Конспект 02 — Логические схемы

Содержание

1. Выкладки из булевой алгебры
2. Графическое отображение формул
    1. Общие представления
    2. Основные принципы и правила
3. Некоторые полезные схемы
    1. Сумматор
    2. Мультиплексор
    3. RS-триггер
4. Синхронизация (#TBA)

”Они заявляют, что якобы умеют. На самом деле ни черта они не умеют.”
— П. С. Скаков

Выкладки из булевой алгебры

Читателю уже известно, что все современные электронные устройства производят вычисления в двоичной системе счисления, а о битах двоичных чисел, в свою очередь, бывает удобно думать как о булевых значениях. Вернёмся в третий класс на урок дискретной математики и вспомним, какие основные операции мы можем с ними производить:

Отрицание	Конъюнкция	XOR	Дизъюнкция
$\neg$ / $\overline{}$	$\wedge$	$\oplus$	$\vee$

В отличие от этого урока, впрочем, в контексте разговоре об архитектуре имеет смысл ввести полноценный приоритет операций, который соответствует порядку их перечисления выше (слева направо приоритет уменьшается).
В языке Си все из них, за исключением XOR, представлены в двух вариантах: побитовом и логическом. Стоит помнить, что побитовые операции применимы только к целочисленным типам.

Мы помним, что достаточно комплексные выражения с булевыми переменными можно записывать в виде булевых формул. При работе с вычислительной техникой такой

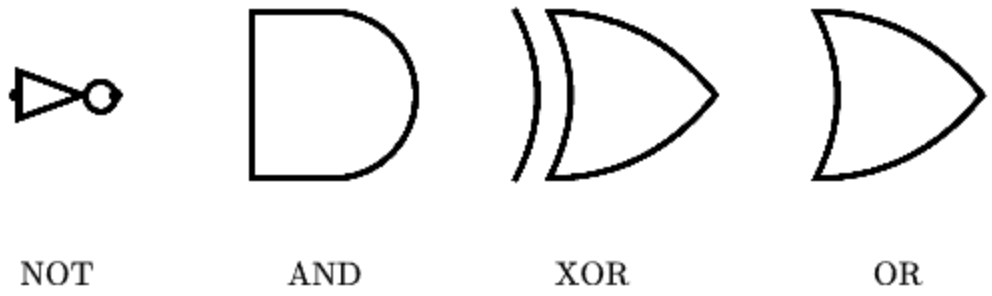
прямой подход, впрочем, обладает рядом недостатков (главным из которых является, пожалуй, недостаточная наглядность), в связи с чем мы вводим другой, более специфичный для предмета обсуждения, формат записи логических выражений.

Графическое отображение формул

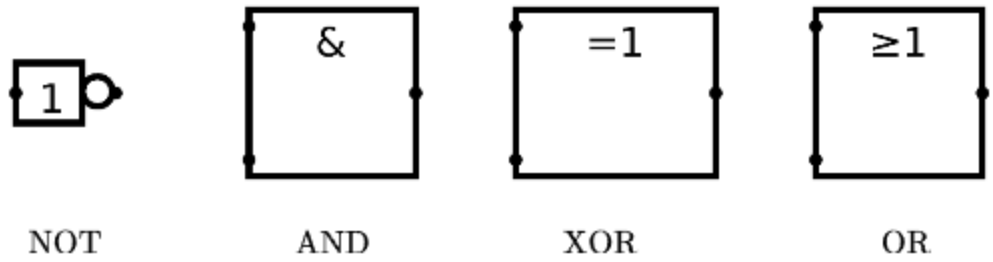
Общие представления

Формулы можно рисовать!
Мы делаем это, соединяя *логические элементы* (они же *gates*), каждый из которых соответствует некоторой булевой функции, причем «некоторой булевой функцией» может являться не только одна из стандартных операций, но и вообще любой составной модуль, который мы сами сконструируем, что очень сильно упрощает чтение схем. Логический элемент обычно представляет собой некоторую замкнутую фигуру с рядом входов (слева), выходов (справа) и, возможно, каким-то дополнительным обозначением.

