Лекция №8
по дисциплине
Моделирование

Группы

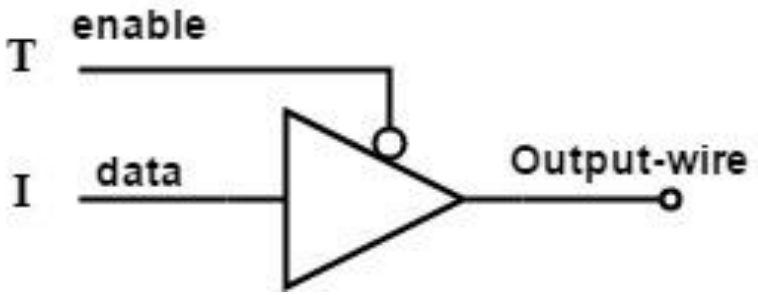
ИВБО-01-17

ИВБО-02-17

Моделирование комбинационных цепей на языке Verilog HDL

Универсальное однозначно синтезируемое описание элемента с тремя выходными состояниями – буфера TRI-state.

Интерфейс и УГО элемента, а также таблица истинности :



T - управление	I – вход данных	O – выход данных
1	X	Z
0	0	0
0	1	1

Описание с помощью оператора assig

assign <output_wire> = <enable> ? 1'bz : <data>;

Выходной сигнал <output_wire> должен иметь тип wire.

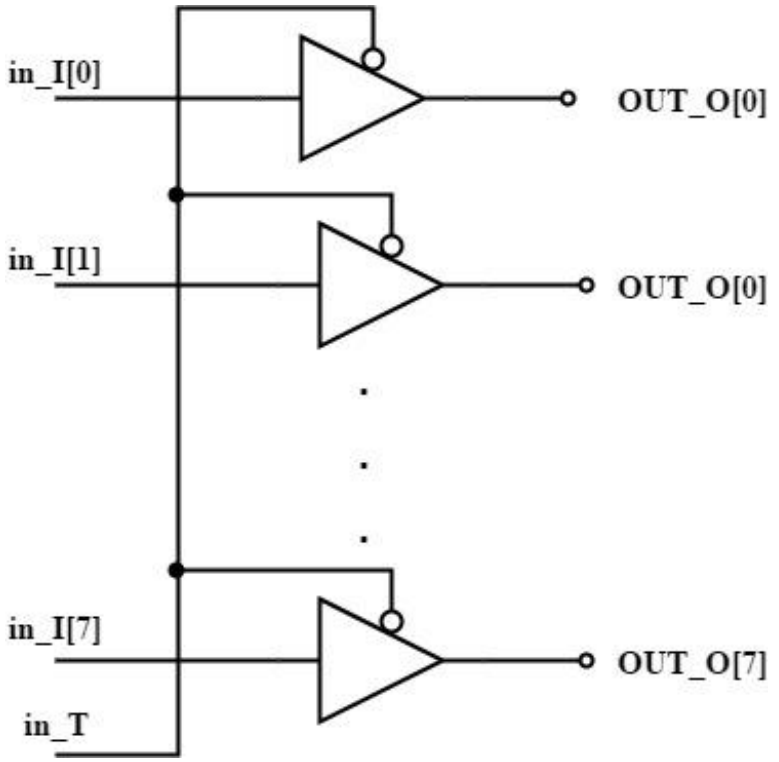
Входные сигналы <enable> и <data> могут быть любого типа: wire или reg.

Для описания разделяемой линии связи, соединяющей выходы нескольких элементов, необходимо использовать несколько операторов assign, присваивающих значение одному и тому же сигналу <output_wire>. Этот случай многократного присвоения значений следует считать исключением. В прочих ситуациях такой подход является неграмотным и может привести к ошибкам синтеза схемы.

