















# 1.1. Элементная база вычислительной системы. Триггеры.

## 1. Базовые элементы

### 1.1. Логический вентиль

**Логический вентиль (функциональный элемент)** - базовый элемент цифровой схемы, преобразующий (в соответствии с какой-то Булевой функцией) множество входных логических (бинарных) сигналов в единственный выходной логический сигнал.  
В устройствах реализуются с помощью транзисторов.

<b>NOT</b>	$\neg x$		
<b>AND</b>	$x \wedge y$		
<b>OR</b>	$x \vee y$		
<b>NAND</b>	$\neg(x \wedge y)$		
<b>NOR</b>	$\neg(x \vee y)$		
<b>XOR</b>	$x \oplus y$		
<b>XNOR</b>	$\neg(x \oplus y)$		

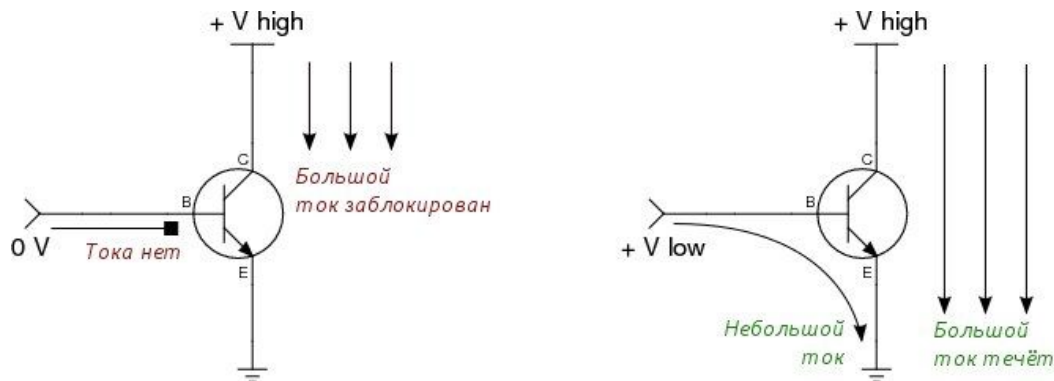
## 1.2. Транзистор

**Транзистор** - основной электронный компонент функциональных элементов, позволяет с помощью небольшого напряжения регулировать большой ток в цепи, что используется для генерации и преобразования электронных сигналов.

Транзисторы имеют три вывода:

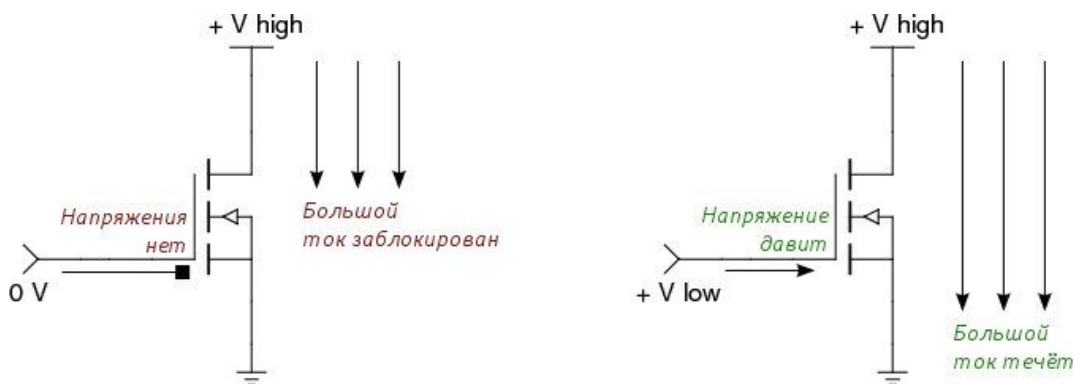
- › коллектор (C) - на него подаётся высокое напряжение, которым хочется управлять
- › база (B) - через нее подается небольшой ток, чтобы разблокировать большой; база заземляется, чтобы заблокировать его
- › эмиттер (E) - через него проходит ток с коллектора и базы, когда транзистор "открыт"

Транзисторы бывают **биполярными** и **полевыми**.



Биполярные, в свою очередь, делятся на **NPN** и **PNP**. NPN пропускает ток от коллектора к эмиттеру при подаче положительного напряжения, PNP - наоборот, при его отсутствии.

**+переключаются быстрее**



Полевые делятся на **N-Channel** и **P-Channel** по тому же самому принципу (сверху, к примеру, N-Channel).