Касаемо внешнего вида самих элементов можно выделить два наиболее широко используемых стандарта. Первый разработан Американским национальным институтом стандартов (ANSI) и предоставляет следующие формы для базовых модулей:



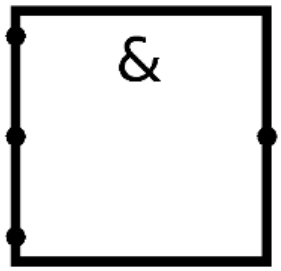
Другой разработан Международной электротехнической комиссией (IEC) и, помимо прочего, практически идентичен ГОСТу:



К стандарту ANSI мы чаще всего не будем к нему прибегать из соображений как эстетических, так и исторических, поскольку в странах Европы и СНГ распространён именно второй вариант, и, следует заметить, не совсем безосновательно. Во-первых, он

предоставляет стилистически единообразный способ обозначения как стандартных, так и составных модулей: прямоугольник с идентификатором. Во-вторых, он позволяет без каких либо усилий расширять число входов модуля, что, помимо прочего, ещё и соотносится с тем, как описываемые схемы реализуются аппаратно:

Тернарный AND



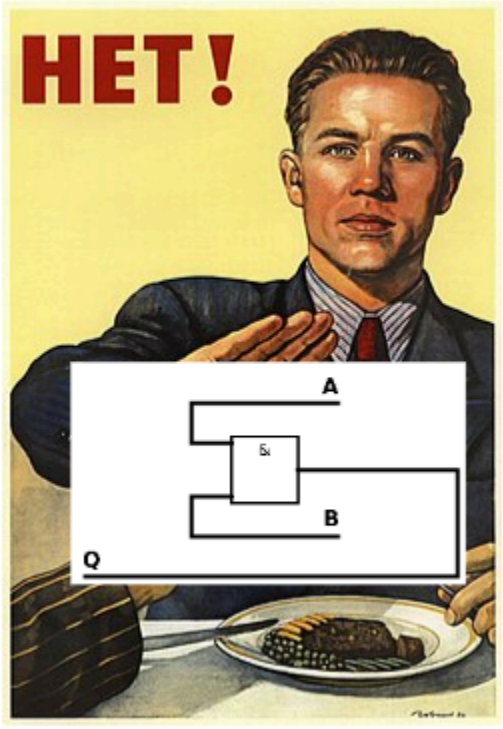
Следует также отметить, что ГОСТ и IEC всё же не являются одним и тем же, хотя и очень похожи. Основное отличие состоит в принятом стандартном обозначении модулей «НЕ» и «ИЛИ»: отечественный формат не использует обозначения для отрицания (в любом из стандартов оно обычно обозначается просто кругом на входе или выходе элемента в зависимости от того, что мы хотим инвертировать), а также обозначает дизъюнкцию 1 вместо ≥ 1. Можно предположить, что вся эта странная эпопея с единицами нужна только для того, чтобы при повороте схем не возникало неоднозначностей, связанных с горизонтальной симметрией традиционно использующихся в записи формул галочек.

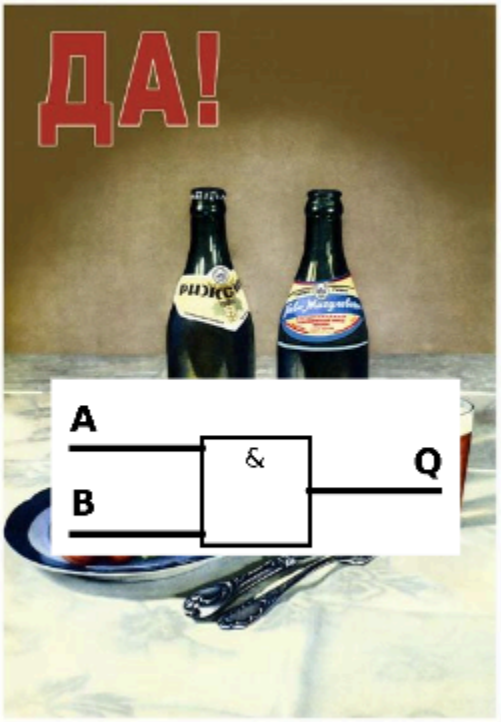
Стоит отметить, что один и тот же результат, то есть одну и ту же формулу, можно представить очень разными способами, и при этом совершенно нельзя однозначно сказать, что какой-то из них правильный, а какой-то — нет. В этом контексте мог бы зайти разговор об оптимизации логических схем, но это отдельная огромная область, в которую мы не будем залезать.

