистры. Циклический регистр. Выбор направления. Предустановка значений. 88](#_Toc137472519)

[3.20 Динамический одноступенчатый D-триггер 89](#_Toc137472520)

[3.21 Динамический D-триггер с асинхронным сбросом, установкой 90](#_Toc137472521)

[3.22 JK-триггер, синхронный, статический 91](#_Toc137472522)

[3.23 Динамический JK-триггер 92](#_Toc137472523)

[3.24 Динамический JK-триггер на D-треггерах 93](#_Toc137472524)

[3.25 T-триггер на JK-триггере 94](#_Toc137472525)

[3.26 Синхронный счётчик на JK-триггере 95](#_Toc137472526)

[3.27 Сброс триггера. Асинхронные, синхронные входы. 96](#_Toc137472527)

[3.28 Задачи на подумать. 97](#_Toc137472528)

[3.28.1 Счётчик в базисе 2И-НЕ 98](#_Toc137472529)

[3.28.2 Имеется кольцевой счётчик. Как загнать «1» или несколько «1» и двигать их по кругу? 99](#_Toc137472530)

# Предметодье (как предисловие)

*Я никогда не учу своих учеников. Я только даю условия, при которых они могут сами учиться*

(Эйнштейн)

*Без примеров невозможно ни правильно учить, ни успешно учиться*

(Пацанский паблик ВК)

Целью данной методы не является создать методу ради методы, хочется искренне всё понять, а это возможно только в том случае, если сделать всё самому. Поэтому при создании вашей части методы старайтесь «научить» читателя чему-то, заставить его попробовать реализовать что-то самостоятельно, ведь только в этом случае можно проникнуться всей красотой цифровой электроники и микропроцессоров, и сделать шаг к торжеству автоматизации рутинных задач и процессов.

Обозначения и сокращения:

* О – обозначение
* ТИ – таблица истинности
* АВ – алгебраические выражения
* ВД – временная диаграмма.

# Комбинационная логика

Комбинационная логика — это раздел цифровой логики, который занимается анализом и проектированием комбинационных схем.

Комбинационные схемы — это логические схемы, в которых выходные значения зависят только от текущих входных значений без учета предыдущих состояний или последовательности входных сигналов, что отличает их от последовательностных схем, в которых следующее состояние зависит от предыдущего.

Комбинационная логика используется для создания цифровых схем, которые выполняют определенные логические операции над входными данными и производят соответствующие выходные результаты. Эти схемы широко применяются в различных устройствах, включая компьютеры, микропроцессоры, телефоны, автомобильные системы и многие другие электронные устройства.

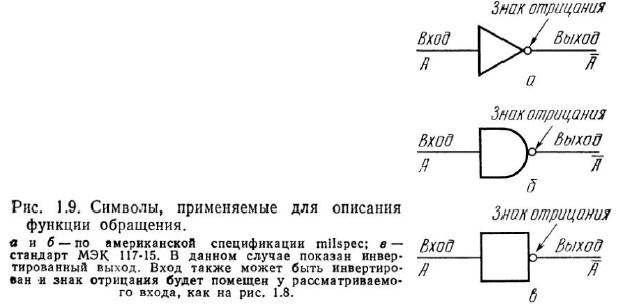
Основная цель комбинационной логики — это создание схем, которые могут выполнять определенные логические операции, такие как логическое И, ИЛИ, НЕ, XOR и другие, и комбинировать эти операции для получения желаемых выходных результатов. Комбинационные схемы также могут выполнять арифметические операции, сравнения, кодирование и декодирование информации, управление данными и т. д.

Преимущества использования комбинационной логики включают быстродействие, надежность, компактность и низкую стоимость. Она позволяет эффективно решать задачи обработки информации и управления в цифровом виде.

## Цифровые элементы (О, ТИ, АВ)+

В системе МЭК (Международная Электротехническая комиссия) используется описание логической функции с помощью квадратной рамки, в которой находится символ данной функции.

1. Инвертор – думаю, здесь комментарии не нужны...ведь да? В алгебраических выражениях инверсия отображается чертой над логическим элементом/логической операцией: .



|  |  |
| --- | --- |
|  |  |
| 0 | 1 |
| 1 | 0 |

2. ИЛИ – «что-нибудь или всё». Пример обозначения приведён для 3ИЛИ, но это не имеет значения, так как мы можем сделать хоть 111111ИЛИ, все они будут собраны через элементы 2ИЛИ.

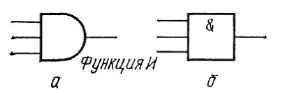
В алгебраических выражениях записывается следующим образом:



|  |  |  |
| --- | --- | --- |
| ­ |  |  |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

3. И – «всё или ничего». Пример обозначения приведён для 3И, но это не имеет значения, так как мы можем сделать хоть 111111И, все они будут собраны через элементы 2И.

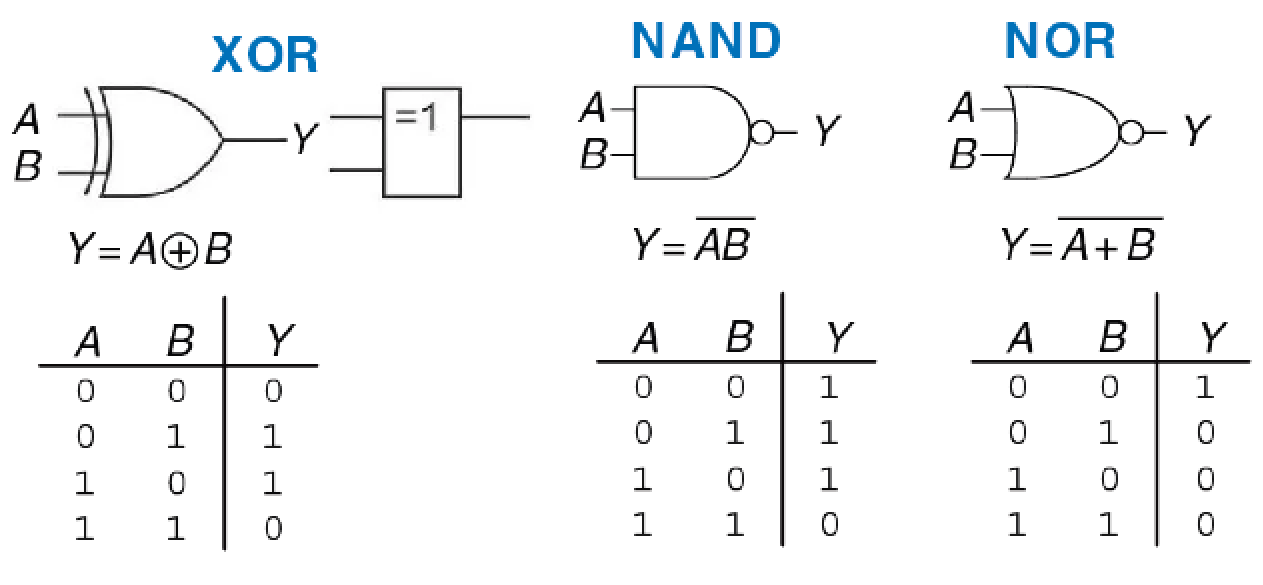
В алгебраических выражениях записывается следующим образом:



|  |  |  |
| --- | --- | --- |
| ­ |  |  |
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

4. XOR («Исключающее ИЛИ»), NAND (2И-НЕ), NOR(2ИЛИ-НЕ)

Из элементов 1-3 можно собрать и выдумать множество других элементов, примеры наиболее распространённых и их таблицы истинности приведены на рисунке ниже:



Исключающее ИЛИ в виде алгебраического выражения можно представить несколькими способами:

Последний вариант может использоваться в картах КАРНО, когда имеет место следующая таблица:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| b | 1 | 0 |
|  | 0 | 1 |

= – Исключающее ИЛИ-НЕ.

## Арифметические операции с числами

### Позиционный vs Непозиционный

В непозиционной системе счисления изменение положения символа в числе не влияет на значение самого числа.

Отличие позиционных и непозиционных систем хорошо видно при сравнении арабских и римских чисел. Числа, записанные арабскими цифрами, составляются в позиционной системе. И здесь важно учитывать понятие разрядности. Одна и та же цифра, в зависимости от того, в каком разряде числа она записывается, обозначает разную числовую величину. Например, в числе 234 цифра 2 обозначает величину двести, а в числе 324 – соответствует двадцати.

В римской системе счисления, цифра, в какое положение ее не помещай, всегда означает одно и то же. Например, с помощью римских цифр V и I, эквивалентных арабским 5 и 1, можно составить числа VI и IV, что соответствует 6 и 4.

### Коды двоичных чисел

[Ссылка на источник](https://microkontroller.ru/programmirovanie-mikrokontrollerov-avr/pryamoy-obratnyiy-dopolnitelnyiy-kod-dvoichnogo-chisla/) – примеры вычитания и сложения

**Прямой код** – способ представления двоичных чисел с фиксированной запятой. Главным образом используется для записи неотрицательных чисел.

Используется для записи:

|  |  |
| --- | --- |
| неотрицательных чисел | как положительных, так и отрицательных чисел |
| Неотрицательные числа в прямом коде  В этом варианте (для восьмибитного двоичного числа) мы можем записать максимальное число 255 (всего чисел 256 - от 0 до 255) | Двоичные числа в прямом коде  В этом случае диапазон десятичных чисел, которые можно записать в прямом коде, составляет от - 127 до +127: |

**Обратный код** – метод вычислительной математики, позволяющий вычесть одно число из другого, используя только операцию сложения.

Обратный двоичный код положительного числа состоит из одноразрядного кода знака (битового знака) – двоичной цифры 0, за которым следует значение числа.

Обратный двоичный код отрицательного числа состоит из одноразрядного кода знака (битового знака) – двоичной цифры 1, за которым следует инвертированное значение положительного числа.

Для неотрицательных чисел обратный код двоичного числа имеет тот же вид, что и запись неотрицательного числа в прямом коде.

Для отрицательных чисел обратный код получается из неотрицательного числа в прямом коде, путем инвертирования всех битов (1 меняем на 0, а 0 меняем на 1). Для преобразования отрицательного числа записанное в обратном коде в положительное достаточного его проинвертировать.



**Дополнительный код** – наиболее распространенный способ представления отрицательных чисел. Он позволяет заменить операцию вычитания на операцию сложения и сделать операции сложения и вычитания одинаковыми для знаковых и беззнаковых чисел.

В дополнительном коде (как и в прямом и обратном) старший разряд отводится для представления знака числа (знаковый бит).

Диапазон десятичных чисел которые можно записать в дополнительном коде от -128 до +127. Запись положительных двоичных чисел в дополнительном коде та же, что и в прямом и обратном кодах.



### Умножение двоичных чисел

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

## Вещественные числа. Хранение вещественного числа в компьютере+

1. Алгоритм представления вещественного числа (числа с плавающей точкой) в компьютере:

1) Перевод числа в двоичную (бинарную) систему счисления.

2) Представление числа в нормализованной записи.

3) Нахождение степени.

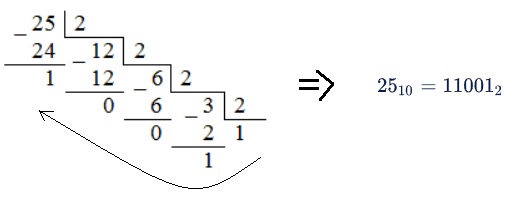
4) Размещение знака, степени и мантиссы в соответствующие разряды.

Рассмотрим этот процесс на примере.

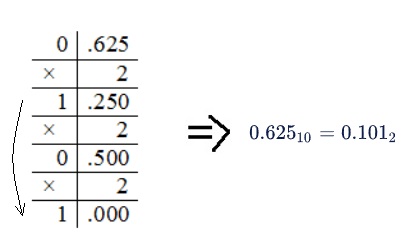
Дано число в десятичной системе счисления -25.625

1) Перевод в bin систему счисления:

а) Переведем целую часть числа:



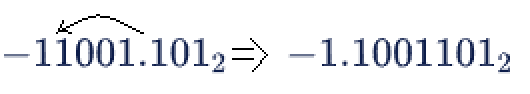
б) Переведем дробную часть числа:



в) Полученное число в бинарном виде:



2) Необходимо сместить точку к «первой» единице в числе:

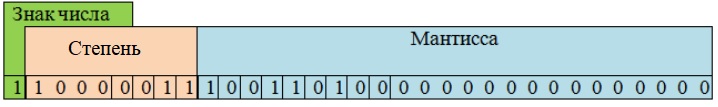


Отсюда получаем , т.к. сместили точку на 4 знака. Мантиссой будет являться часть полученного числа после точки.

3) Степень находим из следующего выражения:

Представляем степень тоже в бинарном виде:

4) Распределяем знак, степень и мантиссу в соответствующие разряды.

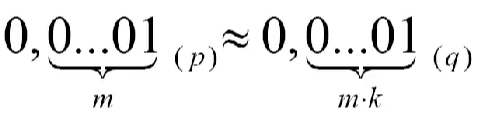


2. Рассмотрим следующую задачу: Задано число, нужно с точностью до Х знаков перевести в двоичную систему счисления. Как узнать число двоичных чисел после запятой, которых будет достаточно для заданной точности?

При переводе числа из десятичной системы счисления с двоичную необходимо ограничить число знаков после запятой при этом сохранив необходимую точность. Согласно теории погрешностей, точность числа считается равной единице младшего знака (разряда).

Так, например, точность для исходного числа 0,02 равна 0,01. Чтобы узнать какой же знак двоичной записи примерно равен 0,01 можно воспользоваться выражением . Отсюда . Отсюда следует что необходимо сохранить 7 знаков после запятой, чтобы точность представления числа не уменьшалась.

В общем случае:

**

Где m – требуемая точность *p*-ичной дроби;

k – показатель, характеризующий степенную зависимость между *p* и *q*, т.е. .

В нашем случае . Отсюда следует Округлив, получаем 7, что и было получено выше.