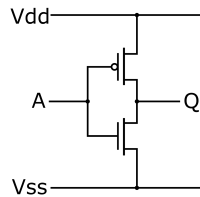
**+потребляют меньше энергии**

**+скорости все равно достаточно** (порядка наносекунд)

**+в основном сейчас используются полевые**

## 1.3. Логические элементы на транзисторах. Представление сигнала

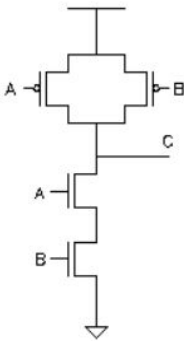
### Инвертор (NOT)



При входном сигнале  $A = 0$ , открывается верхний транзистор, а нижний остается закрытым. Таким образом, напряжение на выходе соответствует единичному сигналу.

Аналогично, при  $A = 1$ , открыт нижний, но не верхний, и напряжение на выходе соответствует нулевому сигналу

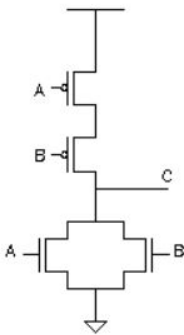
### NAND



Для того, чтобы напряжение соответствовало нулевому сигналу, необходимо, чтобы оба транзистора A и B были открыты, то есть  $A = 1$  и  $B = 1$ .

При этом если хоть один из A и B = 0, хотя бы один из верхних открыт и напряжение соответствует единице

### NOR



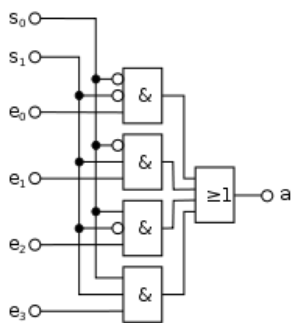
Работает полностью аналогично предыдущей схеме: если хоть один равен единице, то значение равно нулю, иначе - значение равно единице

Все простейшие булевы операции выражаются с помощью **N-** и **P-транзисторов** (которые открыты при значении базы 1\0 соответственно)

## 2. Сложные элементы

### 2.1. Мультиплексор и демультиплексор

#### **MUX** (мультиплексор)



логический элемент с  $2^N + N$  входами и 1 выходом

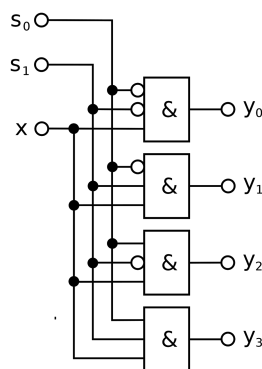
первые  $N$  входов образуют двоичное представление числа  $T$ , а затем на выход подается номер  $T$  из последних  $2^N$  входов

проверить, что текущий элемент имеет номер  $T$  - проверить, что соответствующие биты совпадают со входными

каждый блок **AND** вернет  $e_i$  при  $i = T$ , иначе 0

последний блок **OR** вернет 1 если и только если  $e_T = 1$

#### **DEMUX** (демультиплексор)



логический элемент с  $N + 1$  входом и  $2^N$  выходами

первые  $N$  входов образуют двоичное представление числа  $T$ , затем на все выходы, кроме  $T$ , подаются нули, а на него - то же, что было на последнем входе

аналогично, здесь каждый блок **AND** вернет  $x$ , если все поданные биты числа  $T$  совпали со всеми битами числа  $i$ , иначе 0

тогда  $y_T = x$ , а все остальные выходы вернут 0

#### **DECODER** (дешифратор)

-

демультиплексор часто объединяют с дешифратором в одно, однако простейший дешифратор, в отличие от **DEMUX**, не имеет входа  $x$

работает аналогично **DEMUX**, но возвращает  $y_T = 1$ , а остальные 0

## 2.2. Триггеры

**Триггер** - класс электронных устройств, обладающих способностью длительно находиться в одном из нескольких устойчивых состояний и чередовать их под воздействием внешних сигналов. Состояние триггера распознается по значению выходного сигнала.

Триггеры подразделяются на **синхронные** и **асинхронные**.

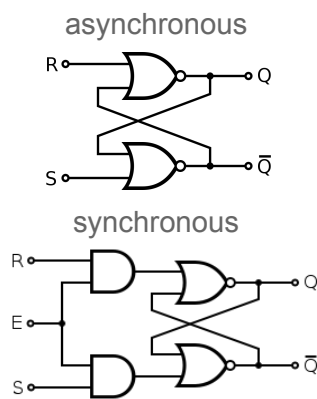
- Асинхронные меняют свое значение в реальном времени (при изменении входного сигнала).
- Синхронные - только после подачи специального синхронизирующего сигнала.