Основные принципы и правила

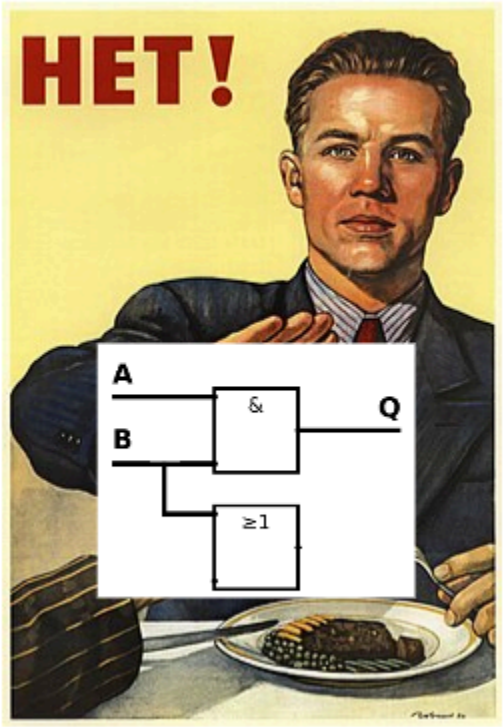
Принципы построения логических схем кажутся достаточно интуитивно понятными и во многом таковыми и являются. Тем не менее, во избежание недопониманий и откровенных глупостей стоит регламентировать ряд моментов, которые, в общем-то, очевидны, но всё же могут вызывать неоднозначности.

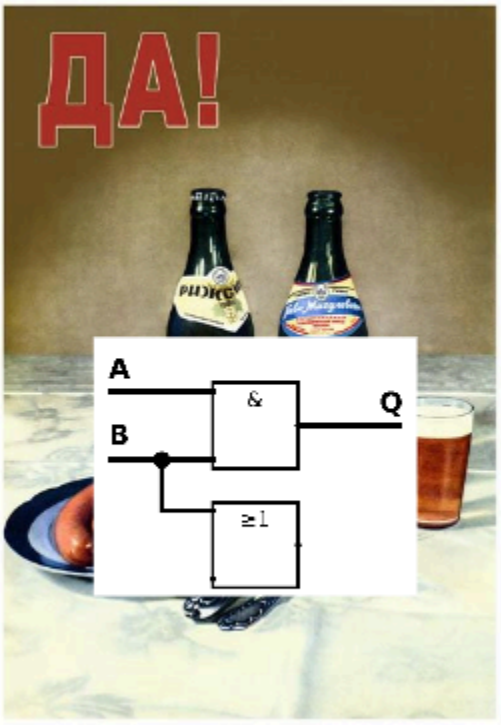
Во-первых, входы *всегда* располагаются в левой части схемы, а выходы — в правой:



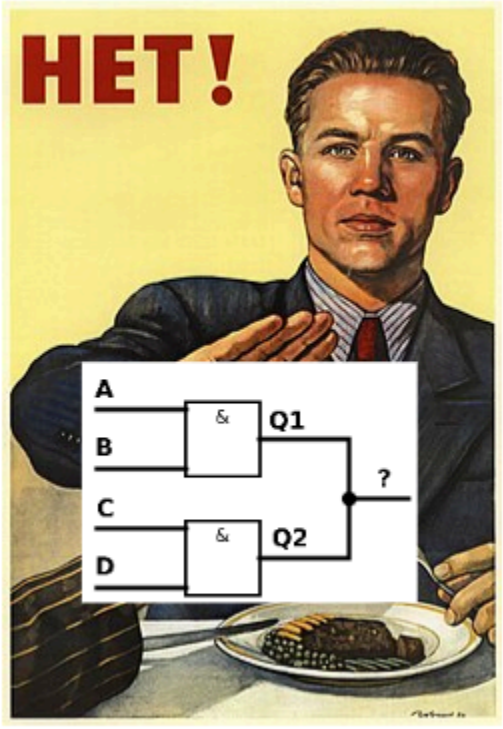


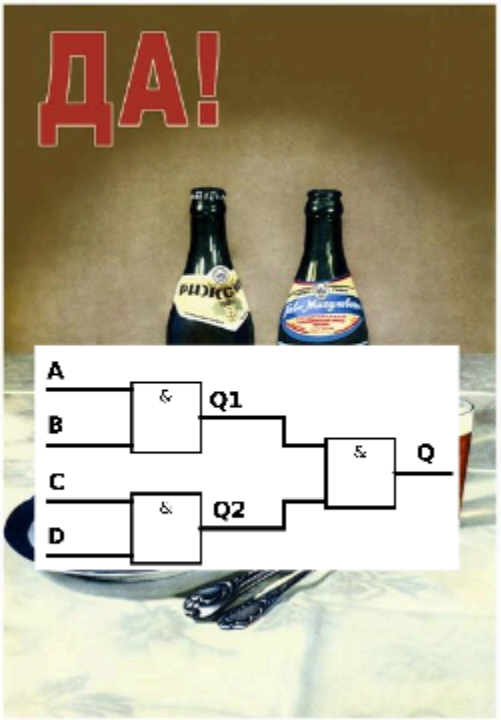
Во-вторых, на месте соединения проводов всегда ставится точка, которая позволяет отличить его от пересечения:





В-третьих, мы можем легально разветвлять единственное соединение, но ни в коем случае не можем без соответствующего логического элемента соединять их:





В общем, не делайте всякую ерунду и слушайте маму.

Некоторые полезные схемы

Сумматор

Попробуем построить схему, которая осуществляет сложение двух чисел. Начнём с более простой задачи: построить *полусумматор*, который принимает два бита и возвращает их сумму и перенос. Чтобы определить, как вычислять выходные значения, составим таблицу истинности этого элемента:

A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Отсюда видно, что $S = A \oplus B$ и $R = A \wedge B$. В виде логической схемы это можно изобразить следующим образом:

Полусумматор HSUM

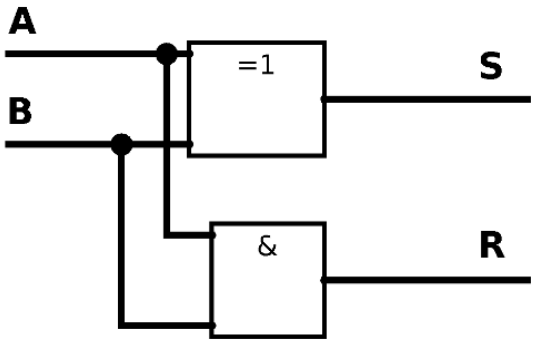


Схема *сумматора* принимает не два, а три значения: два слагаемых и перенос. Её таблица истинности имеет следующий вид:

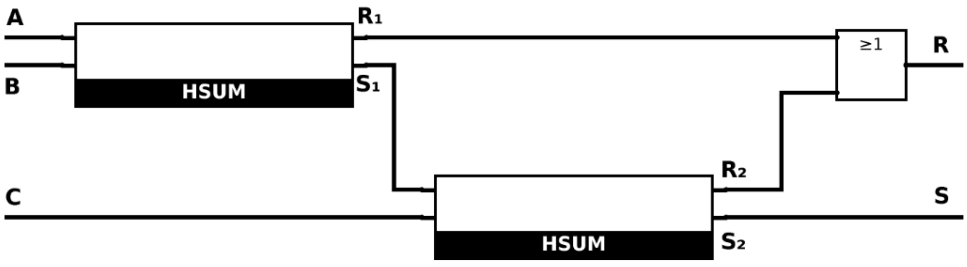
A	B	C	S	R
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Было бы здорово, если бы её можно было реализовать при помощи двух последовательных полусумматоров, и её действительно можно реализовать при помощи двух последовательных полусумматоров, если заметить, что они не могут одновременно вернуть единичный остаток. Из этого замечания следует, что

$$S = \mathrm{HSUM}(C,\ \mathrm{HSUM}(A, B)_S)_S \qquad R = \mathrm{HSUM}(A, B)_R \vee \mathrm{HSUM}(C,\ \mathrm{HSUM}(B, C)_S)_R$$

или, в виде схемы,

Сумматор SUM

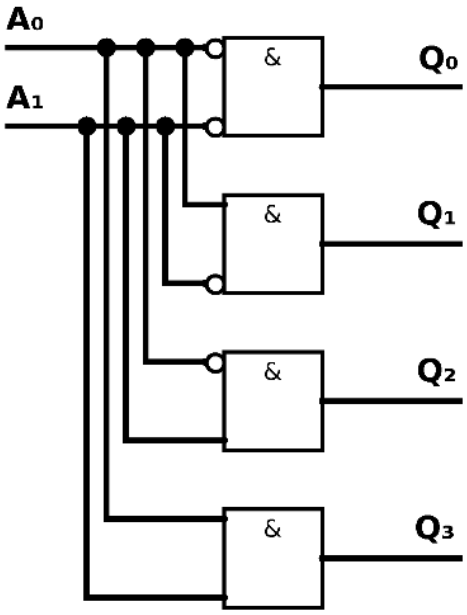


Мультиплексор

Реализуем схему, имеющую четыре входа данных и два входа адреса. Адрес воспринимается как двоичное число, кодирующее номер входа данных, который следует подать на единственный выход.

Решение этой задачи удобно начать с вынесения логики перевода двоичного числа в десятичное в отдельный модуль:

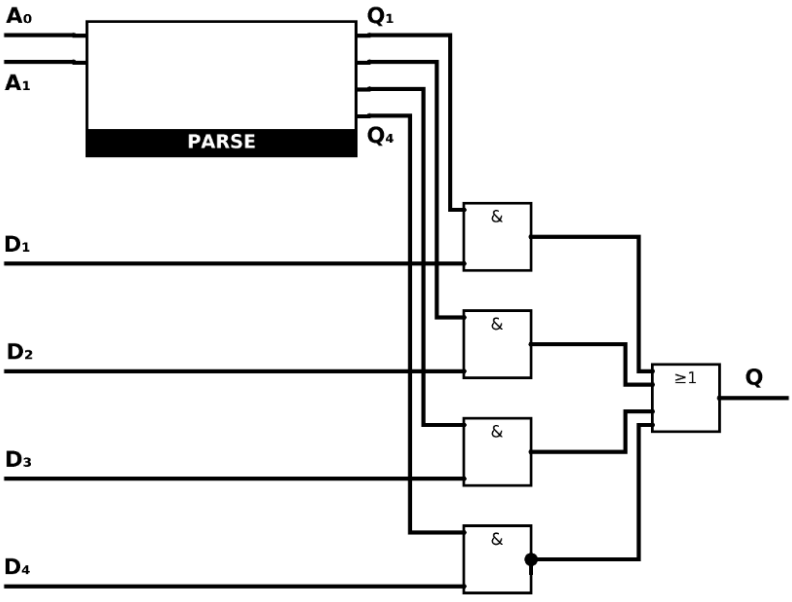
Схема PARSE перевода двоичного числа в десятичное



Эта схема принимает два входа адреса и возвращает 1 на тот выход, который соответствует заданному на входе числу. На все остальные выходы возвращается 0.