Также можно использовать следующую формулу:

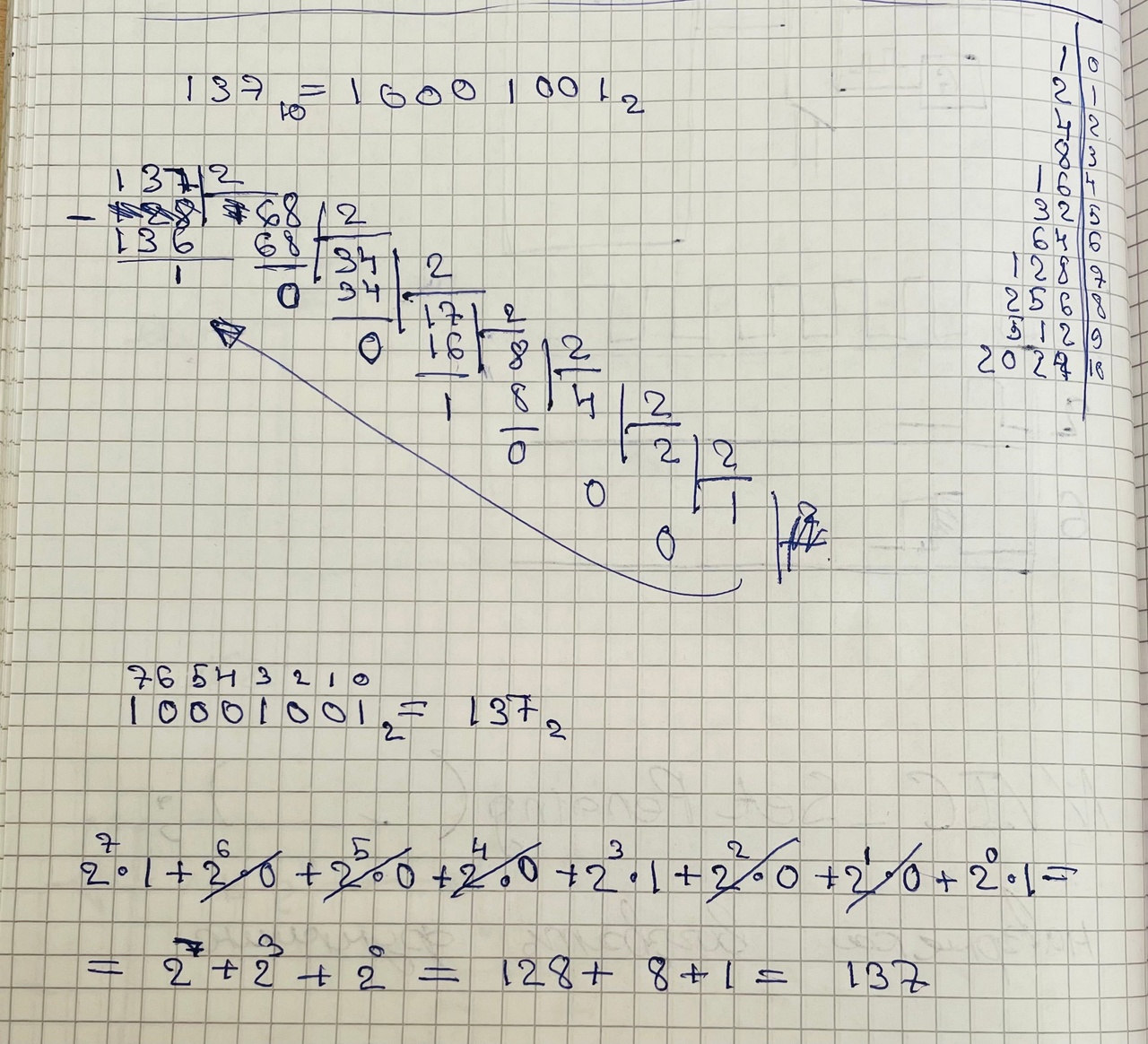
, где – округление до ближайшего целого в большую сторону.

## Перевод чисел

Hex, bin, dec. Почему hex код удобен?

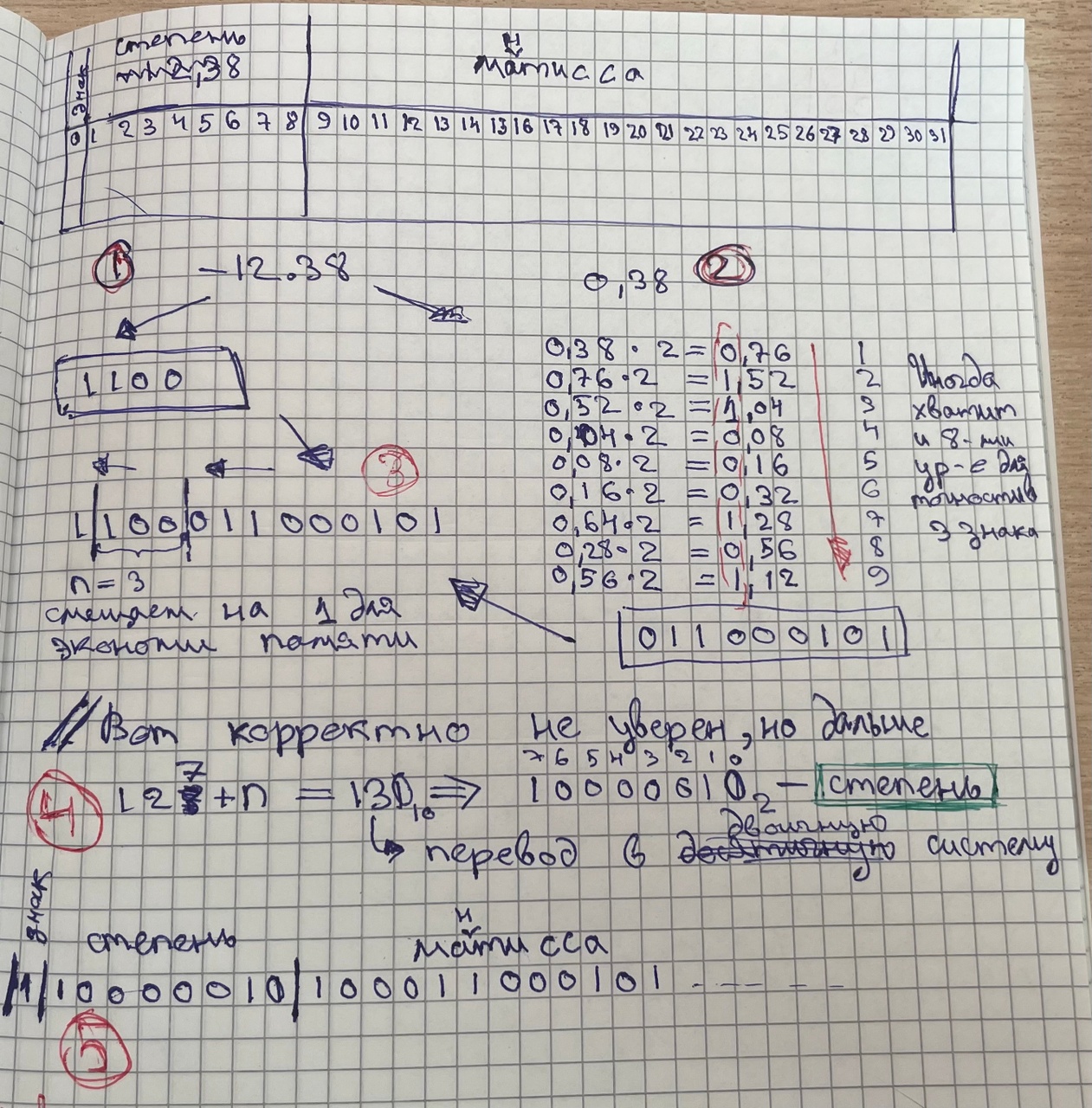
Перевод чисел в двоичную систему счисления из десятичной осуществляется делением его на 2. Деление идет пока остаток не будет равен 1 или 0. Получившееся число снова делим на 2 и так пока не останется 1.

Перевод из двоичной в десятичную систему осуществляется путем номирации двоичного числа с права налево от нуля до n. И эта нумерация будет степенями двойки, каждая двойка в степени умножается на номер i двоичного числа, если это 0, то выбывает соответственно, а потом результат складывается:



Так предствленны unsigned int в языке C.

Числа double представлены иначе, алгоритм перевода представлен ниже:



hex – 16-ричный формат числа 0x*число*

dec – 10-тичный формат числа *число*

bin – 2-ичный формат числа bin'*число*'

16-ричный формат числа удобен тем, что он представолен тетрадой(4-ойкой), это удобно при отладке, программировании.

16-ричный формат 0xFF = 1111 1111 в десятично формате, удобно смотреть и считать по битам.

## Таблицы истинности и алгебраические выражения+

Основные законы алгебры логики (минимизации).

Комбинационные схемы — это логические схемы, в которых выходные значения зависят только от текущих входных значений без учета предыдущих состояний или последовательности входных сигналов. Комбинационная схема есть чёткая зависимость между входом и выходом.

Известно два способа задания логических функций: с помощью формулы и с помощью таблицы истинности. По формуле легко составляется таблица. На практике при конструировании различных электронных устройств часто возникает обратная задача – от таблицы истинности перейти к формуле, чтобы на ее основе построить функциональную схему.

Таблицу истинности можно записать в виде алгебраического выражения. Для этого существует 2 способа:

1) СДНФ, простыми словами, - «сумма произведений», в которых все произведения содержат ВСЕ логические переменные;

2) СКНФ, простыми словами, - «произведение сумм», в которых все суммы содержат ВСЕ логические переменные.

А если не простыми, то вот….

***Элементарной конъюнкцией*** называется конъюнкция нескольких переменных, взятых с отрицанием или без отрицания, причем среди переменных могут быть одинаковые (b).

***Элементарной дизъюнкцией*** называется дизъюнкция нескольких переменных, взятых с отрицанием или без отрицания, причем среди переменных могут быть одинаковые ().

Всякую дизъюнкцию элементарных конъюнкций назовем ***дизъюнктивной нормальной формой*** (ДНФ).

Всякую конъюнкцию элементарных дизъюнкций назовем ***конъюнктивной нормальной формой*** (КНФ).

***Совершенной дизъюнктивной нормальной формой*** (СДНФ) называется ДНФ, в которой нет одинаковых элементарных конъюнкций и все конъюнкции состоят из одного и того же набора переменных, в который каждая переменная входит только один раз (возможно, с отрицанием).

***Совершенной конъюнктивной нормальной формой*** (СКНФ) называется КНФ, в которой нет одинаковых элементарных дизъюнкций и все дизъюнкции состоят из одного и того же набора переменных, в который каждая переменная входит только один раз (возможно, с отрицанием).

**Алгебраические выражения по таблице истинности**

Построим случайную таблицу истинности:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | y |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 |

Запишем алгебраические выражения:

1) СДНФ строится по «1», то есть находим «1» в столбце «» и записываем последовательность соответствующих ей (при этом, если в столбце любого из x стоит «1», то просто записываем , а если «0», то , то есть «не 0» = «1») в виде произведения. Данное действие повторяем для каждой «1» и складываем получившиеся элементарные конъюнкции:

макстерм

2) СКНФ строится по «0», то есть находим «0» в столбце «» и записываем последовательность соответствующих ей (при этом, если в столбце любого из x стоит «0», то просто записываем , а если «1», то , то есть «не 1» = «0») в виде суммы. Данное действие повторяем для каждой «1» и перемножаем получившиеся элементарные дизъюнкции:

минтерм

**Представим обратную задачу: записать таблицу истинности по алгебраическому выражению**

Запишем случайное алгебраическое выражение:

1) СДНФ

Всё просто: берём наши конъюнкции и заполняем строчки таблицы. Записав 1-ую, таблица имеет вид:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | y |
| 0 | 0 | 0 |  |
| 1 | 0 | 0 |  |
| 0 | 1 | 0 |  |
| 1 | 1 | 0 |  |
| 0 | 0 | 1 |  |
| 1 | 0 | 1 |  |
| 0 | 1 | 1 |  |
| 1 | 1 | 1 | 1 |

2-ую:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | y |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 |  |
| 0 | 1 | 0 |  |
| 1 | 1 | 0 |  |
| 0 | 0 | 1 |  |
| 1 | 0 | 1 |  |
| 0 | 1 | 1 |  |
| 1 | 1 | 1 | 1 |

3-ую и 4-ую:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | y |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 |  |
| 0 | 1 | 0 | 1 |
| 1 | 1 | 0 |  |
| 0 | 0 | 1 |  |
| 1 | 0 | 1 | 1 |
| 0 | 1 | 1 |  |
| 1 | 1 | 1 | 1 |

Оставшиеся строчки заполняем «0»:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | y |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 |

2) СКНФ

Всё просто: берём наши дизъюнкции и заполняем строчки таблицы. Записав 1-ую, таблица имеет вид:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | y |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 |  |
| 0 | 1 | 0 |  |
| 1 | 1 | 0 |  |
| 0 | 0 | 1 |  |
| 1 | 0 | 1 |  |
| 0 | 1 | 1 |  |
| 1 | 1 | 1 |  |

2-ую:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | y |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 |  |
| 0 | 1 | 0 |  |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 1 |  |
| 1 | 0 | 1 |  |
| 0 | 1 | 1 |  |
| 1 | 1 | 1 |  |

3-ую и 4-ую:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | y |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 |  |
| 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 |  |
| 0 | 1 | 1 |  |
| 1 | 1 | 1 |  |

Оставшиеся строчки заполняем «1»:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | y |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

**Основные законы минимизации**

1. Коммутативный (переместительный)

2. Ассоциативный (сочетательный)

3. Дистрибутивность (распределительный)

Доказательство последней строчки:

4. Теорема Де Моргана

– штрих Шефера

*–* стрелка Пирса

5. Закон поглощения

6. Склеивание

7. Соотношение, с помощью которого строится ИСКЛЮЧАЩЕЕ-ИЛИ на 4-ёх элементах

Доказательство:

Полезные выражения:

## Карты Карно

Причём тут код Грэя? Перевод бинарных чисел в код Грэя и в обратную сторону. Принцип составления карт Карно для СКНФ и СДНФ. Как можно объединять 1 (0) в ячейках? Что даст диагональное расположение 1 (0)?

## Метод Квайна+

Принцип метода Квайна основан на использовании логических законов и тождеств для упрощения алгебраических выражений. Основная идея заключается в том, чтобы преобразовать выражение в эквивалентное, более простое выражение, используя логические свойства операций и логических законов.

