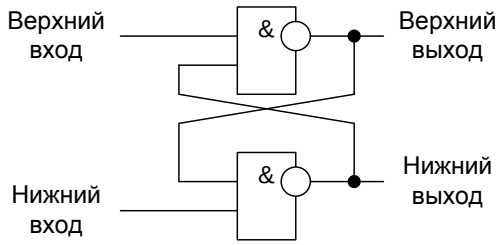
Триггеры

RS триггер

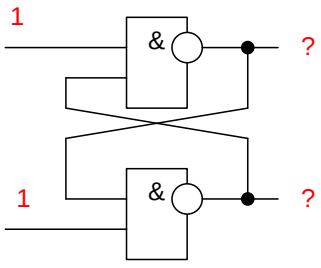
До сих пор мы рассматривали обработку сигналов без их хранения. Проводя аналогии с языком Си, мы писали функции, которые обрабатывают аргументы, приходящие извне, формируя какой-то результат, не вводя локальных переменных. Пришла пора познакомится с ними (то есть, с переменными). Начнём с весьма замысловатой конструкции из двух логических элементов.



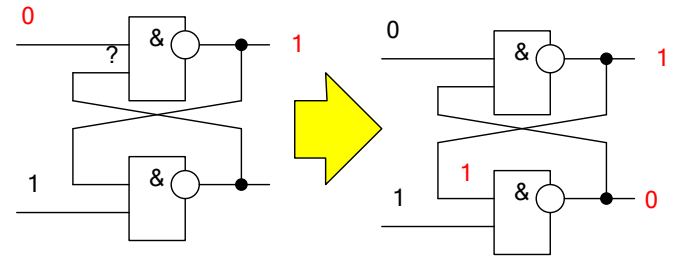
Забавная кракозябла, неправда ли? Эту кракозяблу желательно выучить наизусть, она пригодится. А пока – давайте посмотрим, как она работает. Давайте подадим на оба входа по единице (забегая вперёд, скажу, что обычно кракозяблы имеют на входе обе единицы 99% рабочего времени, поэтому я с такого состояния и начал). Напомню таблицу истинности для элемента 2И-НЕ:

X1	X2	Y
0	0	1
0	1	1
1	0	1
1	1	0

Зная эту таблицу, получаем:



Состояние верхнего выхода зависит от состояния верхнего входа (там у нас 1) и нижнего выхода (но мы не знаем, что там). Поэтому что будет на верхнем выходе сразу после включения питания схемы – не знает никто. Аналогично не знает никто и состояния нижнего выхода. И зачем тогда вообще смотреть на такие непонятные штуки? А вот зачем. Подадим-ка мы на верхний вход ноль…

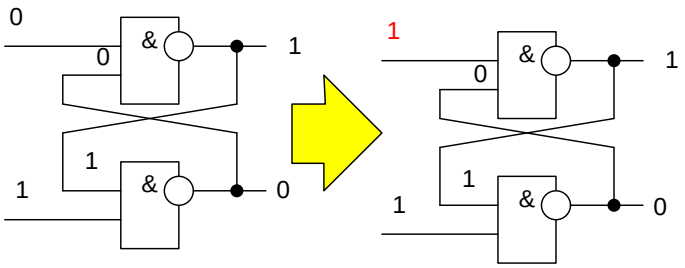


Сразу после подачи нуля, верхний выход станет 1

А переход верхнего выхода в 1, автоматически перебросит нижний выход в 0

Зная таблицу истинности элемента 2И-НЕ, мы можем точно сказать, что если хоть на одном из его входов есть 0, то выход точно перейдёт в единицу, независимо от того, что на втором входе, так что плевать нам на то, что мы не знаем его состояния… А раз верхний выход перешёл в гарантированную единицу, то с этого момента мы выяснили, что у нижнего выхода оба входа равны единице, из таблицы истинности следует, что выход стал равен нулю. Ну ладно, хоть какая-то определённость вырисовывается.