Синхронизация применяется для регулировки времени изменения состояния триггера, когда имеют место более одного контролирующего входа, между изменениями которых может быть задержка.

### RS (reset-set) flip-flop

- 0  $\approx$  no change
- R  $\approx$  reset ( $Q=0$ )
- S  $\approx$  set ( $Q=1$ )
- RS forbidden

При отсутствии сигналов не меняет значение, сигналы сбрасывают его в 0 или устанавливают в 1 соответственно



верхний и нижний выход всегда противоположны

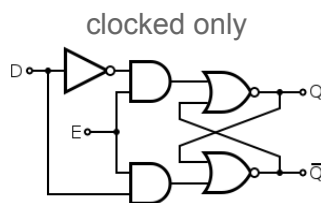
- если не подана ни одна команда, верхний **NOR** вернет Q, нижний - его инверсию
  - при подаче сигнала *reset*, верхний **NOR** возвращает 0
  - при подаче сигнала *set*, нижний **NOR** возвращает 0
- если синхронизация E не подана, то на оба **NOR** поступают нули  
при поступлении сигнала синхронизации, на них поступают текущие значения R и S

Состояние **R=1 & S=1** является запрещенным, так как при таком входе Q, и !Q оба станут нулями. Это нарушает логику работы триггера. Поэтому для **RS-триггера гарантируется**, что **RS** никогда не будут поданы одновременно

### D (data/delay) flip-flop

- 0  $\approx$  no change
- E  $\approx$  set ( $Q=D$ )

При отсутствии сигналов не меняет значение, иначе сбрасывается в 0 или устанавливается в 1



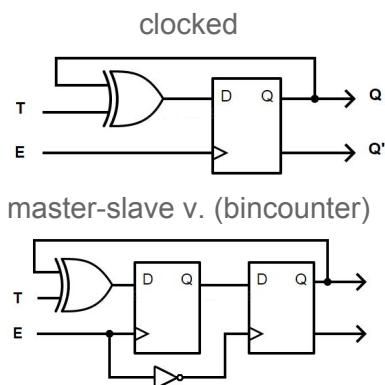
не имеет смысла без синхронизации  
представляет собой надстройку на **RS**, которая при синхронизации

- при D = 0 передает *reset*
- при D = 1 передает *set*

### T (toggle) flip-flop

- 0  $\approx$  no change
- E  $\approx$  toggle ( $Q^A=T$ )

При отсутствии сигналов не меняет значение, сигналы сбрасывают его в 0 или устанавливают в 1 соответственно



редко используется без синхронизации  
при подаче синхронизации переключает выходной сигнал в соответствии со входным

- при T = 0 не меняет значение
- при T = 1 переключает значение

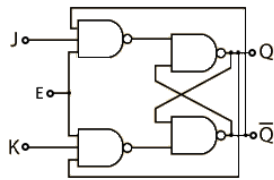
master-slave работает так же, но по принципу "пока синхронизация есть, первый изменяет значение, а второй - нет, как только синхронизация падает на 0, второй сохраняет новое значение первого"  
таким образом, такой триггер меняет свое значение только при изменении значения синхронизации с 1 на 0

## JK (jump-kill) flip-flop

- 0  $\approx$  no change
- J  $\approx$  jump ( $Q=1$ )
- K  $\approx$  kill ( $Q=0$ )
- JK  $\approx$  invert ( $Q \neq 1$ )

При отсутствии сигналов не меняет значение, сигналы по одному сбрасывают его в 0 или устанавливают в 1 соответственно, а вместе - инвертируют

synchronous only



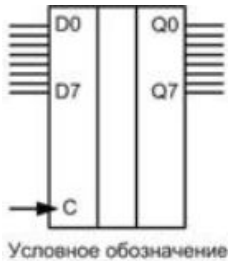
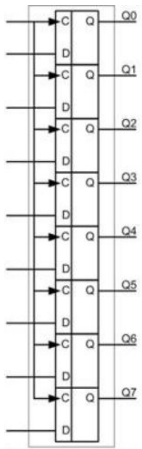
верхний и нижний выход всегда противоположны