Разберём последовательность операций метода на КНФ и ДНФ алгебраического выражения:

макстерм

* ДНФ

В 1 колонку таблицы запишем все макстермы. Во второй и последующих колонках с помощью операции склеивания будем уменьшать количество выражений до того момента, когда больше не сможем сократить.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 |  | 1-2: (1’) | 1’-2’: (1’’) |  |
| 2 |  | 1-5: (2’) | 3’-4’: c (2’’) |  |
| 3 |  | 2-3: (3’) | (3’’) |  |
| 4 |  | 2-5: (4’) |  |  |
| 5 |  | 3-5: (5’) |  |  |
|  |  | (6’) |  |  |

Далее запишем полученные в последнем столбце выражения в первый столбец новой таблицы. «Заголовками» остальных столбцов будут все изначальные макстермы. Ставим Х в ячейке, когда в макстерме содержится полученное упрощенное выражение из первого столбца. Сначала выбираем колонки, в которых стоит всего один Х. Так как таких колонов для из первого столбца много, то целесообразно выбрать ту, макстерм которой наипростейший, то есть последний столбец. Остальные колонки с Х, встречающиеся на пути при движении справа налево выбранного Х (показано черной стрелкой), вычеркиваются. Упрощённое выражение составляется из выражений первого столбца: если в строке имеется Х, то выписываем макстерм в конечное выражение

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | X | X | X |  | X |
|  |  |  |  | X |  |

* КНФ

Последовательность действий не изменяется и все операции осуществляются аналогичным образом только для минтермов

минтерм

|  |  |  |
| --- | --- | --- |
| 1 |  | 1-3: |
| 2 |  | 2-3: |
| 3 |  | 3-4: |
| 4 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | X |  | X |  |
|  |  | X | X |  |
|  |  |  | X | X |

Разберём ещё один случай:

|  |  |  |
| --- | --- | --- |
| 1 |  | 1-2: (1’) |
| 2 |  | 1-3: (2’) |
| 3 |  | 3-4: (3’) |
| 4 |  | 4-5: (4’) |
| 5 |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | X | X |  |  |  |
|  | X |  | X |  |  |
|  |  |  | X | X |  |
|  |  |  |  | X | X |

Таким образом, «живыми» остались все строчки, но в одном из столбцов имеется несколько «Х», и они никем не «убиваются», поэтому имеем два варианта записи упрощённого алгебраического выражения:

## Базис 2И-НЕ

Базисы нужны для упрощения жизни – когда все одинаковое становится приятно внутреннему перфекционисту и проще разрабатывать новые схемотехнические решения.

|  |  |
| --- | --- |
| Обозначение элемента 2И-НЕ (Штрих Шеффера) | |
|  |  |

|  |  |  |
| --- | --- | --- |
| **X** | **Y** | **X | Y** |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Теорема де Моргана используется для перевода в базис 2И-НЕ

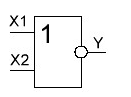


Перевод АВ в базис 2И-НЕ:

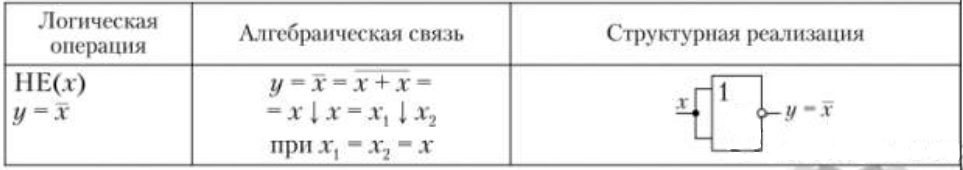
## Базис 2ИЛИ-НЕ+

Необходимость базисов заключается в следующем: цифровое устройство, выполняющее любую логическую функцию, можно построить, имея элементы только одного вида («ИЛИ-HE» либо «И-НЕ»). При интегральной технологии с точки зрения надежности и стоимости использование однотипных элементов весьма предпочтительно.

Элемент 2ИЛИ-НЕ:



Реализация простейших логических операций в базисе 2ИЛИ-НЕ:



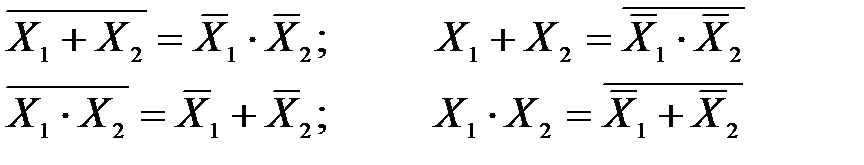


Для перевода АВ в базис 2ИЛИ-НЕ необходимо:

1) Наложить двойную инверсию.

2) Применить теорему де Моргана.

Теорема де Моргана заключается в следующем:

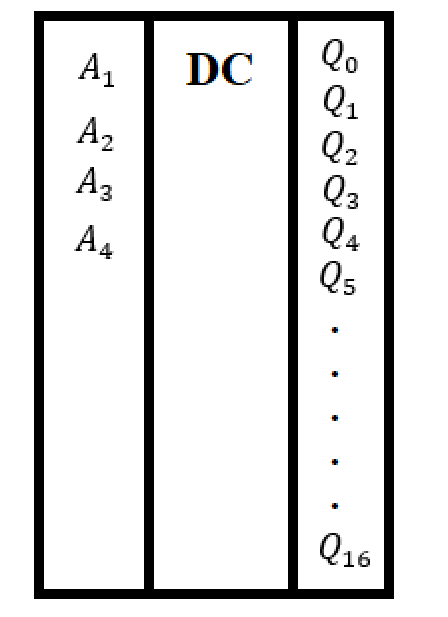


## Шифратор (О, ТИ, АВ)

Что это и зачем нужно? Можно ли использоваться комбинацию входных сигналов с несколькими 1? Обобщённая схема. Схема в базисе 2ИЛИ-НЕ, 2И-НЕ

## Дешифратор+

Дешифратор (DC – decoder) – цифровое устройство, предназначенное для преобразования двоичного кода в десятичный. Обозначается следующим образом:



Количество входов и выходов дешифратора связаны следующим соотношением:

, где n – число выходов, k – число входов

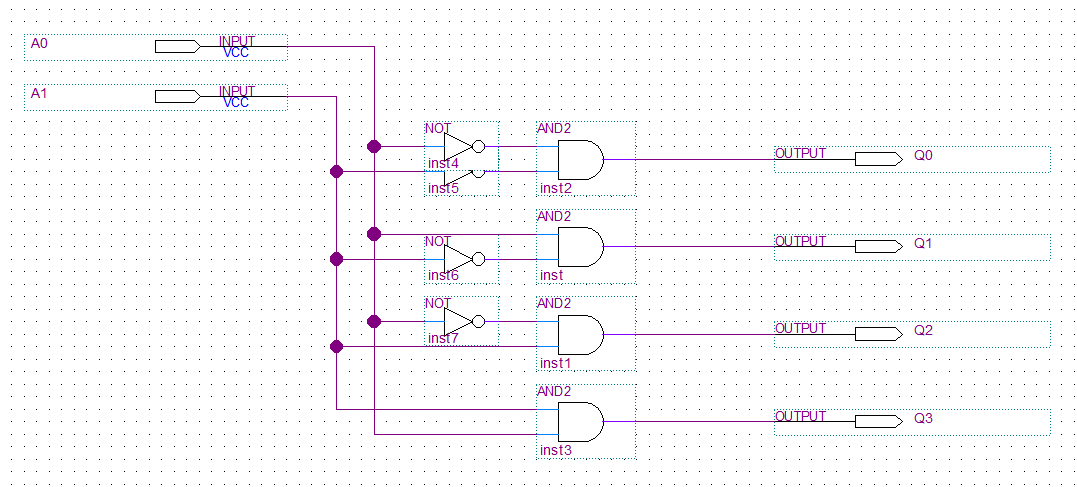
Логика работы: при подаче на входы двоичного числа на выходе с десятичным номером, соответствующим входному двоичному числу в десятичной системе, устанавливается логическая «1».

Составим таблицу истинности для дешифратора с 2-мя входами:

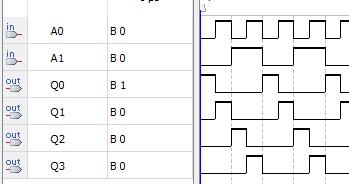
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

Из неё, очевидно:

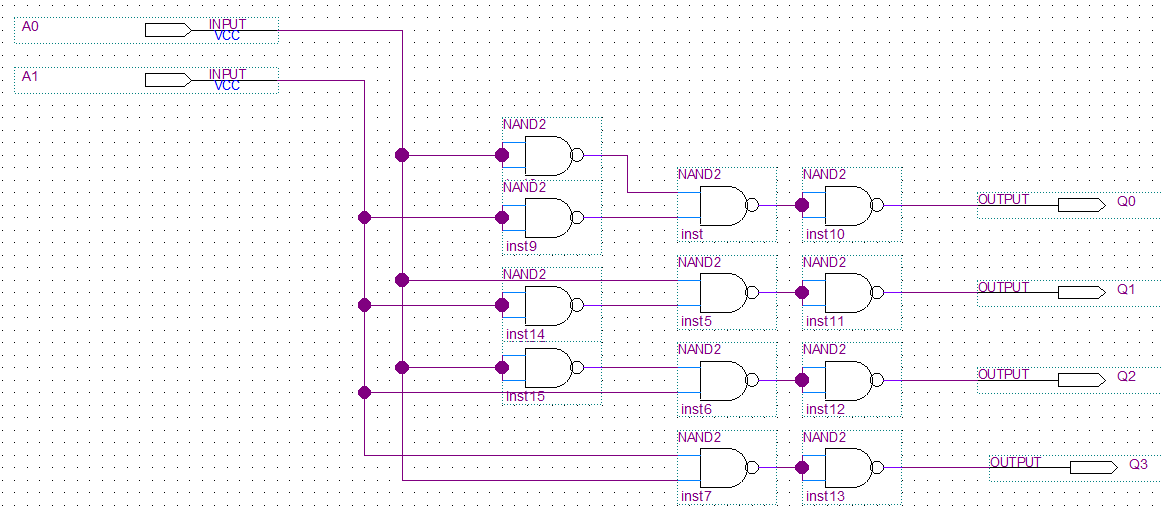
Соответственно схема имеет вид:

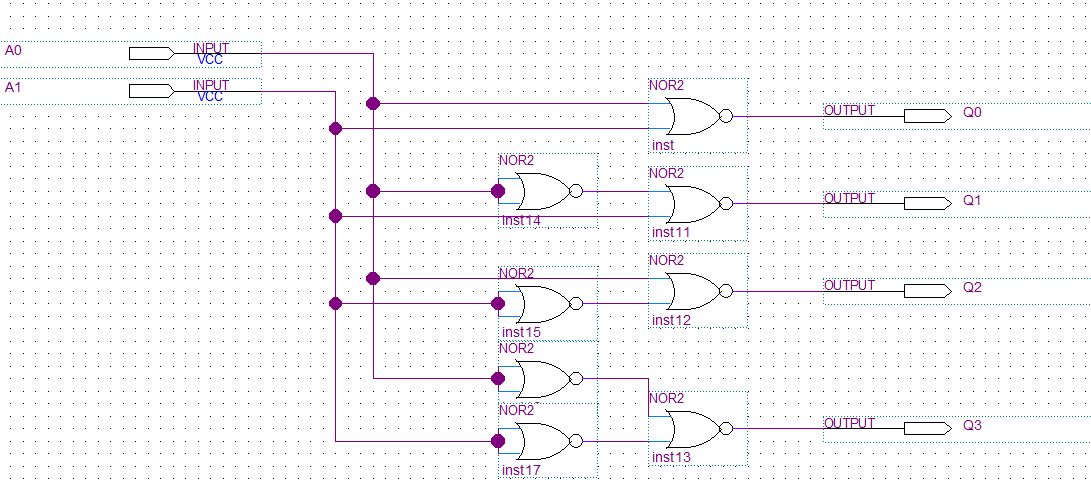


Временная диаграмма:



Представление схемы в базисе 2И-НЕ, 2ИЛИ-НЕ не представляет трудностей. Сделаем это через выражения и затем представим схему:





## ССИ

Устройство, ТИ, Схемы с ОК и ОА. Пример составления выражения для одного индикатора, его минимизации и перевод в базис (любой), схема для индикатора.

## Приоритетный шифратор+

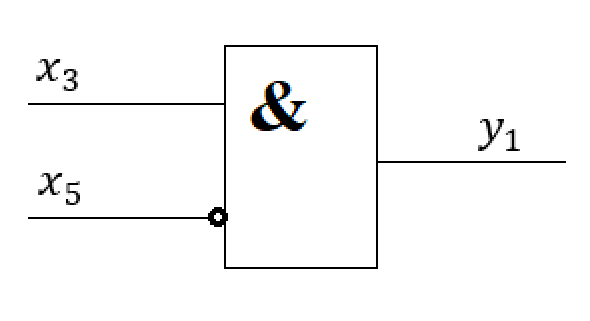
Приоритетный шифратор отличается от шифратора наличием дополнительной логической схемы выделения активного уровня старшего входа для обеспечения условия работоспособности шифратора (только один уровень на входе активный). Уровни сигналов на остальных низших входах схемой игнорируются. То есть говоря простым языком, если, например, на имеем логическую «1», то для нас не имеет значения, что находится на . Таблица истинности имеет следующий вид:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| X | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| X | X | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| X | X | X | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| X | X | X | X | X | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| X | X | X | X | X | X | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| X | X | X | X | X | X | X | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| X | X | X | X | X | X | X | X | 1 | 0 | 0 | 0 | 0 | 1 |
| X | X | X | X | X | X | X | X | X | 1 | 1 | 0 | 0 | 1 |

Х – любое состояние

Схема приоритетного дешифратора строится по следующим принципам:

* При получении высших комбинаций (например, 1010 для ) нас не интересуют низшие входы (), а имеют значения только высшие (,);
* Для исключения влияния входов, к примеру на вход , необходимо при построении схем для «выключать» их, когда на находится логическая «1». То есть, например, имеем помимо логической «1» на логическую «1» на . Тогда, на выходе может образоваться «1» из-за влияния , но если мы применим, например, элемент 2И, то мы исключим такое поведение:



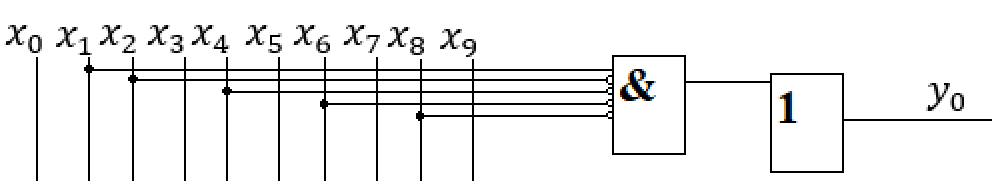
Таким образом, при появлении «1» на не имеет значения уровень , ведь он никак не повлияет на .