Теперь возвращаем верхний выход в единицу…

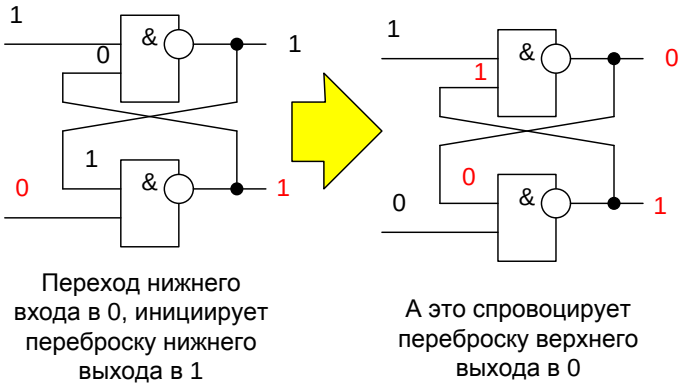


Было

Вернули верхний вход в единицу

Согласно таблице истинности 2И-НЕ, состояние верхнего выхода не изменилось. А значит, не изменилось и состояние нижнего выхода. Схема осталась в том же состоянии. И сколько бы мы не дёргали верхний вход то в 0, то в 1 – она останется стоять, как вкопанная.

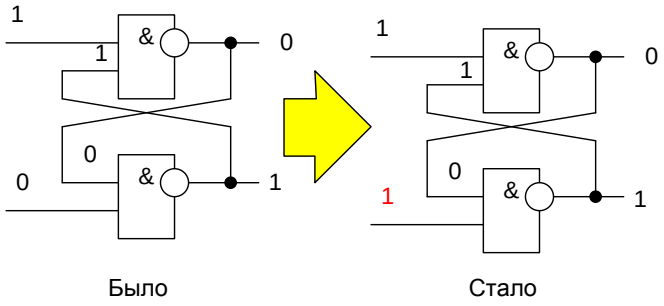
Что ж. Попробуем теперь перевести в ноль нижний вход.



Переход нижнего входа в 0, инициирует переброску нижнего выхода в 1

А это спровоцирует переброску верхнего выхода в 0

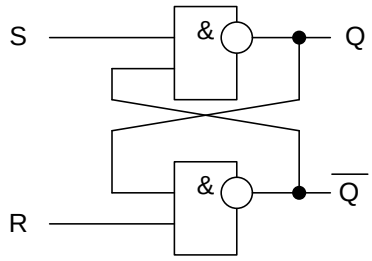
Забавно. Но вернём нижний вход в единицу



Состояние выходов не изменилось.

Итак. У нас есть система, которая хранит своё состояние, если на обоих входах будут единицы, поднимает верхний вход в 1, если подать ноль на верхний вход, сбрасывает верхний вход в 0, если подать ноль на нижний вход, а нижний выход всегда инверсен верхнему. Оба входа в 0 мы ронять не будем. Это запрещённое состояние. Мы никогда не должны так делать, по крайней мере, если хотим получить какой-либо хороший результат.

Значит верхний вход производит установку (SET), назовём его S. Нижний – производит сброс (RESET), назовем его R. Верхний выход – это прямой выход, назовём его Q. А нижний – это инверсный, назовём его Q с надчёркиванием (со времён теоремы Де Моргана, мы помним, что радиотехники любят отмечать инверсию надчёркиванием над названием).

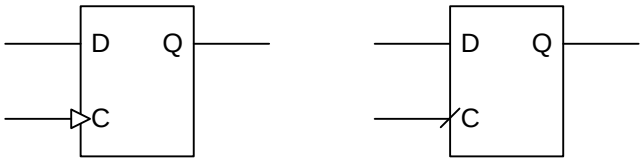


Вот так, легко и непринуждённо, мы познакомились с простейшим триггером. Триггер – это такой элемент, который может находиться в устойчивом состоянии при некоторых дёрганиях входов. Он аналогичен битовой переменной с точки зрения компьютера. Правда, любой программист воскликнет: «Постойте! Это же нонсенс! Отдельное воздействие для установки в единицу и отдельное – для установки в ноль! Даёшь заполнение переменной значением, какое бы оно ни было». И это верно. RS триггеры не используются, как ячейки хранения в чистом виде. Как они используются – рассмотрим в дальнейшем. А так много времени им было уделено в силу простоты их конструкции. Благодаря этому, мне удалось показать, как из обычных логических элементов, которые по определению ничего не помнят, удалось создать ячейку, которая помнит всё.