Пример описания 8-разрядного шинного формирователя (буфера)

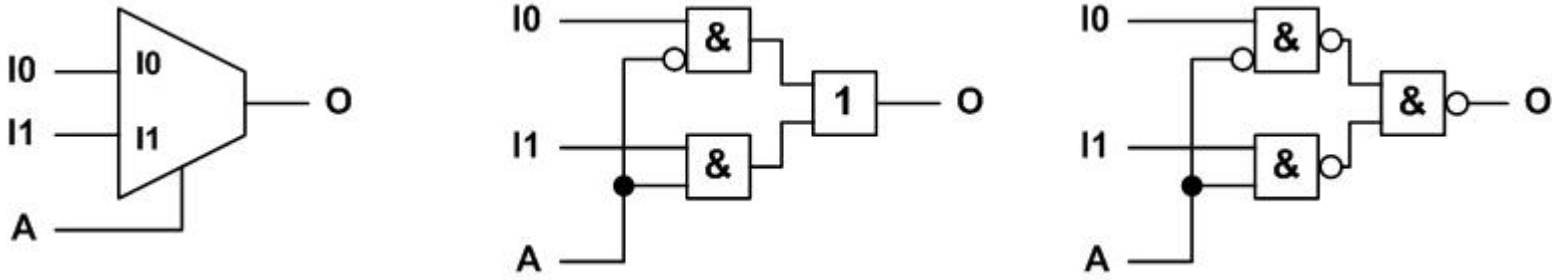
Шинный формирователь представляет собой элемент, состоящий из нескольких буферов TRI-state с общим управлением и независимыми сигналами данных.

```
`timescale 1us/1 ps
module BUFT_8_EXAMPLE(
input         in_T;
Input  [7…0] in_I,
output [7…0] OUT_O);
assign OUT_O = in_T ? 8'hZZ : in_I;
endmodule
```



Внутренняя организация и синтезируемая модель мультиплексора 2:1

Мультиплексор 2:1 является комбинационной схемой, транслирующей на выход значение одного из двух входных сигналов. Имеет интерфейс и внутреннюю организацию согласно рисунку, и функционирует по таблице истинности, приведённой ниже:



I0	I1	A	O
0	X	0	0
1	X	0	1
X	0	1	0
X	1	1	1

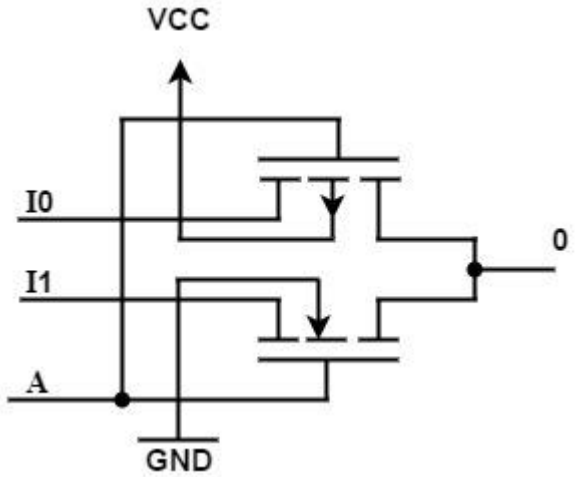
Описание мультиплексора 2:1 с помощью оператора assign языка Verilog:

assign <output_wire> = <address> ? <input_1> : <input_0>;

Выходной сигнал <output_wire> должен иметь тип wire.

Входные сигналы <address>, <input_0> и <input_1> могут быть любого типа: wire или reg.

Простейший мультиплексор 2:1 можно построить на основе двух комплементарных МОП транзисторов по следующей схеме:



В зависимости от адресного входного сигнала открыт один из транзисторов, а другой транзистор закрыт. Открытый транзистор выполняет функцию электронного ключа, замыкающего один из информационных входов с выходной цепью.