- если не подана ни одна команда, оба левых **NAND** дадут 1 и оба правых сохраняют свои значения
- при подаче сигнала *jump*, верхний **NAND** возвращает Q, и если тогда правый верхний даст отрицание 0, то есть 1
- при подаче сигнала *kill*, нижний **NAND** возвращает отрицание Q, и тогда правый нижний даст 1, и  $Q = 0$
- при подаче *invert*
  - если  $Q = 0$ , то в нижний левый **NAND** будут поданы E, K, Q, то есть 1, 1, 0, а в верхний - E, J, !Q, то есть 1, 1, 1. Это ровно то же самое, как и если бы мы просто подали *jump*
  - если  $Q = 1$ , то в нижний левый **NAND** будут поданы 1, 1, 1, а в верхний - 1, 1, 0. Это то же самое, что и подать *kill*

## 2.3. Регистры и счетчики

**Регистр** - устройство, используемое для хранения  $n$ -разрядных двоичных данных и выполнения преобразований над ними.

Представляет собой упорядоченный набор **D-триггеров**, число которых соответствует числу разрядов в слове.

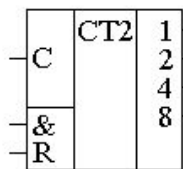
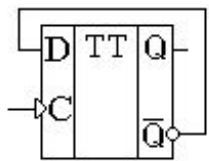


Слева схема простого **8-разрядного регистра**.

Однако одного сигнала синхронизации не хватит, чтобы все 8 разрядов получили напряжение, считаемое за логическую 1.

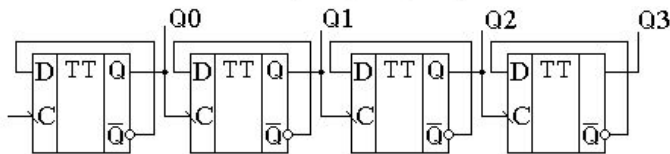
В таком случае подается **инвертированная синхронизация**: на общий синхронизационный выход подается отрицание C, и перед каждым C-входом стоит **NOT**. За счет этого, при  $C = 1$ , на проводе логический 0, и каждый инвертор превращает этот 0 в полноценный единичный сигнал.

**Счетчик** - устройство, предназначенное для счета количества тактов, используется, например, для деления частоты и в схемах таймеров, а также для выбора инструкций из ПЗУ в микропроцессорах.



Простой двоичный счетчик реализуется на основе **T-триггеров**.

Сам по себе **T-триггер** становится **счетчиком до двух**, если реализовать его как два связанных **D-триггера** с обратной связью (см. master-slave t flip-flop). При подаче синхронизации второй будет работать на сохранение и значение поменяется только в момент, когда синхронизация спадет с 1 на 0.

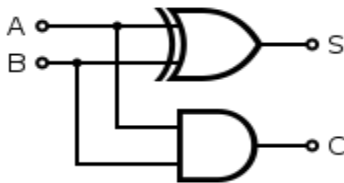


Многоразрядный счетчик получается при подключении нескольких счетчиков подряд.

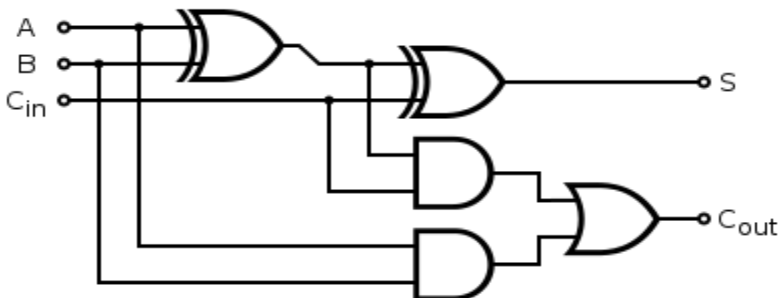
Тогда  $Q_0Q_1Q_2Q_3$  - двоичное представление подсчитанного числа (следующая цифра меняется, когда предыдущая изменилась с 1 на 0, как и должно быть).

## 2.4. Сумматор

**Сумматор** - устройство, осуществляющее суммирование двоичных чисел.



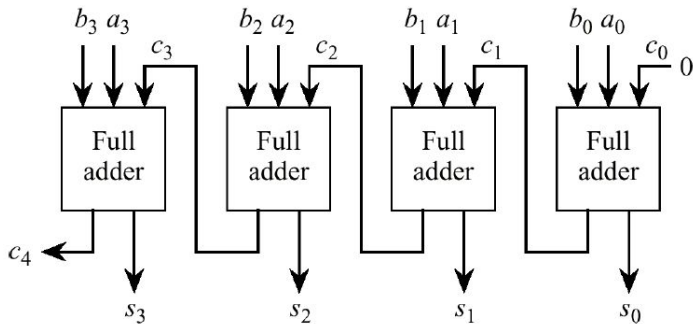
**Неполный сумматор** - логическая схема, имеющая два входа и два выхода (двухразрядный сумматор, бинарный сумматор). Позволяет вычислять сумму  $A + B$ , где A и B - это одноразрядные двоичные числа. При этом результатом будут два бита S и C, где S - это бит суммы по модулю 2, а C - бит переноса.



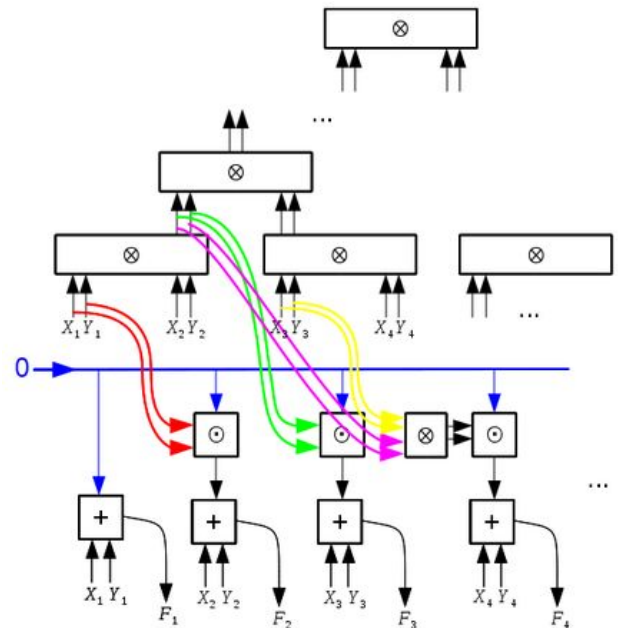
**Полный сумматор** - логическая схема, имеющая три входа A, B и  $C_{in}$  и два выхода S и  $C_{out}$ , где S - бит суммы по модулю 2, а  $C_{out}$  - бит переноса.

Основное отличие в том, что любой выход возможен, тогда как  $S = 1$  и  $C = 1$  в неполном не бывает.

**Полный сумматор** используется для построения **каскадного сумматора** и **двоичного каскадного сумматора**, которые вычисляют сумму многозначных чисел (их отличие во времени работы -  $O(n)$  против  $O(\log(n))$ ), где  $n$  - количество разрядов в складываемых числах.



Сверху - краткая схема **каскадного сумматора** на основе **полных сумматоров**, в которых в качестве входов на  $i$ -й сумматор подаются  $i$ -е биты чисел и перенос из предыдущего бита, после чего выход, соответствующий их сумме по модулю 2, идет куда нужно, а перенос - к следующему сумматору.



Справа - схема **двоичного каскадного сумматора**. В ее основе лежит идея о том, что текущий перенос зависит от всех предыдущих и от текущих битов складываемых чисел. Поэтому, чтобы посчитать все биты переноса, мы строим на них дерево отрезков и считаем биты переносов на префиксах параллельно за  $O(\log(n))$ . После чего все биты результата сложения могут быть также посчитаны параллельно.



### 3. FAQ

-Построить полный сумматор\AND на полевых транзисторах

-Построить мультиплексор\демультиплексор

-Построить RS-триггер

-Почему JK-триггер не может быть асинхронным?

**Ответ 1:** потому что асинхронный JK-триггер не позволяет совершить переход из состояния (0, 0) в (1, 1), так как какой-то из входов обязательно поменяется. Тогда вместо необходимого инвертирования значения, оно сначала будет заменено 0 или 1, в зависимости от того, какой вход раньше, а затем инвертировано, что некорректно.

К примеру, последовательности изменений

<b>Q = 1</b> (0, 0) - (0, 1) <b>Q = 0</b> (0, 1) - (1, 1) <b>Q = 1</b>	и <b>Q = 1</b> (0, 0) - (1, 1) <b>Q = 0</b>
--	--

приводят к различным результатам.

**Ответ 2:** потому что при отсутствии синхронизации при подаче сигнала *invert*, значение на выходе будет постоянно меняться. Кажется, на полевых транзисторах в этом случае происходит утечка тока.

-Построить счетчик на 3 выхода

-Построить “умножитель”, получающий на вход два трехбитовых числа и возвращающий 5 бит их произведения