Соберём схему приоритетного шифратора для нашей таблицы истинности:

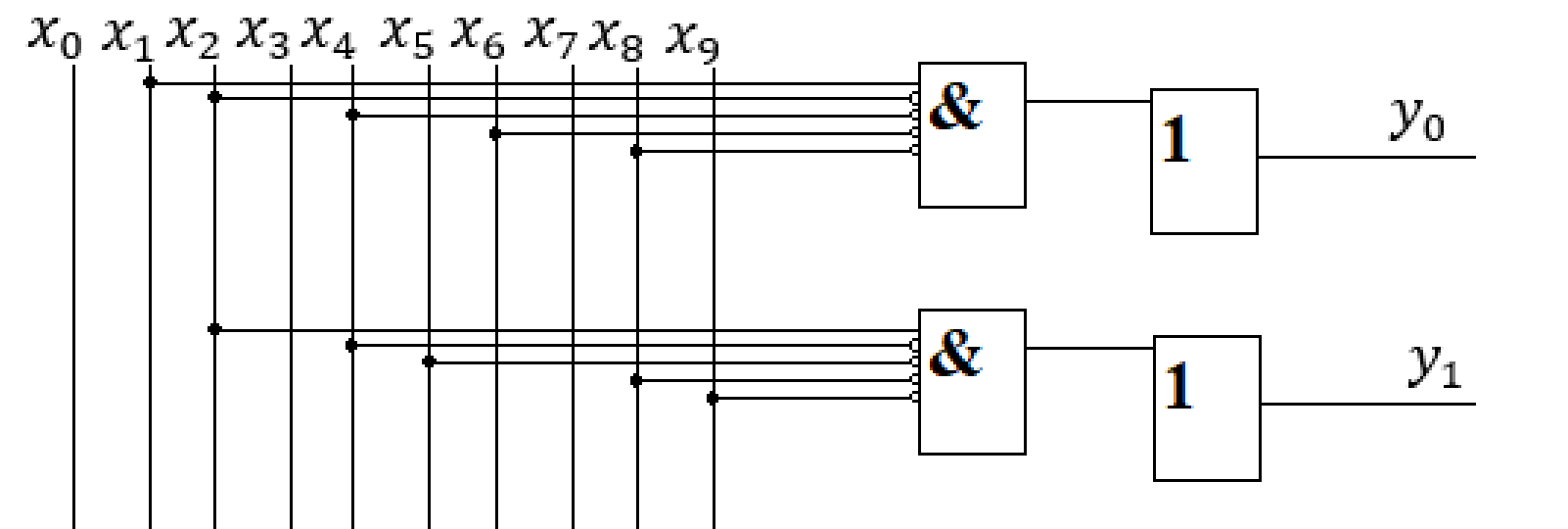
* При «1» на на находятся нули, поэтому схема не нужна
* При «1» на на комбинация 1000, то есть необходима «1» на . Нужна? Ну пожалуйста:



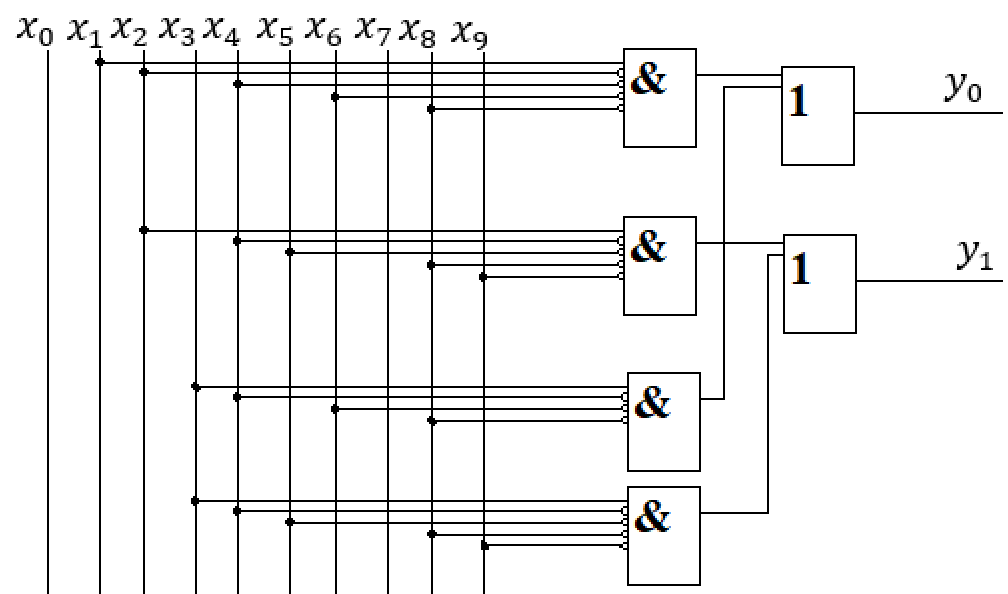
Но в таком случае «1» будет появляться и в тех случаях, когда «1» на и т.д., а её там быть не должно, ведь вход является низшим и значение на нём не должно влиять на высшие входы. Поэтому просто будем исключать влияние на , если на входах находится «1». Почему именно эти входы? Потому что в их выходных комбинациях отсутствует «1» на (смотреть таблицу истинности), когда на входах нам не нужно блокировать влияние «1» с , потому что в их выходных комбинациях и так есть «1» на . Таким образом, элемент для имеет следующий вид:



* Точно по таким же соображениям собираем схему для :

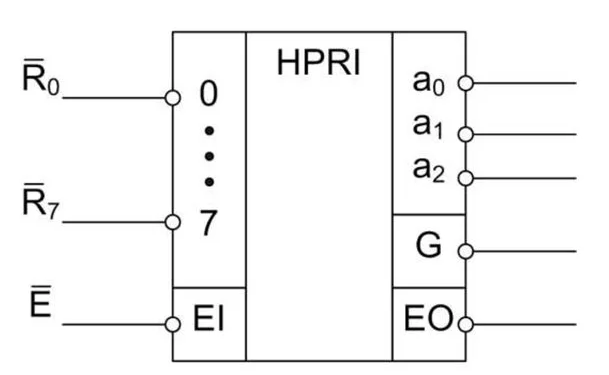


* :



Полная схема приведена в «priority\_encoder.bcd»

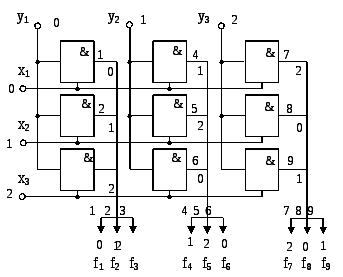
P.s. как обозначается - не нашёл сто процентной информации, но очень часто встречается на различным сайтах и картинкой с подписью HPRI:



## Дешифратор матричного типа

Прикольный шифратор, зачем нужен не ясно, зато красивый.

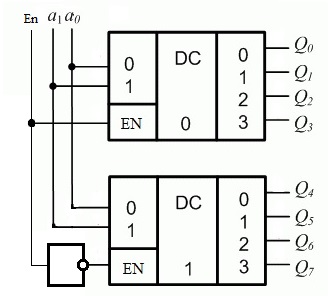
входов, входов.



|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

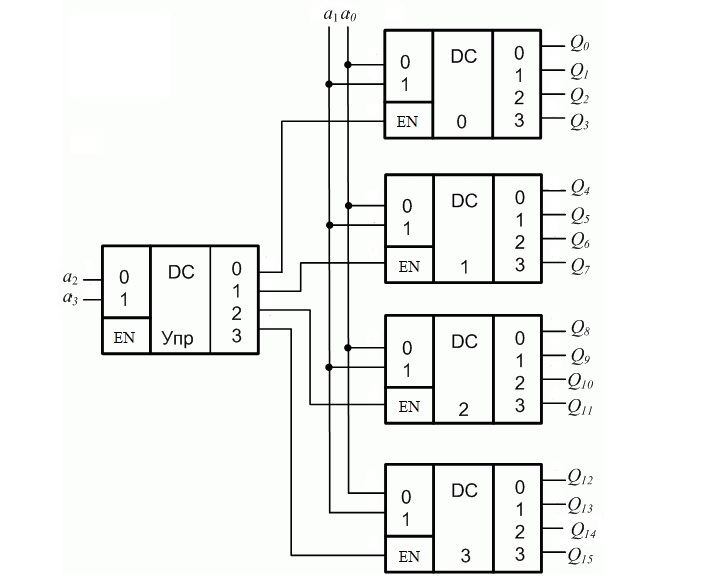
## Каскадное включение дешифраторов+

Необходимость каскадного соединения нескольких дешифраторов возникает в том случае, когда разрядность одной ИС оказывается недостаточной для адресации большого количества различных устройств. Например, в нашем распоряжении схемы дешифратора на 2 входа и, соответственно, 4 выхода. А необходимо организовать возможность обращения к 8 цифровым устройствам. Тогда можем использовать схему каскадного включения двух таких дешифраторов:



Как видим, с помощью EN мы можем выбрать четверку устройств из которых мы будем производить свой выбор в дальнейшем при подаче *a0* и *a1* .

Рассмотрим случай, когда нам необходимо иметь возможность обращения к 16-ти цифровым устройствам, тогда можем использовать следующую схему каскадного включения:



В данном случае используется 5 дешифраторов, один из которых (управляющий) используется для выбора четверки устройств, из которых будет производиться дальнейший выбор с помощью оставшихся дешифраторов.

## Мультиплексор (О, ТИ, АВ)

Что это, зачем? Обобщённая схема, схема в базисах 2И-НЕ, 2ИЛИ-НЕ

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

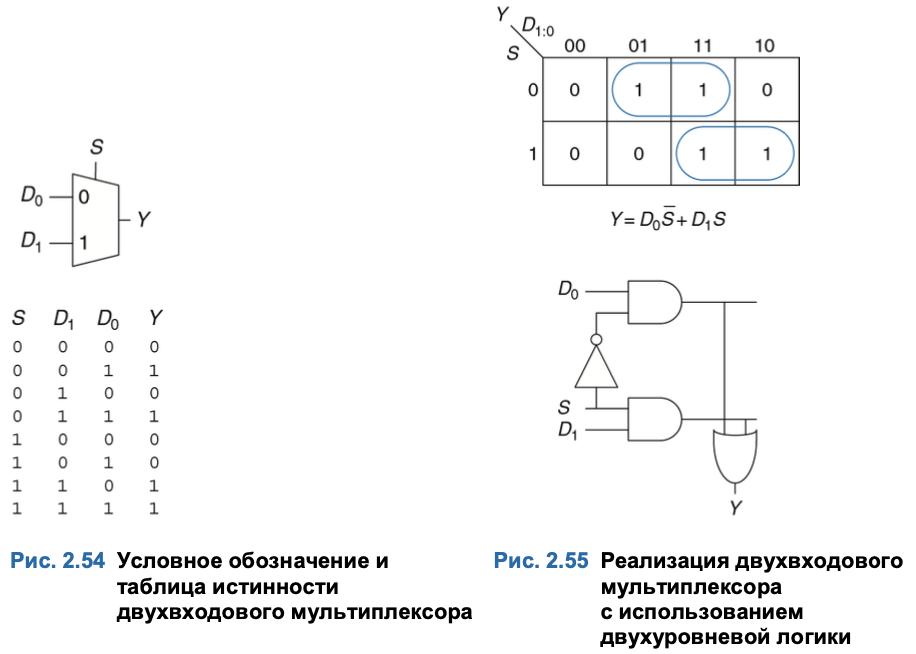
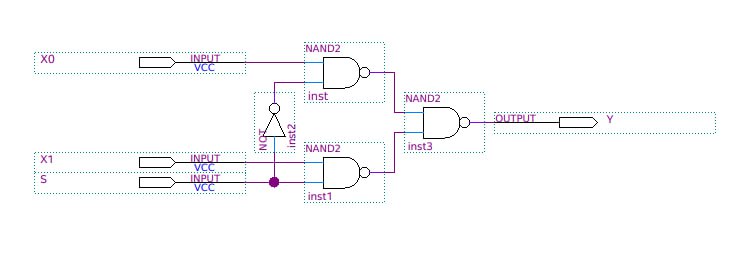




Схема мультиплекстора на 2И-НЕ



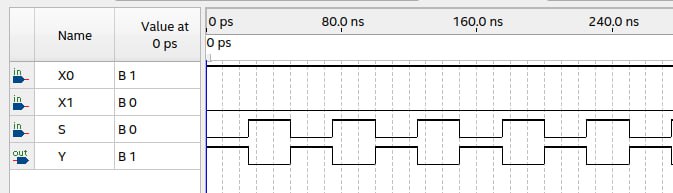
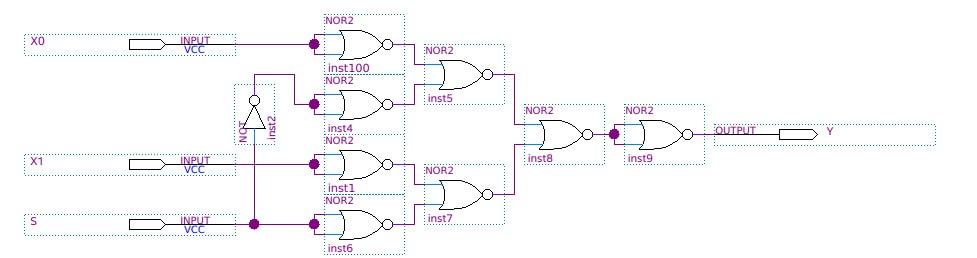
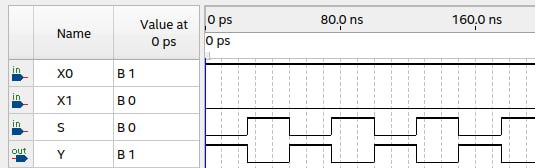


Схема мультиплекстора на 2ИЛИ-НЕ





## Демультиплексор (О, ТИ, АВ)+

Что это, зачем? Обобщённая схема, схема в базисах 2И-НЕ, 2ИЛИ-НЕ

Демультиплексор – цифровое устройство, позволяющее передавать информацию с единственного информационного входа на один из выходов.

Таким образом, демультиплексор (DMX) имеет один вход для подачи информации, адресные входы для коммутации каналов, выходы. Количество адресных входов и выходов демультиплексора связны соотношением:

, где n – число выходов DMX, k – число адресных входов

Логика работы: при подаче на адресные входы двоичного числа на выходе с десятичным номером, соответствующим входному адресному двоичному числу в десятичной системе, устанавливается логический уровень информационного входа «D».

Обозначение:



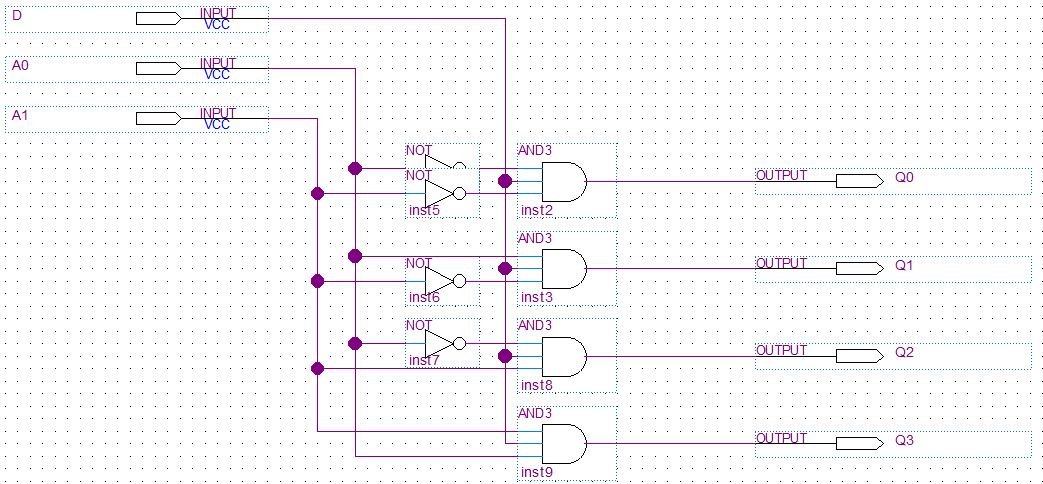
Если вы уже успели прочесть методичку с самого начала, то наверняка испытываете некоторое чувство дежавю на счёт прочитанной информации. Не волнуйтесь, всё хорошо! Причина этому следующая: демультиплексор = дешифратор + информационный вход «D». Какие-либо ещё отличия, кроме смысла применения, отсутствуют. Вот так вот….