D-триггер

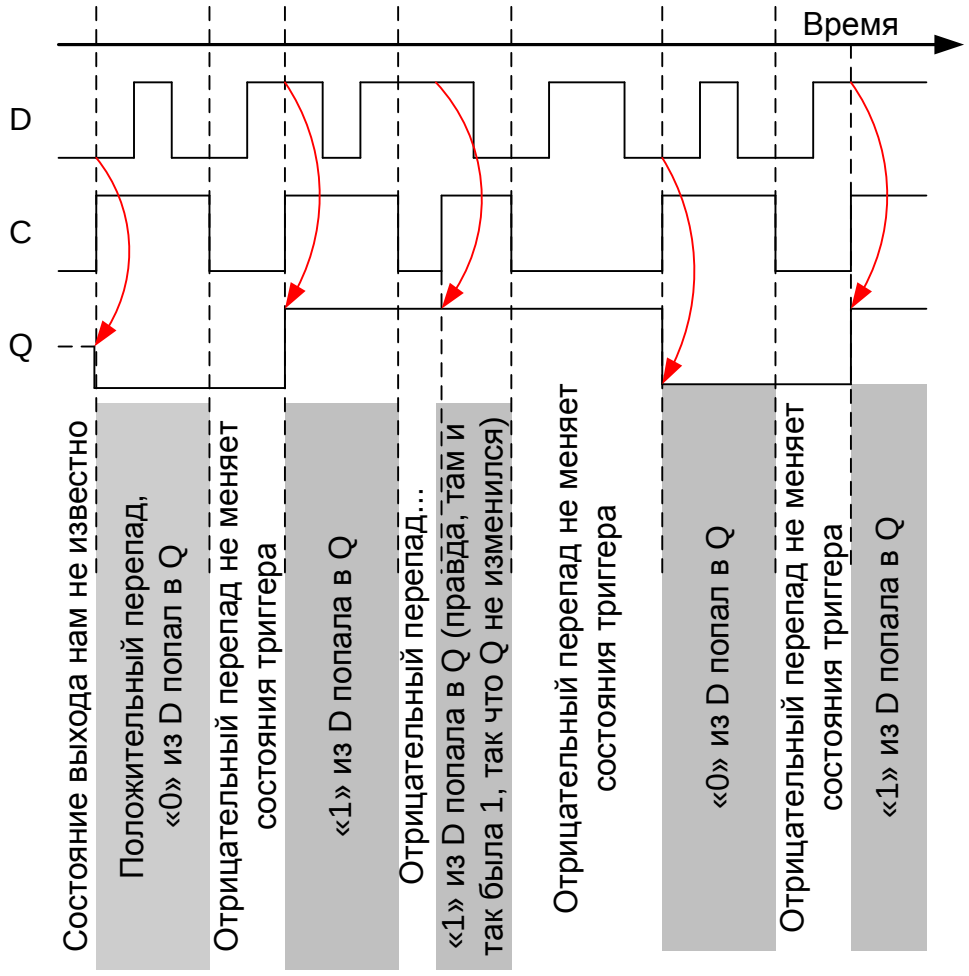
А если взять побольше логических элементов и объединить их похитрее, то можно создать конструкцию, которую называют D-триггером. Желающие найдут его внутреннюю структуру в справочниках, а так – я на одной лабораторной работе структуру зарисовал, понял, как она работает и… через неделю забыл. И это при том, что я

моделировал всё в виде мультфильма, а не на сухих словах. Поэтому, не будем впустую марать экраны, давайте договоримся, что такая штука имеется, что она тоже создаётся из кучки логических элементов, и что она работает. А коли работает, то будем ею пользоваться. Давайте взглянем на схемотехническое обозначение D-триггера:



Вариант 1 Вариант 2

Некоторые на входе C рисуют треугольник, некоторые – поднимающуюся палочку, что это такое? Это обозначение динамического входа. Сигнал со входа D перескочит на выход Q тогда и только тогда, когда вход C будет переходить из состояния 0 в состояние 1. Рассмотрим пример

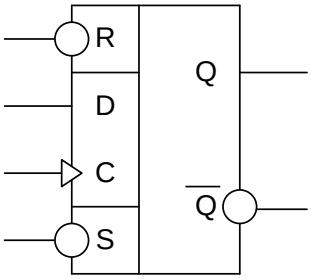


Как видим, выход Q изменяет своё значение только при переходе C из 0 в 1. Как бы ни колбасило вход D во всё остальное время, триггеру всё равно. Итого, с программистской точки зрения, вход «C» - это сигнал «Запомни переменную», а вход D – это значение, которое следует запомнить. Правда, если и другие точки зрения, но их мы рассмотрим как-нибудь попозже.

Совмещённый D и RS триггеры

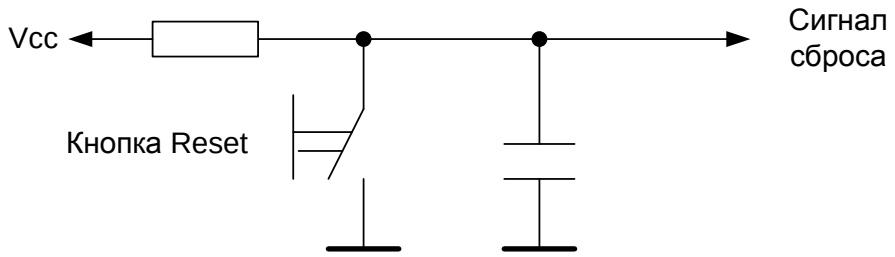
ВНИМАНИЕ! Всё, изложенное в этом разделе, касается классических микросхем и CPLD! Раньше я думал, что это касается и FPGA, но этой весной огрёб много проблем во время проекта PNRS, а детальные исследования показали, что в FPGA всё несколько примитивнее. Как именно – рассмотрим, если дойдём до этого. Но запомните! В FPGA нет совмещённых D и RS триггеров. Что есть – зависит от производителя, но именно D+RS нет, нет, нет, нет, нет! Не попадайтесь!

Самым популярным типом триггера является совмещённый вариант D и RS триггера.