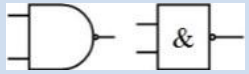
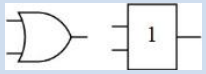
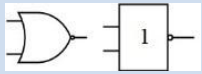
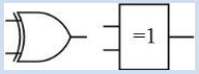
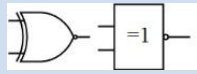
Следует отметить, что эта схема является двунаправленной, и может пропускать сигнал как слева направо, так и справа налево. Поэтому её часто называют «аналоговый ключ». Тем не менее, нет препятствий использования такого решения в цифровой схемотехнике.

Логические примитивы языка Verilog HDL

В языке Verilog помимо логических операторов:

- инверсия - ~ ,
- И - & ,
- ИЛИ - | ,
- исключающее ИЛИ (mod2) - ^

есть примитивы – элементы, предопределённые стандартом языка. В таблице приведены многовходовые логические примитивы с равнозначными входами:

Примитив	and	nand	or	nor	xor	xnor
Элемент						
Функция	0X|0 X0|0 11|1	0X|1 X0|1 11|0	1X|1 X1|1 00|0	1X|0 X1|0 00|1	10|1 01|1 00|0 11|0	00|1 11|1 01|0 10|0

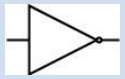
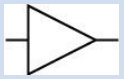
Общий синтаксис для многовходовых примитивов с равнозначными входами следующий:

```
and
nand
or
nor
xor
xnor          <имя_примитива>      (выход,вх,вх,вх…);
```

ключевое слово языка Verilog - название примитива	необязательное имя экземпляра примитива	список цепей начинается с выхода, затем входы

Выходной сигнал примитива должен иметь тип wire.
Входные сигналы могут быть любого типа: wire или reg.

Примитивы с одним входом представлены инвертором и буфером
с выходами типа push-pull:

Примитив	not	buf
Элемент		
Функция	1|0 0|1	0|0 1|1

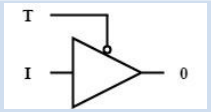
Общий синтаксис примитивов с одним входом следующий:

```
not
buf                    <имя_примитива>        (выход,вх);

ключевое слово         необязательное         список цепей
языка Verilog -        имя экземпляра         начинается с выхода,
название               примитива              затем вход
примитива
```

Выходной сигнал примитива должен иметь тип wire.
Входной сигнал может быть любого типа: wire или reg.

Примитив языка Verilog HDL, определяющий буфер TRI-state:

Примитив	bufif0
Элемент	
Функция	1X|Z 00|0 01|1

Синтаксис примитива буфера TRI-state:

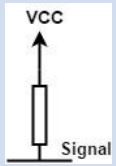
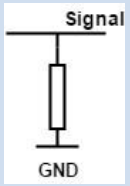
```
bufif0              <имя_примитива>      (выход, вх_данных, вх_упр);

ключевое слово      необязательное       список цепей
языка Verilog -     имя экземпляра       начинается с выхода,
название            примитива            затем вход данных,
примитива                                и третий – управляющий вход
```

Выходной сигнал примитива должен иметь тип wire.
Входные сигналы могут быть любого типа: wire или reg.

Некоторые синтезаторы не поддерживают данный примитив! Не рекомендуется его использовать.

Язык Verilog определяет примитивы, описывающие активную резистивную или ёмкостную нагрузку:

Примитив	pullup	pulldown
Элемент		
Функция	Подтяжка линии связи к лог. «1»	Подтяжка линии связи к лог. «0»

Синтаксис применения примитива pullup/pulldown следующий:

```
pullup
pulldown          (<мощность>)     <имя_примитива>  (signal);
ключевое слово    необязательное   необязательное   список цепей:
языка Verilog -   значение         имя экземпляра   название
название          мощности         примитива        линии связи
примитива         примитива
```

Сигнал примитива должен иметь тип wire.