Составим таблицу истинности для демультиплексора с 2-мя адресными входами:

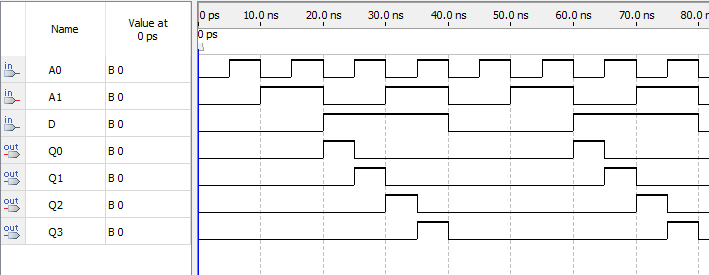
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

Из неё, очевидно:

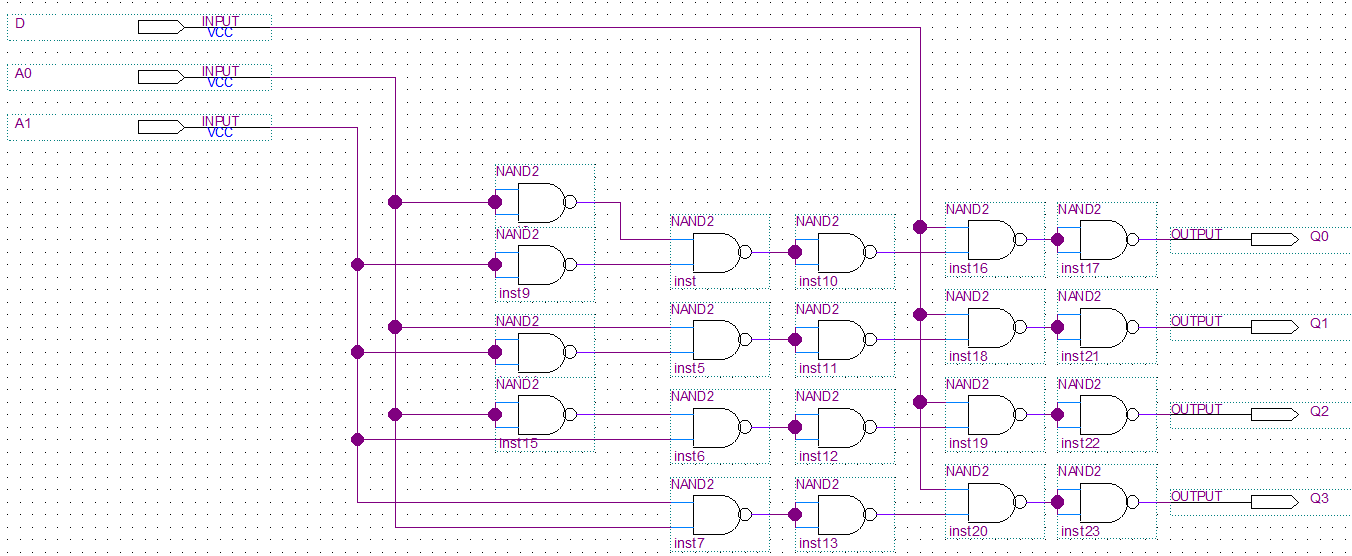
Соответственно схема практически совпадает со схемой дешифратора 2 на 4, а единственным наличием является дополнительный информационный вход «D»:

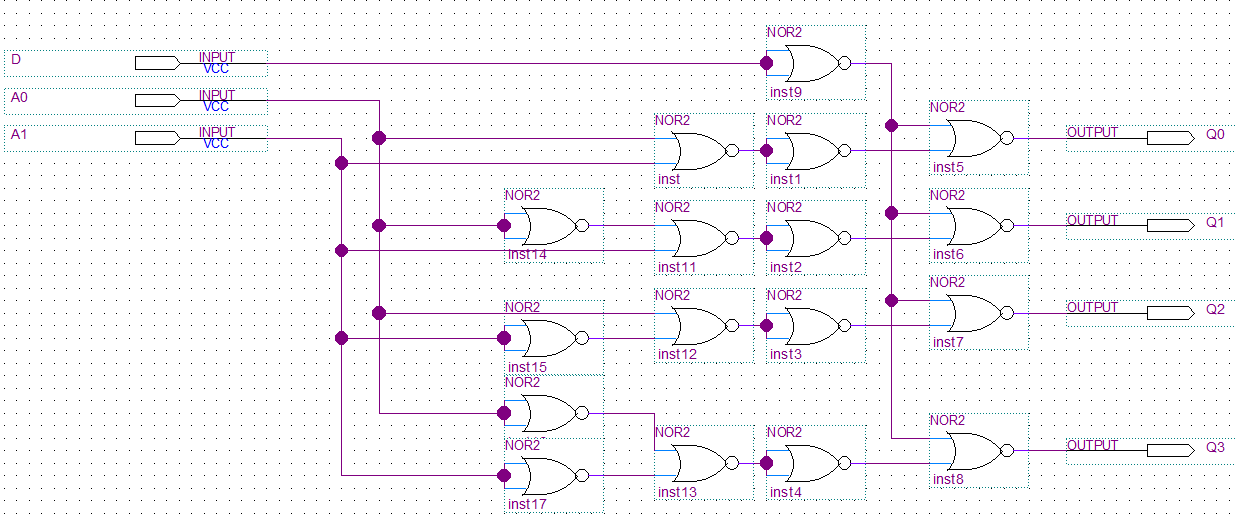


Временная диаграмма:



Представление схемы в базисе 2И-НЕ, 2ИЛИ-НЕ не представляет трудностей. Сделаем это через выражения и затем представим схему:





## Мажоритарный элемент (О?, ТИ, АВ)

Что это, зачем? Обобщённая схема, схема в 2И-НЕ, 2ИЛИ-НЕ.

## Сумматор и полусумматор+

Главное отличие полусумматора от полного сумматора можно увидеть, рассмотрев сложение двух бинарных чисел:

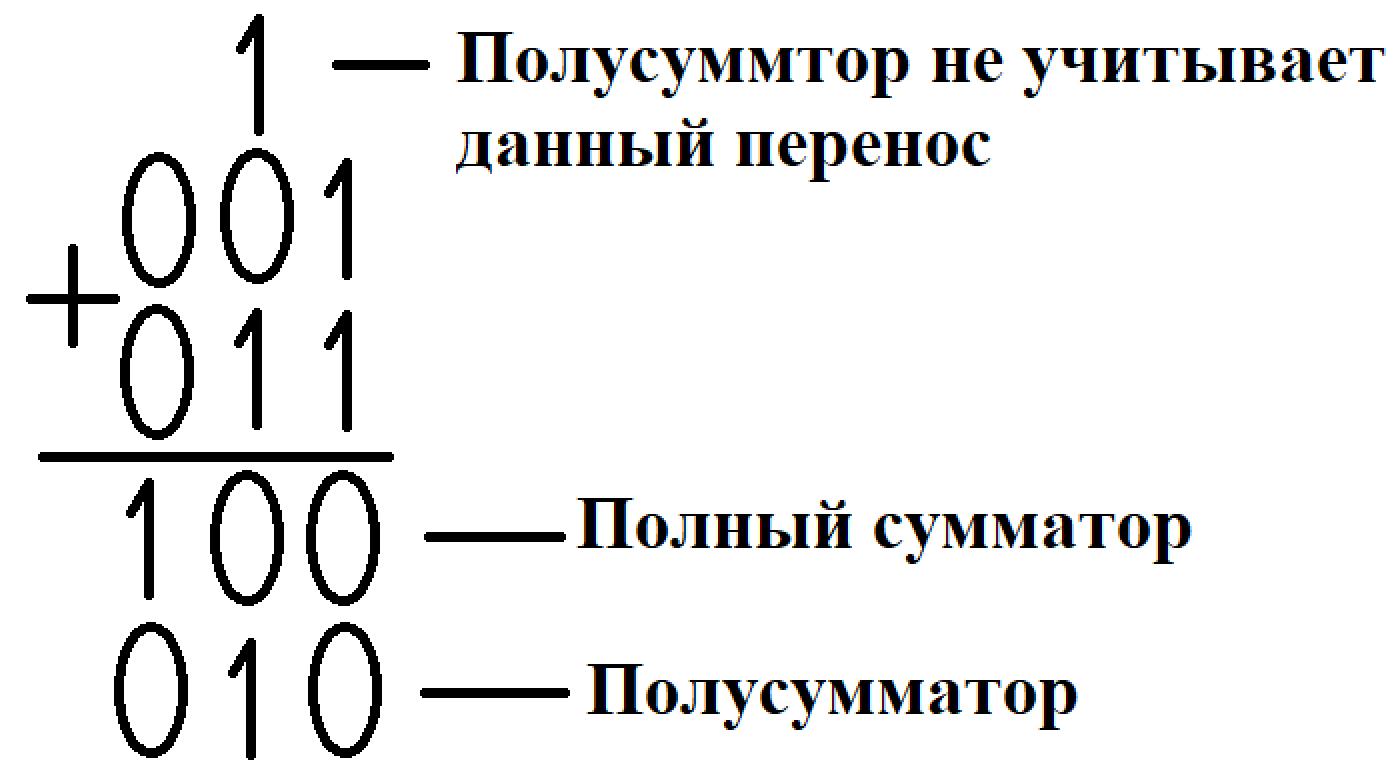
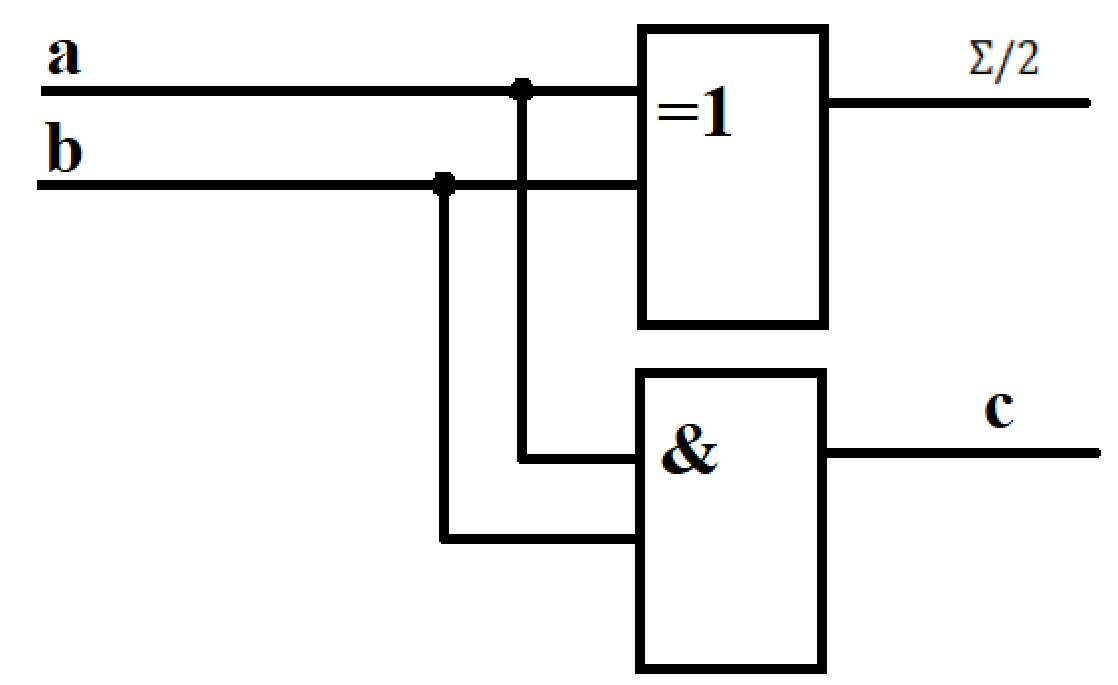


Таблица истинности полусумматора имеет следующий вид:

|  |  |  |  |
| --- | --- | --- | --- |
| a |  |  |  |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Рассмотрев столбец суммы полусuмматора (кек, хахахахахах)….Так, о чём это я…Рассмотрев столбец суммы , заметим, что «1» появляется только в случае, когда «1» присутствует лишь на одном из входов, что соответствует элементу ИСКЛЮЧАЩЕЕ-ИЛИ. Перенос с, в свою очередь, появляется лишь в случае «1» на обеих входах схемы, что соответствует логическому элементу 2И. Таким образом, схема полусумматора, а также выражение в СДНФ имеют следующий вид:

**

Обозначение полусумматора:

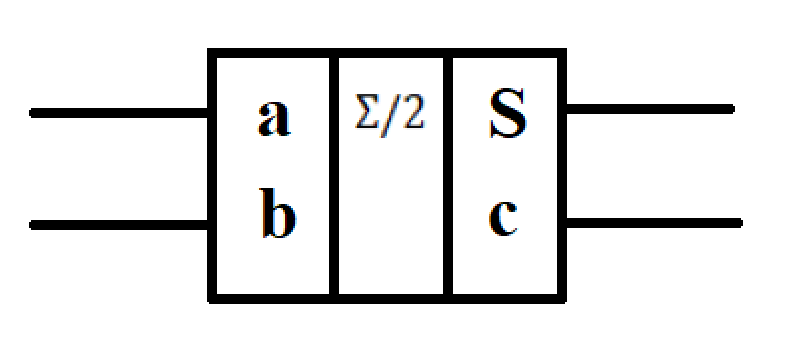
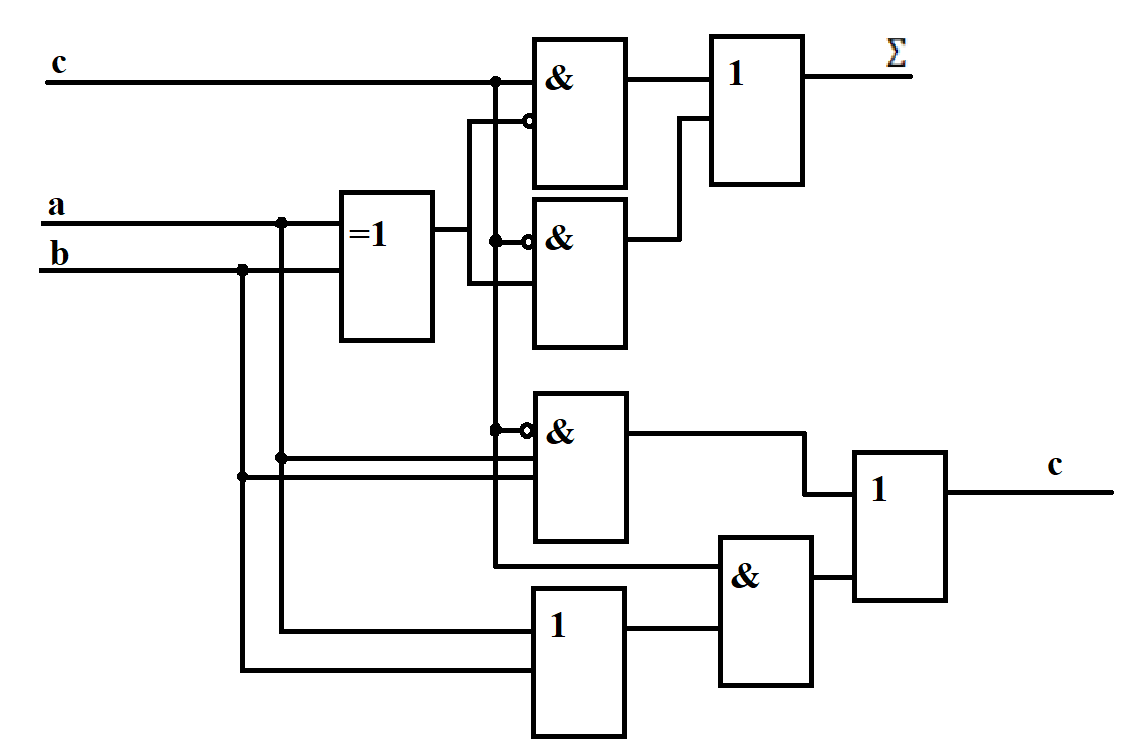