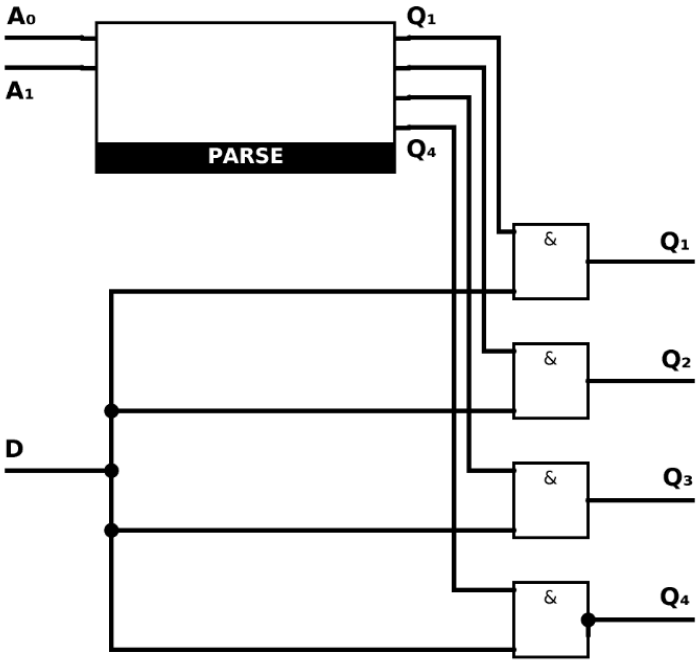
Теперь построим с её помощью мультиплексор:

Схема мультиплексора MUX



Практически идентичным образом можно построить *демультиплексор* — схему, которая возвращает единственный вход данных на один из четырёх выходов в зависимости от переданного адреса:

Схема демультиплексора DEMUX



Мультиплексор и демультиплексор преимущественно служат комбинационными «строительными блоками», то есть вспомогательными модулями для построения более сложных логических схем.

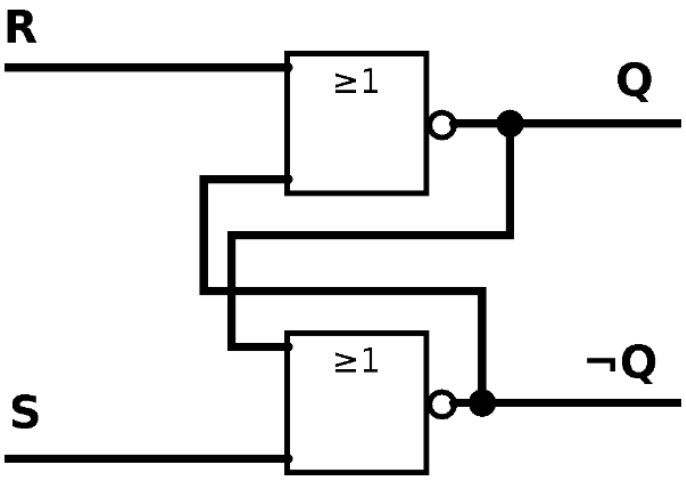
RS-триггер

Пусть мы хотим сконструировать устройство, поведение которого определяется следующей таблицией истинности:

S	R	Q
0	0	Предыдущее значение
0	1	0
1	0	1
1	1	Неважно

Без лишних предисловий приведём его канонический вид:

Схема RS-триггера



Эта схема называется **RS-триггером** (*от Reset / Set*) и является крайне важной, поскольку представляет собой самую примитивную ячейку памяти. Собственно, *триггер* — это устройство, способное длительное время сохранять своё состояние и изменять его под действием внешних сигналов, то есть практически запоминать битовую информацию.

Хотя фактически эта схема имеет два выхода, на самом деле она возвращает единственный бит информации, поскольку второй выход всегда дублирует первый с единственной поправкой на инверсию. Более подробно разобраться в устройстве RS-триггера остаётся читателю в качестве упражнения, поскольку это крайне важно и полезно осознать самостоятельно.

Источники

1. Скаков П. С. — Лекции по аппаратному обеспечению вычислительных систем, 2 семестр, 2026
2. Викиконспекты — Триггеры (https://neerc.ifmo.ru/wiki/index.php?title=Триггеры)
3. Э. Таненбаум — Архитектура компьютера (https://jasulib.org.kg/wp-content/uploads/2023/03/1-Tanenbaum_E_-_Arkhitektura_kompyutera_4-e_izdanie.pdf)
4. Дэвид М. Харрис, Сара Л. Харрис — Цифровая схемотехника и архитектура компьютера (https://is.ifmo.ru/books/2016/digital-design-and-computer-architecture-russian-translation_July16_2016.pdf)
5. Д. Паттерсон, Дж. Хеннеси — Архитектура компьютера и проектирование компьютерных систем (http://178.140.10.58:8083/read/831/pdf)