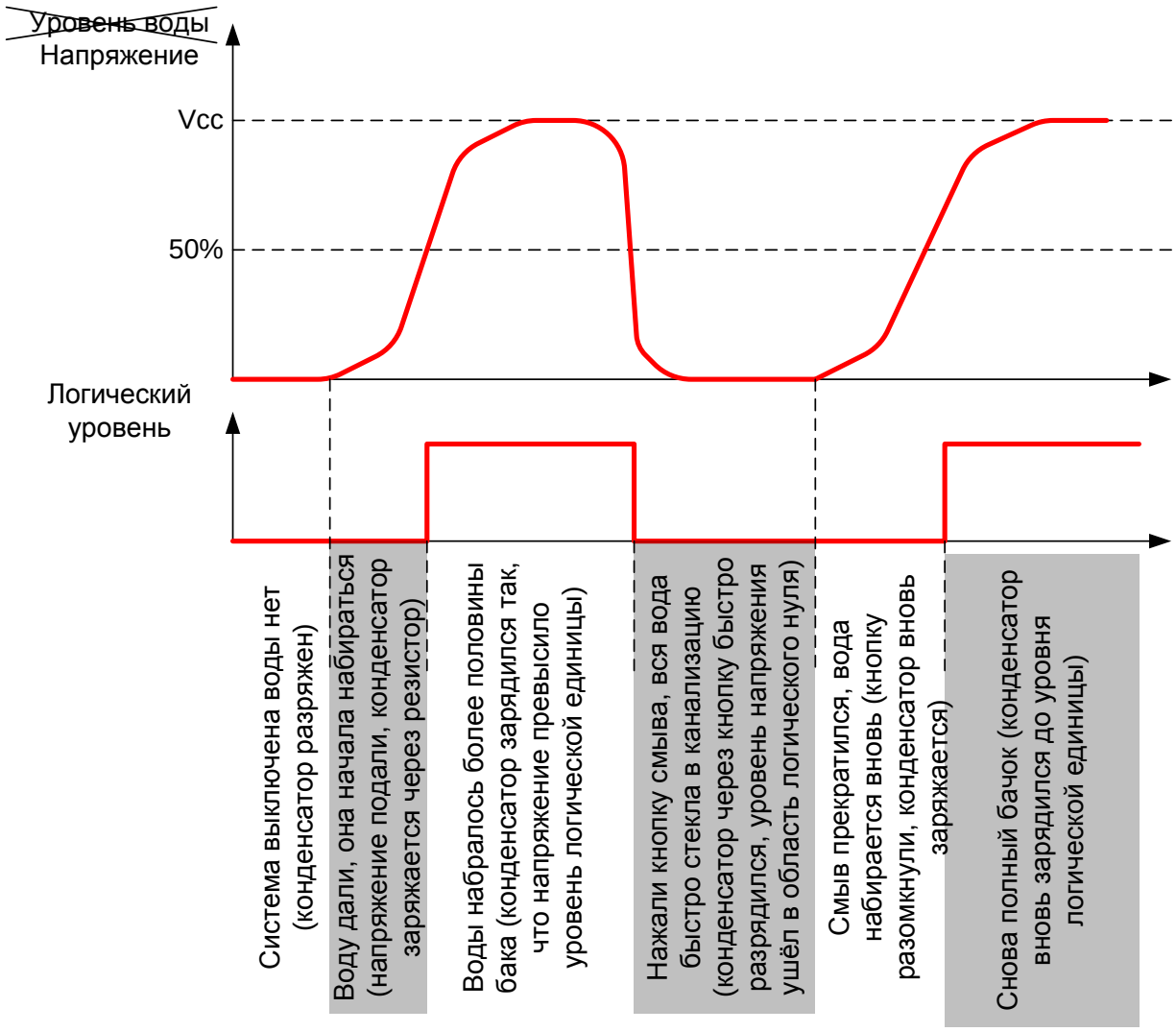


В чём достоинство такого триггера? Давайте рассмотрим самое очевидное (на самом деле, их больше, но будем рассматривать их по мере надобности). Обратите внимание на диаграмму, которую я нарисовал для D-триггера. В самом начале написано: «Состояние не определено». И оно будет не определено до тех пор, пока нам не понадобится заполнить переменную чем-либо. А это может произойти через час работы системы. А от выхода этого триггера может зависеть ещё целая цепочка триггеров. И вообще, а как заполнять самый первый триггер? И как потом гарантированно всё заполнить?

Вспоминаем наш многострадальный смывной бачок. Сегодня мы его нарисуем в том виде, в каком он реализован в любом настоящем компьютере…



Рассмотрим ~~уровень воды в бачке~~ диаграмму напряжения на линии «Сигнал сброса»



Итак, мы видим, что в нормальном состоянии на линии «Сигнал сброса» у нас логическая единица, а в момент включения питания, и когда нажата кнопка Reset, там гарантировано будет логический ноль. Внимательно смотрим на триггер. А там входы R и S – тоже управляются нулём. Вот и замечательно! Какому триггеру при старте системы надо занести ноль – его подкючаем входом R к линии сброса, а какому единицу – его подключаем линией S. И всё. Теперь при старте нашей схемы, все триггеры получат значения по умолчанию, ну прямо как переменные в языке Си.

Именно так инициализируется компиютер, правда, там ещё и счётчик адреса программы в ноль сбрасывается. А что такое счётчик и как он устроен, мы рассмотрим в следующий раз.