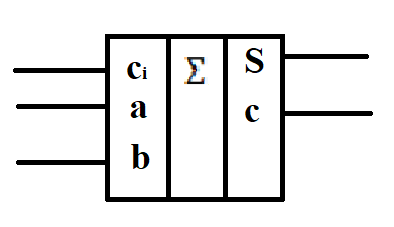
Таблица истинности сумматора имеет следующий вид:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a |  |  |  |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

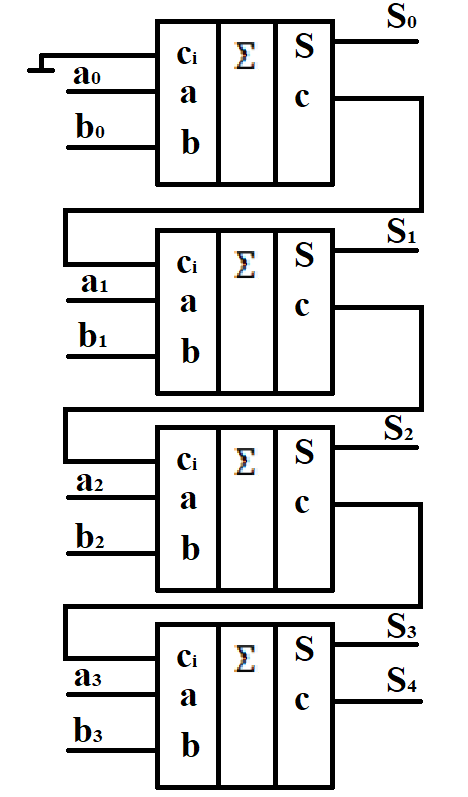
Первые 4 строки полностью повторяют таблицу истинности полусумматора. Заметим, что в строках, где перенос от предыдущего полусумматора (либо полного сумматора) равен «1», образует элемент ИСКЛЮЧАЮЩЕЕ-ИЛИНЕ, а перенос с образует элемент 2ИЛИ. Таким образом, выражения для сумматора и схема имеют следующий вид:



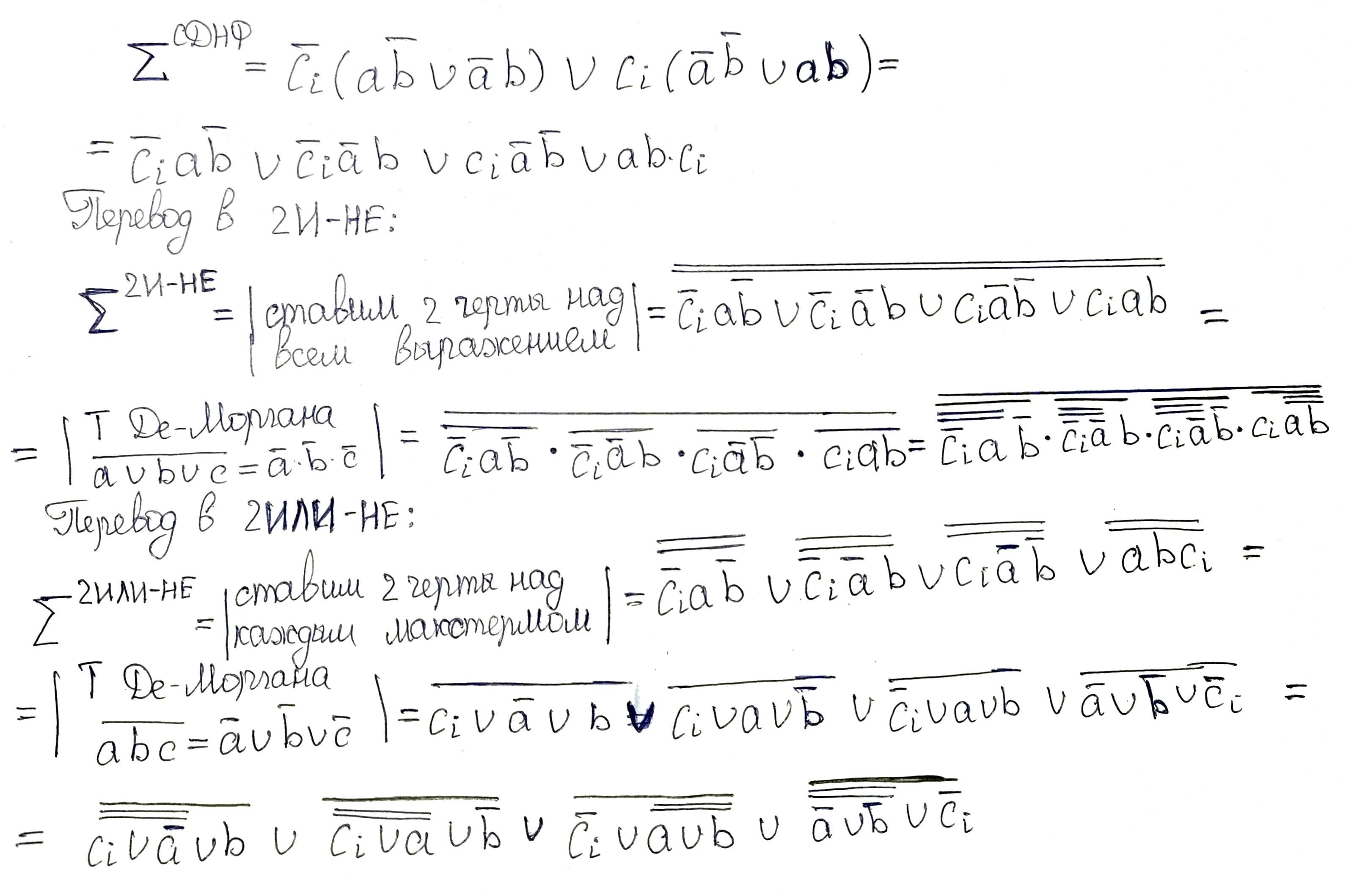
Обозначение сумматора:

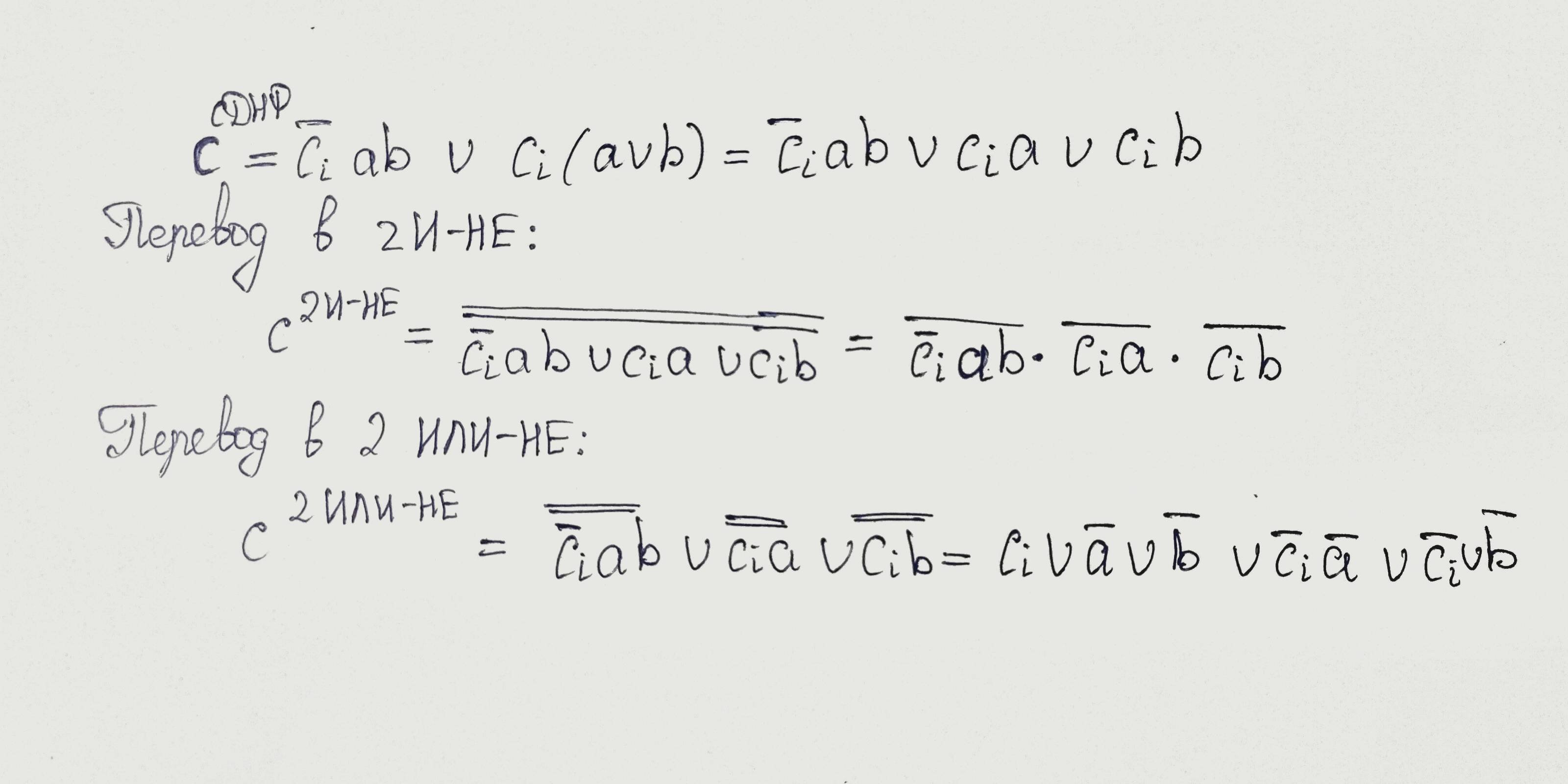


Каскадное включение сумматоров имеет следующий вид. Проверить, как они работают, думаю, можно самостоятельно:



Перевод выражений для сумматора в базисы 2И-НЕ и 2ИЛИ-НЕ:





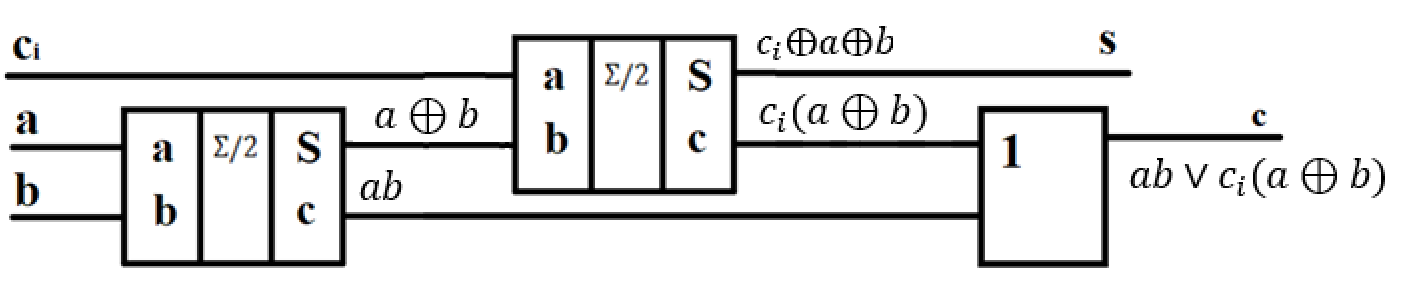
Схемы на 2И-НЕ и 2ИЛИ-НЕ находятся в файлах CNT\_2nand и CNT\_2nor.

Сумматор на полусумматорах. Вернёмся к выражениям для полного сумматора. Произведём для удобства следующую замену:

, тогда

Выражения для полусумматора, как было выведено ранее, имеют следующий вид:

Ну и теперь, основываясь на этих 4-ёх выражениях соберём схему сумматора на полусумматорах:



S полностью совпала, но С нет, как кажется на первый взгляд, но нужно заметить, что:

, ч.т.д.

## Полувычитатель и вычитатель

[Ссылка на источник - Полувычитатель и вычитатель](https://www.youtube.com/watch?v=lqN8xLTtdaA)

[Ссылка на источник - Каскадное включение вычитателей](https://www.youtube.com/watch?v=J7gPUP0aRug)

### Полувычитатель

|  |  |
| --- | --- |
| Пример работы | Таблица истинности |
|  |  |
|  | |

|  |  |
| --- | --- |
| УГО полувычитателя | Обобщенная схема |
|  |  |

xor в базисе 2И-НЕ:

Изображение выглядит как линия, диаграмма, текст, снимок экрана

Автоматически созданное описание

xor в базисе 2ИЛИ-НЕ:

Изображение выглядит как диаграмма, линия, текст, График

Автоматически созданное описание

|  |
| --- |
| Схема 2И-НЕ |
|  |
| Схема 2ИЛИ-НЕ |
|  |

### Вычитатель

Учитывает заем с предыдущей операции ()

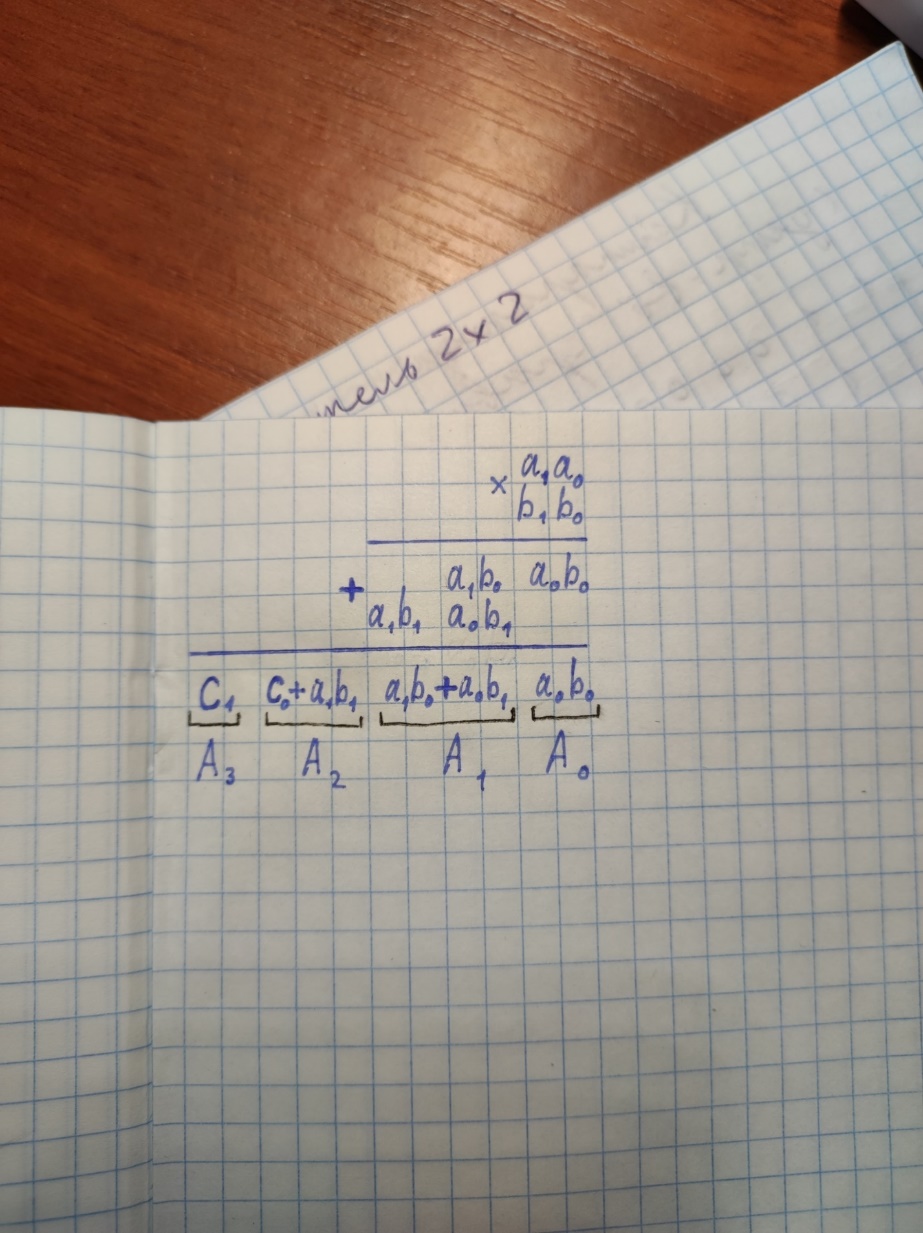
|  |  |
| --- | --- |
| Пример работы | Таблица истинности |
| Для последней строки в таблице истинности |  |
|  | |

|  |  |
| --- | --- |
| УГО вычитателя | Обобщенная схема |
|  |  |
| Вычитатель на полувычитателях | |
|  | |

|  |  |
| --- | --- |
| Схема 2И-НЕ | Схема 2ИЛИ-НЕ |
| В зависимости от базиса вставить соответствующую схему полувычитателя | |

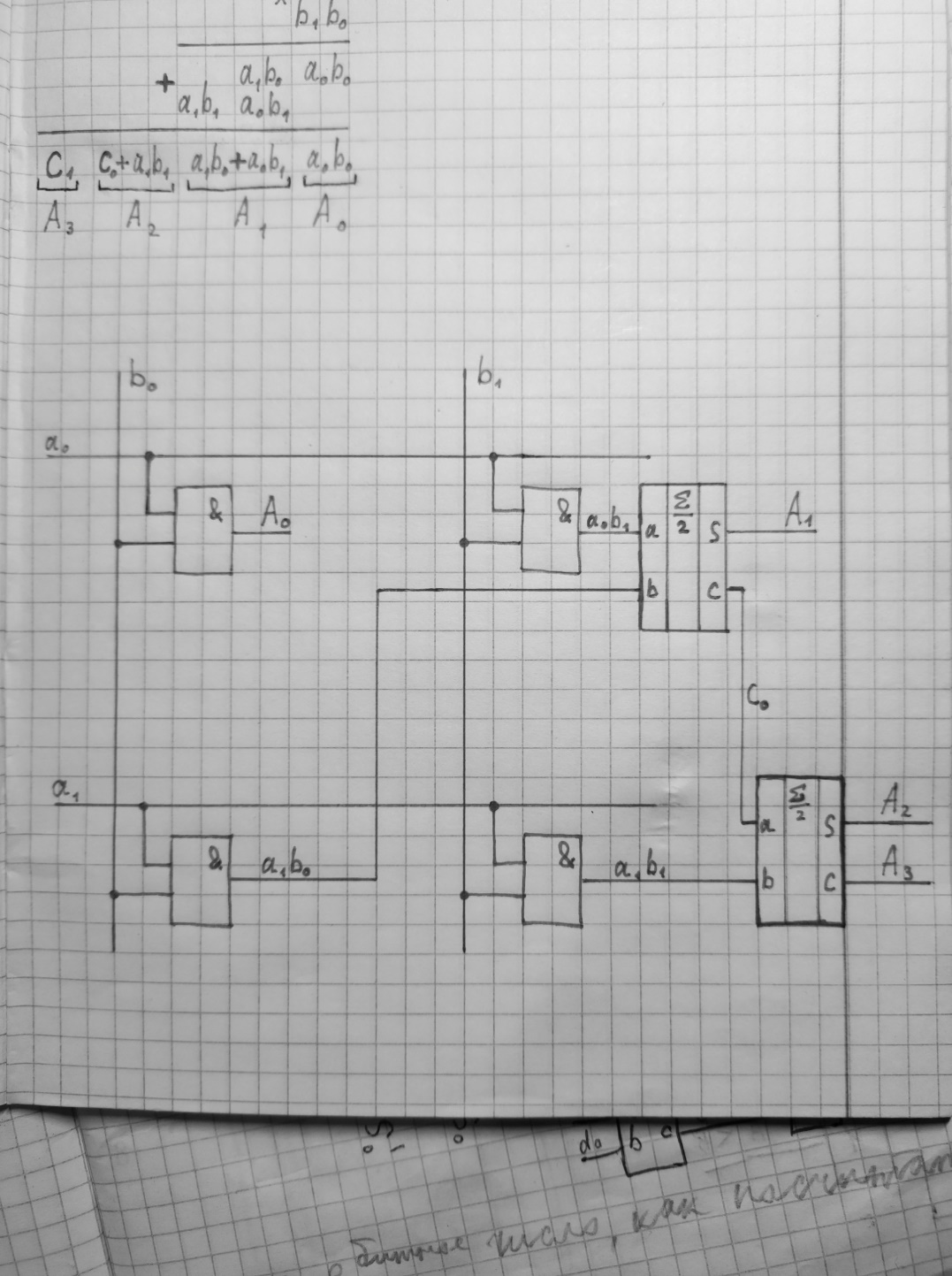
## Умножители. Умножитель 2х2, 3х3, 4х4+

Рассмотрим умножитель 2x2.

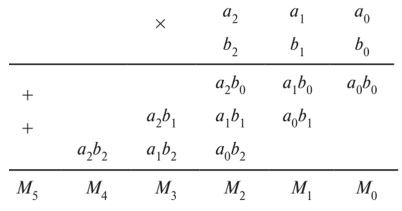


Переносы *c1* и *c0* появляются благодаря суммированию при получении элементов *А2* и *А1* соответственно.

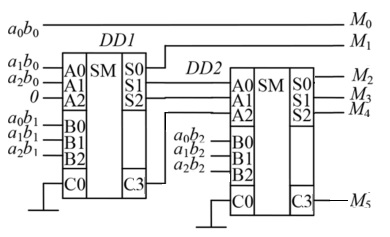
Составим схему соответствующего умножителя 2х2:



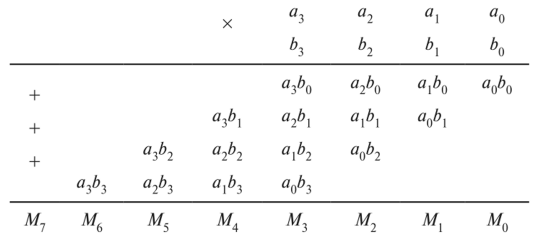
Рассмотрим алгоритм умножения двух трехразрядных чисел:



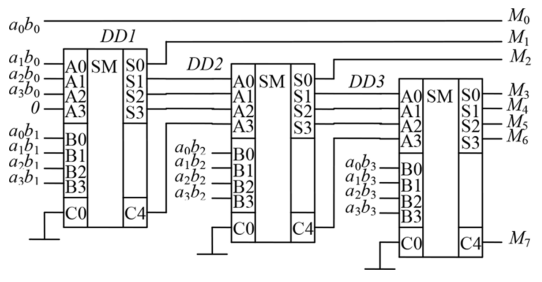
Построим схему умножителя 3х3 на трехразрядных сумматорах:



Рассмотрим алгоритм умножения двух четырехразрядных чисел:



Построим схему умножителя 4х4 на четырехразрядных сумматорах:



## Компаратор (О, ТИ, АВ)

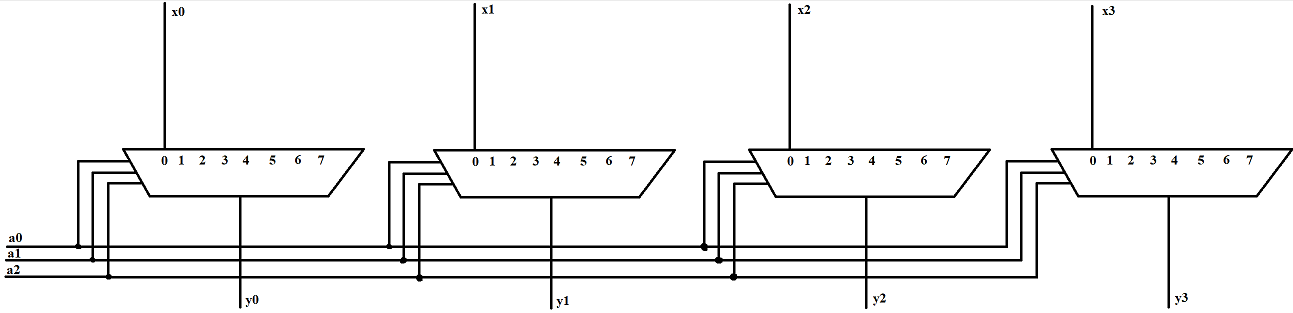
Зачем нужен, принцип работы? Последовательный компаратор. А есть параллельный (на лекциях ничего не написали, нужно почитать)?

## Быстрый сдвиговый регистр (Barrel shifter)+

Устройство быстрого сдвига (Barrel shifter) представляет собой комбинационную логическую схему с n входами данных, n выходами данных и несколькими управляющими входами, сигналы на которых задают сдвиг между данными на входе и данными на выходе.

Называется быстрым, так как построен на мультиплексорах, то есть с использованием только комбинационной логики (триггеры, а значит и последовательностная логика, отсутствуют).

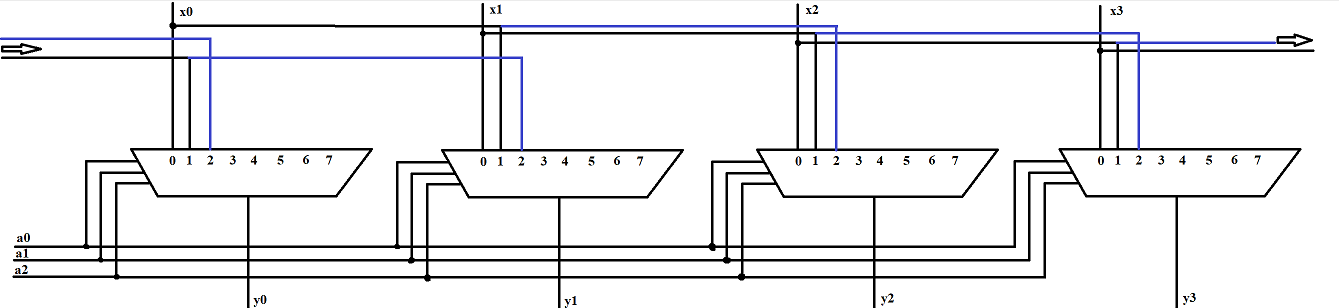
Допустим, будем сдвигать 4-ёх битовое число. Для этого нам понадобятся 4 мультиплексора с 3 адресными входами. На «изначальную» схему на 0 входы мультеплексоров поразрядно поступает наше двоичное число (). На выходах образуется двоичное число после сдвига. Предположим, что (адресные входы). Тогда на выход регистра наше число пройдет без изменений.



Если же установить адресную комбинацию , то число должно быть сдвинуто вправо на «1» разряд. Для этого необходимо соединить со входом «1» второго мультиплексора, – со входом «1» третьего мультиплексора, – со входом «1» четвёртого мультиплексора, – со входом «1» первого мультиплексора:



Если установить адресную комбинацию , то число должно быть сдвинуто вправо на «2» разряда. Для этого необходимо соединить со входом «2» третьего мультиплексора, – со входом «2» чётвертого мультиплексора, – со входом «2» первого мультиплексора, – со входом «2» второго мультиплексор.



По такой логике соединяем все входы, пока не сдадим экзамен Семер…пока входы мультиплексоров не закончатся. Полная схема представлена в Barrel\_shifter.bdf.

## АЛУ

Написали про них мало, но возможно стоит поискать более подробную общую информацию, красиво все описать, а не как в лекции одной картинкой, в которую еле всё поместилось и ниче не понятно.

## Статический сбой (static hazard)+

Пример на какой-то схеме, н-р, как в лекции, но если будут другая, то будет только +

В цифровой электронике статический сбой (static hazard) относится к временному сбою или ошибке, которая может возникнуть в выходном сигнале цифровой схемы из-за несоответствия времени или дисбаланса входных сигналов схемы. Эти помехи могут возникать, когда несколько входных сигналов меняются одновременно или когда в схеме присутствуют задержки, которые не сбалансированы должным образом.

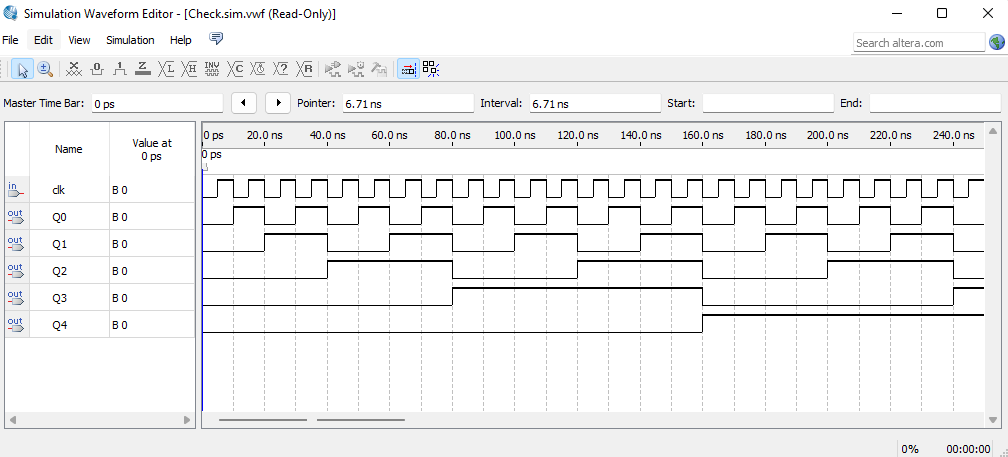
Статические помехи могут привести к неправильным или непредсказуемым значениям выходного сигнала, что может вызвать проблемы в цифровых системах. Они особенно проблематичны в синхронных схемах, где время является критическим фактором. Возникновение помехи, вызванной статической ошибкой, может нарушить намеченное логическое поведение схемы, что приведет к неправильной обработке данных или неправильной функциональности.

Статистический сбой «1» - сбой, при котором должна быть «1», но появляется «0».