Параметр <мощность> определяет величину сопротивления или ёмкости примитивов pullup и pulldown:

Мощность по убыванию	Подтяжка к лог. «1»	Подтяжка к лог. «0»	Тип нагрузки
7	supply1	supply0	резистивная
6	strong1	strong0	резистивная
5	pull1	pull0	резистивная
4	large		ёмкостная
3	weak1	weak0	резистивная
2	medium		ёмкостная
1	small		ёмкостная
0	highz1	highz0	резистивная

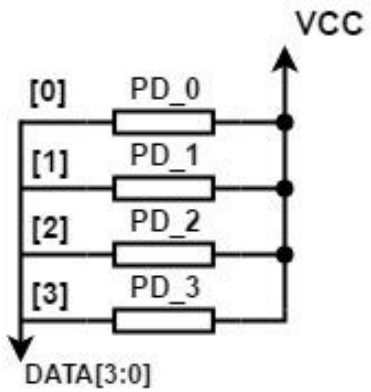
Некоторые синтезаторы не поддерживают данный примитив!
Не рекомендуется его использовать.

Примеры использования примитивов языка Verilog HDL

Блок подтягивающих резисторов на 4-разрядной шине данных:

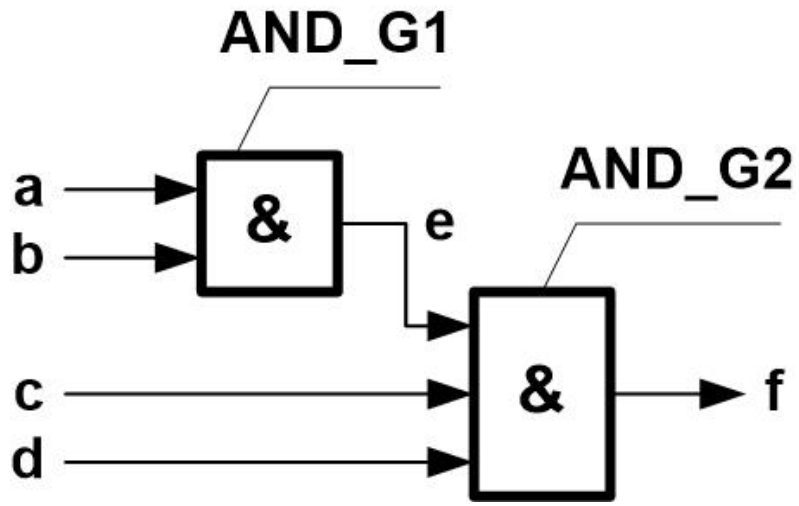
```
pullup (pull1) PD_0 (DATA[0]), PD_1(DATA[1]),
               PD_2 (DATA[2]), PD_3(DATA[3]);
```

Эквивалентная схема:



Следует отметить синтаксическую возможность определения нескольких примитивов одного типа единым выражением языка Verilog, начинающимся с ключевого слова – названия примитива и заканчивающегося точкой с запятой.

Пример описания каскадной комбинационной схемы на примитивах:



Такую схему можно описать следующими способами:

1. and AND_G1(e, a, b), AND_G2(f, e, c, d);
2. and AND_G1 (e,a,b);
   and AND_G2 (f,e,c,d);
3. and (e,a,b);
   and (f,e,c,d);
4. and (e, a, b), (f, e, c, d);
5. and (f, a, b, c, d);

Классификация функциональных элементов со сложным поведением – триггеров и регистров

Статические, или защёлки, или триггера с потенциальным управлением:

- асинхронный R-S триггер;
- асинхронный D триггер типа «защёлка» или Latch;
- регистр из асинхронных триггеров.