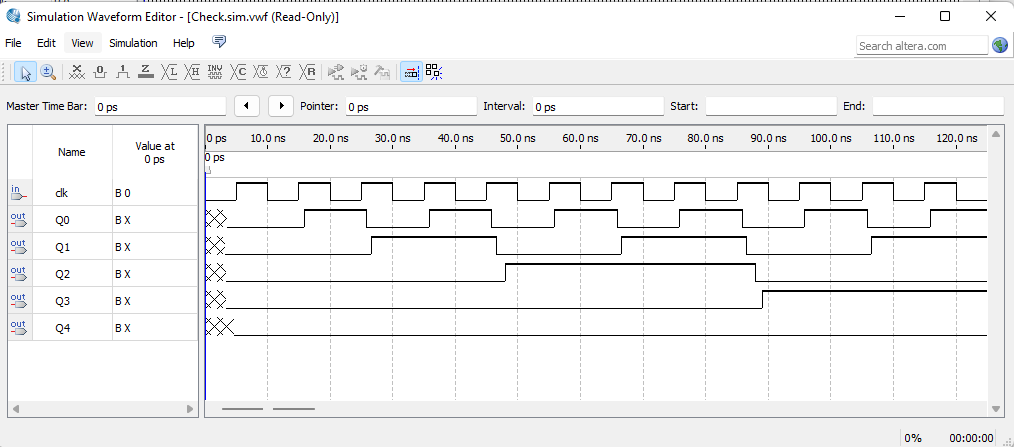
Статистический сбой «0» - сбой, при котором должен быть «0», но появляется «1».

Рассмотрим статистический сбой на примере асинхронного счётчика на T-триггерах по переднему фронту (T\_counter.bdf).

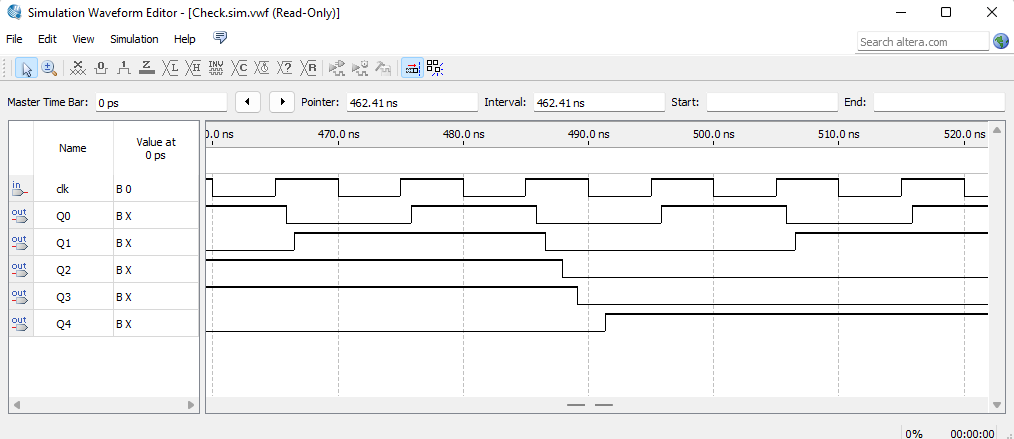
Временная диаграмма идеальной (не имеющей временных задержек) схемы выглядит следующим образом:



Временная диаграмма схемы с временными задержками выглядит следующим образом:



Как видим из диаграммы, в отличие от идеального случая у нас возникает некая временная задержка установления сигнала на выходе при прохождении T-триггера. В связи с тем, что счётчик асинхронный (сигнал на выходе следующего триггера тактируется от сигнала на выходе предыдущего), то временная задержка «накапливается» и может выливаться в следующие казусы:

**

После 15 («01111») на выходах должно установиться число 16 («10000»), но в результате временных задержек на выходах происходит некая суета, и лишь после жесткого тусича сигналов наконец устанавливается 16.

## Вопросы на подумать

Просьба не сразу выдавать решение, а сначала написать некий алгоритм рассуждений, на основе которого можно самому попытаться построить цифровую схему.

При решении вариантов можно не делать задания, которые уже присутствовали в вопросах до этого, если только конечно вы не знаете какой-то изощрённый и отличающийся от представленного в методе ранее способа решения.

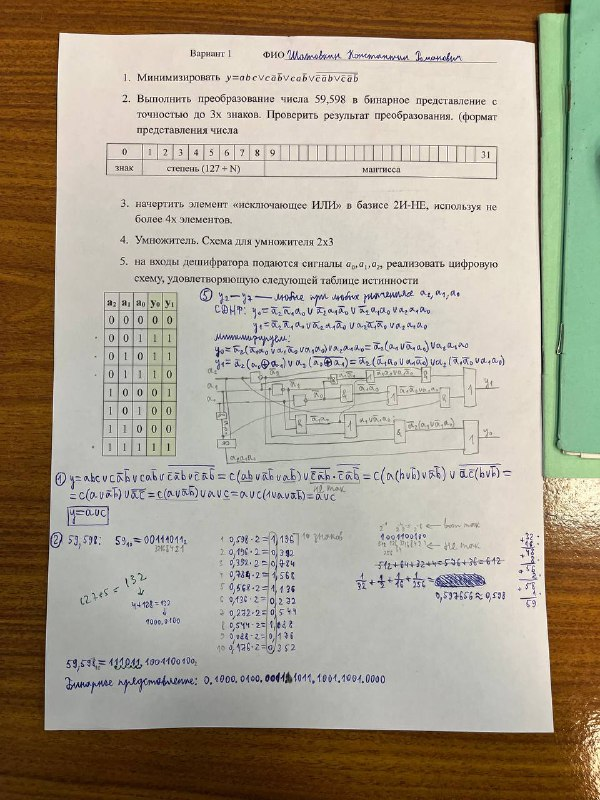
### ИСКЛ ИЛИ на 4 элементах (базис 2И-НЕ). ИСКЛ ИЛИ в базисе 2ИЛИ-НЕ

### Вычитатель на сумматоре

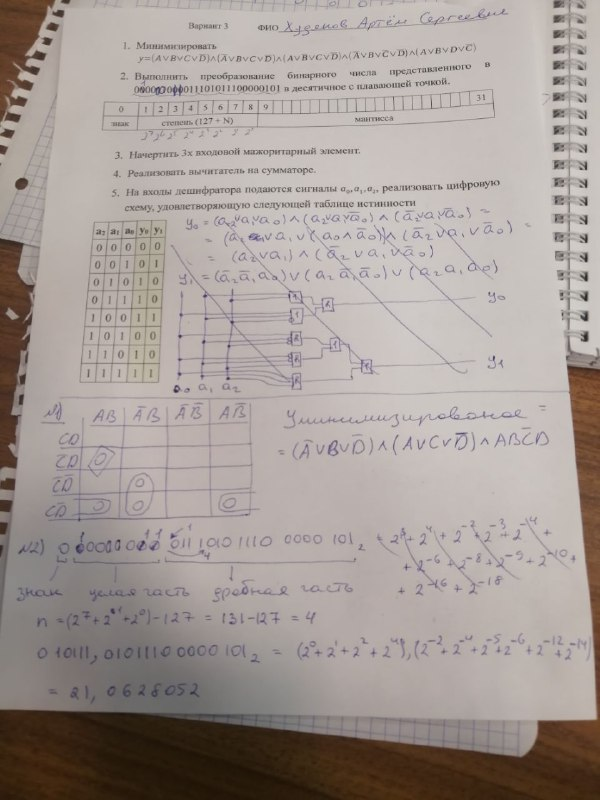
### Сложите 4-5 одноразрядных чисел. Есть восьмиразрядное число, определить количество единиц в нём

### Компаратор на сумматорах.

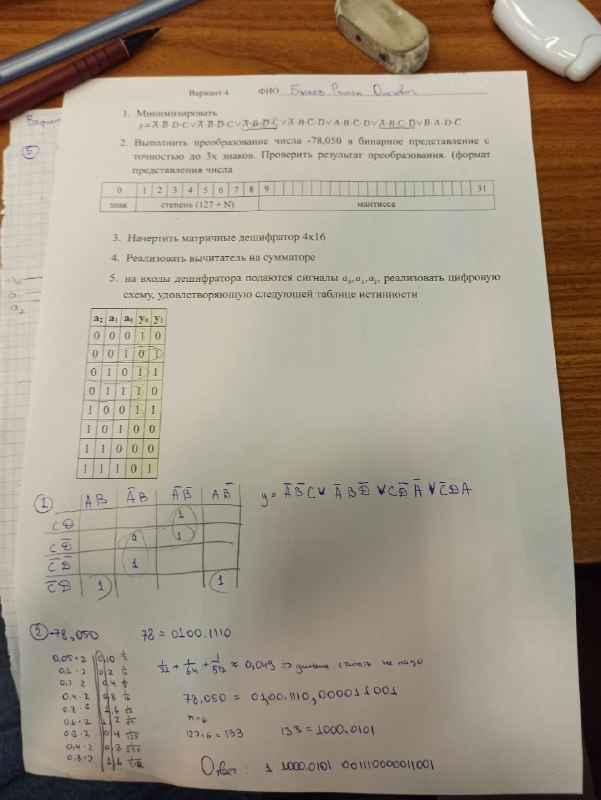
### Вариант 1 РК1



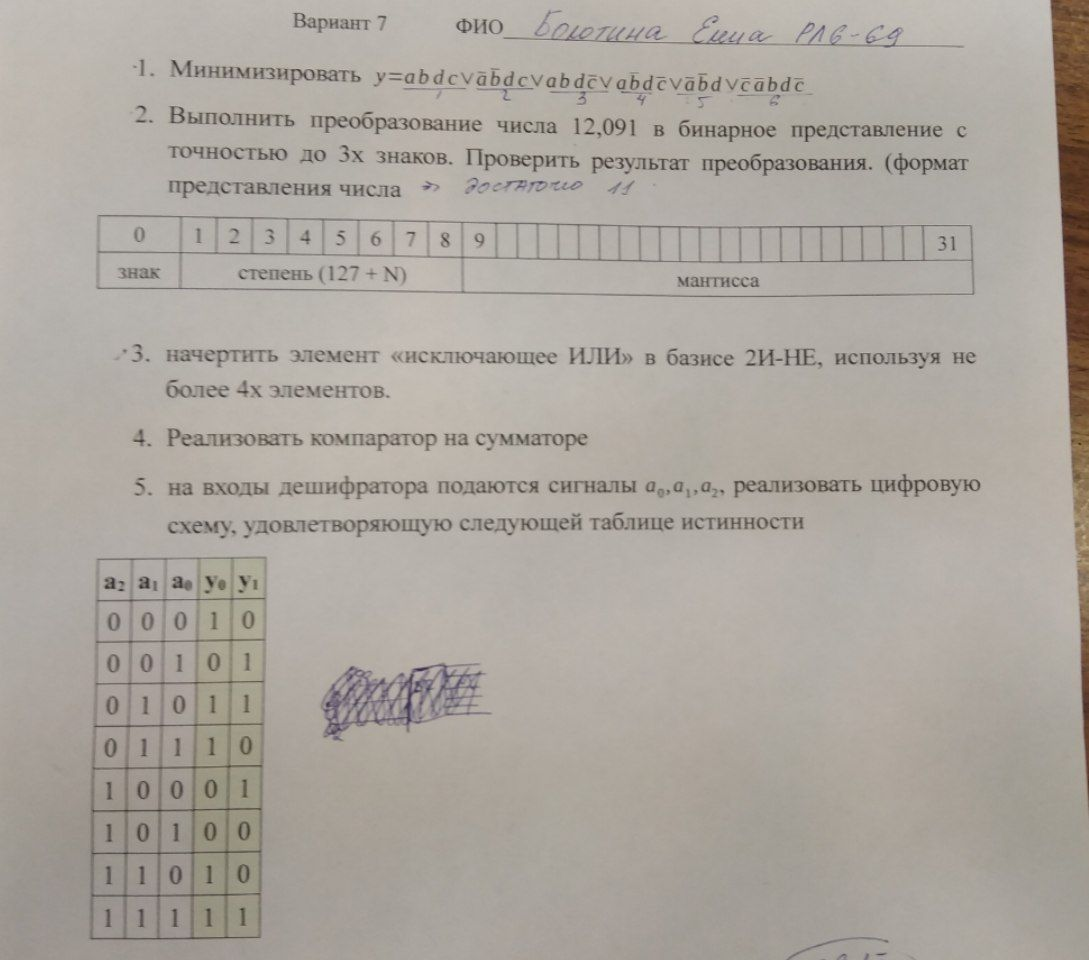
### Вариант 3 РК1



### Вариант 4 РК1



### Вариант 7 РК1



# Последовательстная логика (логика с памятью)

В чём отличие от комбинационной логики? Шо це таке?

## RS-триггер, статический, асинхронный, одноступенчатый

Пояснить статический, асинхр. О, ТИ, ВД. Построить на 2И-НЕ и 2ИЛИ-НЕ. В чём отличие? Как избежать запрещённого состояния?

## RS-триггер, синхронный, статический, одноступенчатый

Пояснить статический, синхр. О, ТИ, ВД. Построить на 2И-НЕ и 2ИЛИ-НЕ. В чём отличие? Как избежать запрещённого состояния?

## D-триггер (D flip-flop), синхронный, статический, одноступенчатый

О, ТИ, ВД. Всегда синхронный! Статический? Реализция D-триггера методом «чёрного ящика». На 2И-НЕ и 2ИЛИ-НЕ.

## Динамический D-триггер, синхронный, двухступенчатый

Переключение по заднему фронту, по переднему? О, ВД.

## T-триггер (счётный)

О, ВД.

## Асинхронный счётчик на D-триггерах.

Что значит Асинхронный? О, ВД. Как реализовать сброс счётчика?

## Динамический RS-триггер

ВД, О, схема. По переднему фронту, по заднему?

## Вычитающий счётчик (асинхронный)

Асинхронный? О, ВД

## Реверсивный счётчик

Принцип, ВД.

## Счётчик с предустановленными значениями

Принцип, ВД.

## Кольцевой счётчик (синхронный)

Принцип, ВД

## Кольцевой счётчик Джонсона

Принцип, ВД, как можно использовать для создания дешифратора?

## Кольцевой счётчик (ходит только одна единица)

Принцип, ВД, самовосстановление счётчика

## Синхронный счётчик вариант 1

Зачем нужны? Принцип работы, ВД. Реальные ВД для синхронного и асинхронного счетчиков, чтобы показать преимущества синхронного.

## Синхронный счётчик вариант 2

## Регистры. Последовательно-параллельные

## Регистры. Параллельно-последовательные

## Регистры параллельно-параллельные

## Регистры. Циклический регистр. Выбор направления. Предустановка значений.

## Динамический одноступенчатый D-триггер

Страшная схема

## Динамический D-триггер с асинхронным сбросом, установкой

Страшная схема со сбросом и с установкой

## JK-триггер, синхронный, статический

ВД, О

## Динамический JK-триггер

## Динамический JK-триггер на D-треггерах

## T-триггер на JK-триггере

## Синхронный счётчик на JK-триггере

## Сброс триггера. Асинхронные, синхронные входы.

## Задачи на подумать.

Просьба не сразу выдавать решение, а сначала написать некий алгоритм рассуждений, на основе которого можно самому попытаться построить цифровую схему.

### Счётчик в базисе 2И-НЕ

### Имеется кольцевой счётчик. Как загнать «1» или несколько «1» и двигать их по кругу?