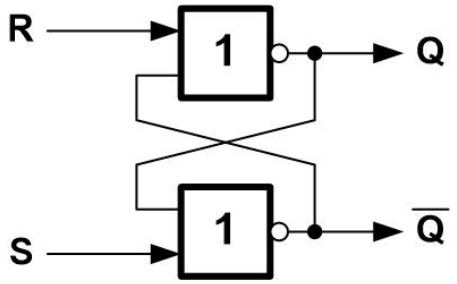
Динамические, или триггера с управлением по фронту, или тип Flip-Flop:

- динамический D триггер или D-type Flip-Flop;
- синхронный RS триггер Synchronous RS Flip-Flop;
- счётный T триггер или T-Flip-Flop;
- JK триггер или JK-Flip-Flop;
- регистр из триггеров, управляемых по фронту;
- сдвиговый регистр вправо, влево, реверсивный;
- бисинхронный RS триггер.

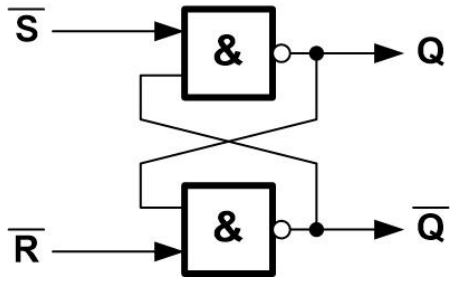
Самый простой функциональный элемент, обладающий сложным поведением, представлен асинхронным RS триггером

Существует два варианта асинхронного RS триггера:

1. С активным «действующим» значением входных сигналов лог. «1».
   Строится на двух элементах ИЛИ-НЕ по следующей схеме:



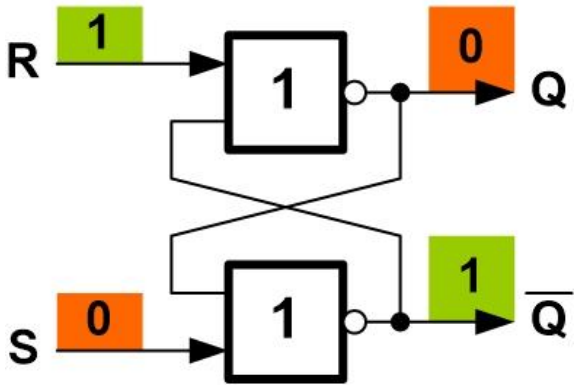
2. С активным «действующим» значением входных сигналов лог. «0».
   Строится на двух элементах И-НЕ по следующей схеме:



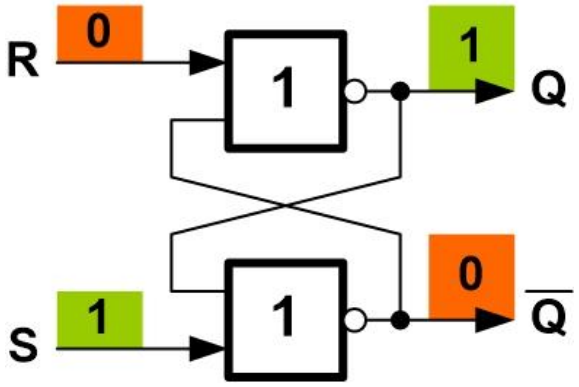
Каждая из двух схем асинхронного RS триггера во время штатной работы может находиться в четырёх состояниях.

Рассмотрим состояния для первой схемы:

1. Установка в состояние «0»:

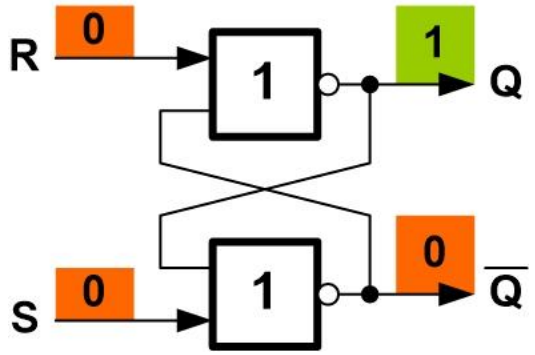


2. Установка в состояние «1»:

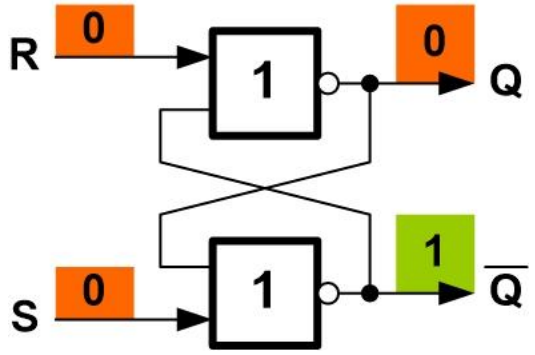


Сложное поведение триггера проявляется в режиме хранения, когда одно и то же пассивное входное воздействие может соответствовать двум различным состояниям схемы, определяющим выходное воздействие:

3. Хранение лог. «1»:

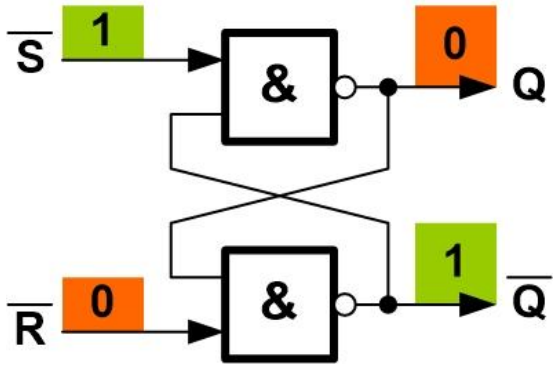


4. Хранение лог. «0»:

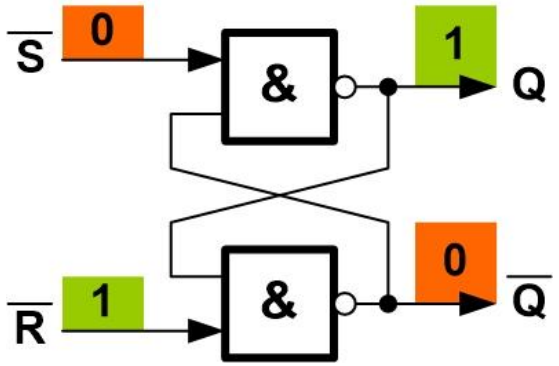


Вторая схема асинхронного RS триггера управляется подачей на входы лог. «0». Рассмотрим её состояния штатной работы:

1. Установка в состояние «0»:

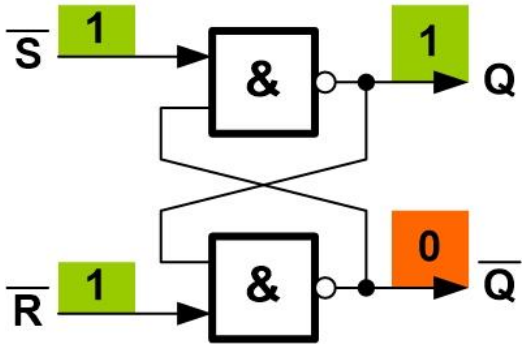


2. Установка в состояние «1»:

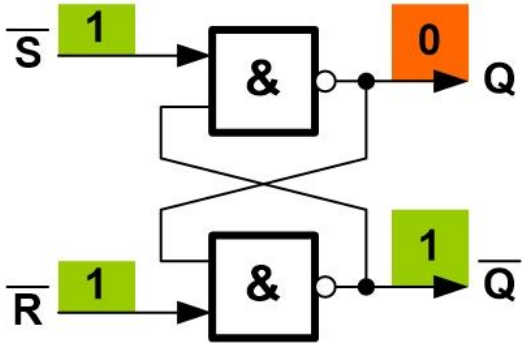


Пассивным входным воздействием для второй схемы является подача комбинации «11»:

3. Хранение лог. «1»:

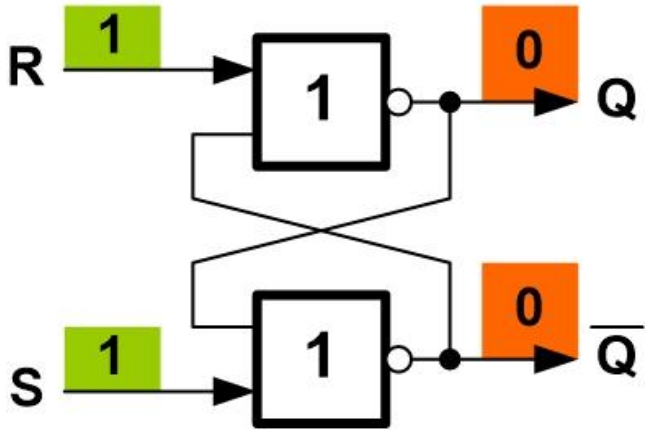


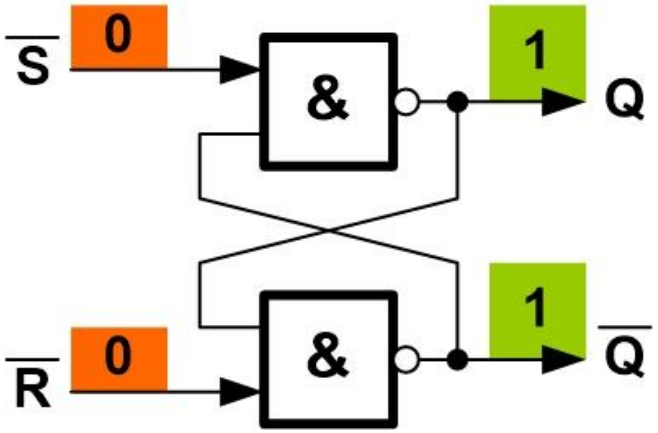
4. Хранение лог. «0»:



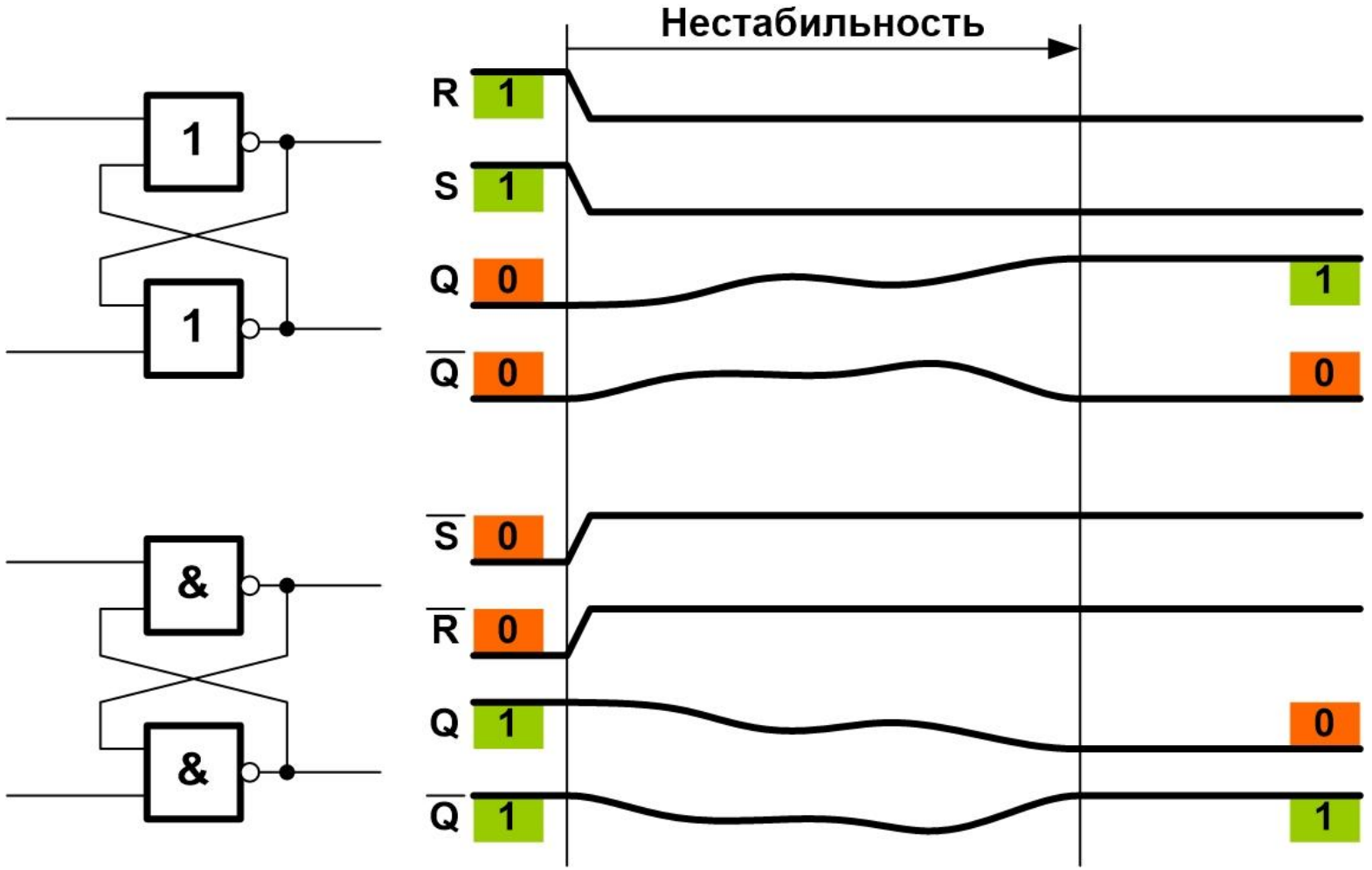
Следует отметить, что подача активных входных значений на оба входа как для первой, так и для второй схемы триггера не выводит его из строя, однако не относится к штатному режиму работы. При смене входного воздействия «оба активные» на «оба пассивные» триггер попадает в метастабильное состояние, при котором в схеме происходит гонка: какому плечу достанется лог. «0», а какому – лог. «1». Эта ситуация напоминает балансирующее в равновесии перевёрнутое коромысло и может длиться значительное время относительно скорости работы схемы. Возникновение такой ситуации в устройстве необходимо минимизировать или устранить.

Запрещённое в штатном режиме работы триггера пятое состояние:

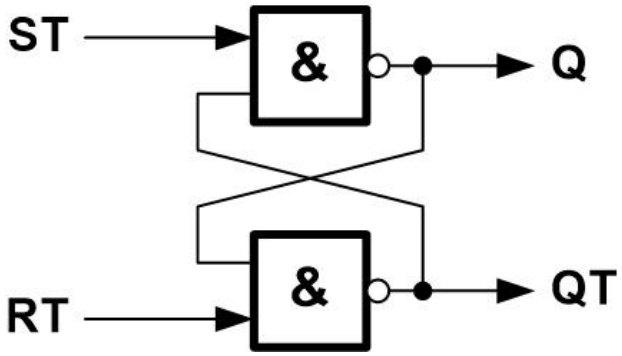




Процесс попадания триггера в метастабильное состояние:



Модель асинхронного RS триггера на примитивах языка Verilog HDL:




```
nand (Q, ST, QT), (QT, RT, Q);
```

Следует отметить, что при проектировании устройств на ПЛИС этот пример считается неграмотным, ибо вызывает петлю на ресурсах комбинационной логики – combinatorial loop.

На практике схема асинхронного RS триггера может использоваться для подавления дребезга контактов от кнопки с нормально замкнутой и нормально разомкнутой парами контактов (при нажатии один контакт отсоединяется от общего контакта, а другой соединяется) или тумблера следующим образом: