AQUARIUS

САПР. Основы симуляции Verilog

Матвеев Борис

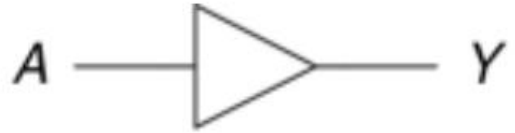
Старший инженер направления функциональной верификации

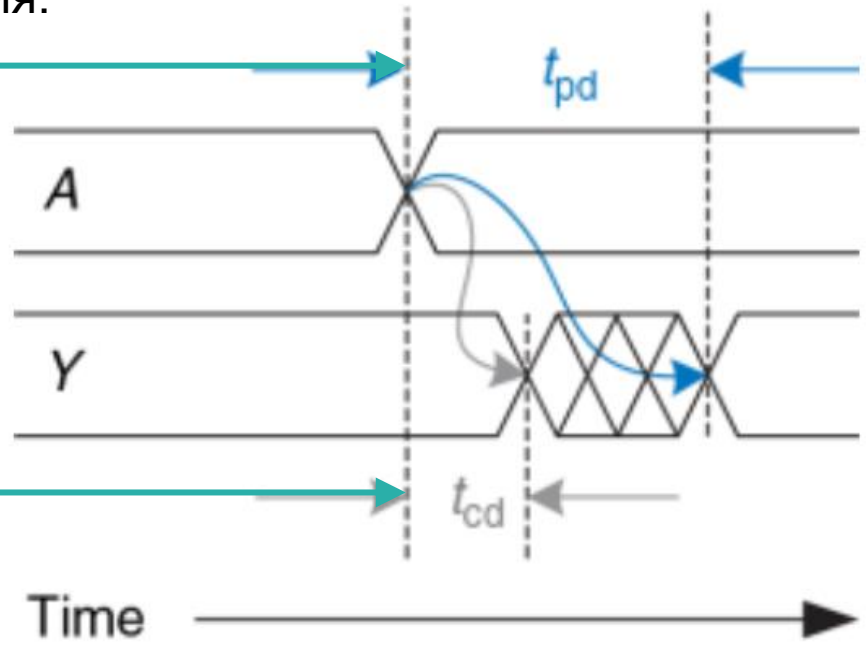
AQUARIUS

Основы симуляции Последовательностной и комбинационной логики

Комбинационная логика

Комбинационные схемы имеют временные задержки. Время, в течение которого **выходные данные** либо **неактуальны**, либо могут принимать **случайные** значения.





tpd – задержка распространения (propagation delay).

*Максимальное время от момента изменения входа схемы до момента, когда все **выходы** достигнут **установившихся** значений.*

tcd – задержка реакции (contamination delay).

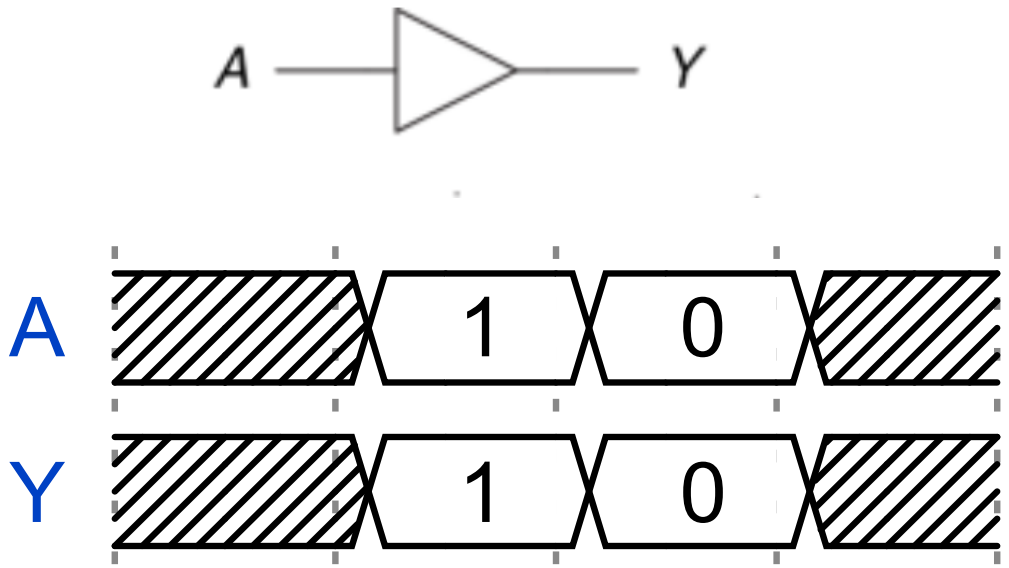
*Время от момента изменения входа схемы до момента **изменения** любого из **выходов**.*

*Иллюстрация из книги Харрис и Харрис «Цифровая схемотехника и архитектура компьютера»

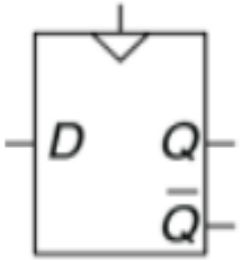
Комбинационная логика

В идеальном случае скорость работы комбинационной логики настолько высока, что задержками можно пренебречь.

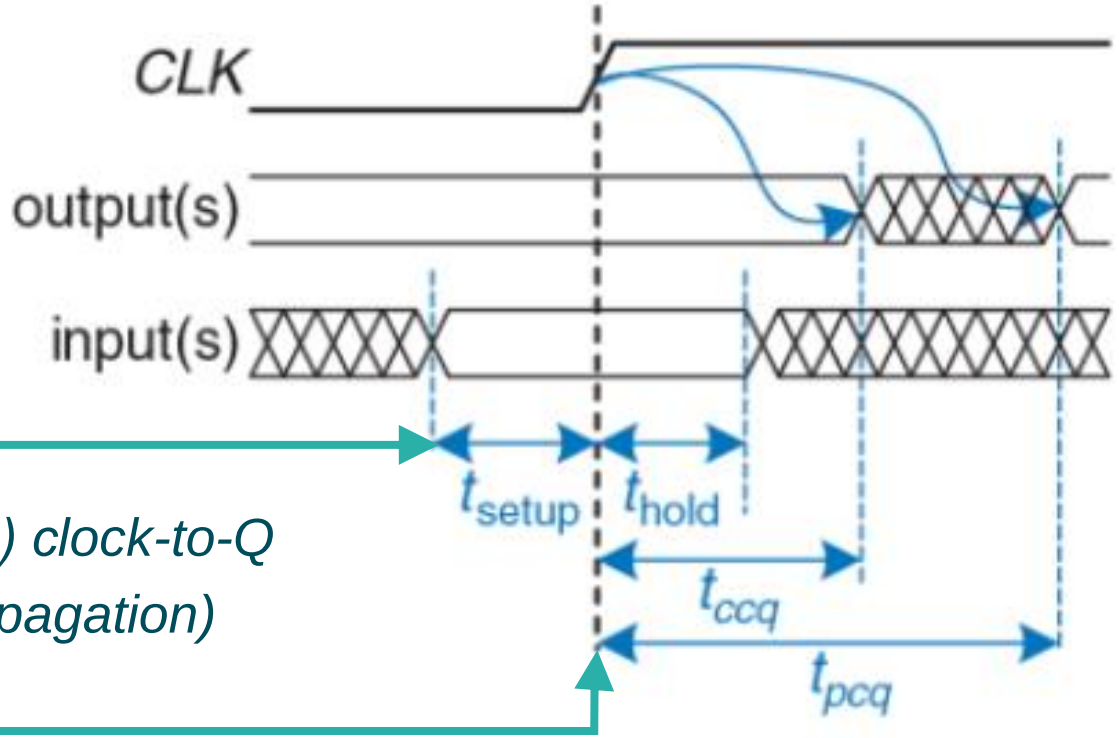
Поэтому в симуляции **выходы комбинационной логики меняются мгновенно** в соответствии с изменениями входов схемы.



Динамические характеристики



Изменения выходов любого триггера происходят с некоторой задержкой относительно тактового сигнала.(tccq)



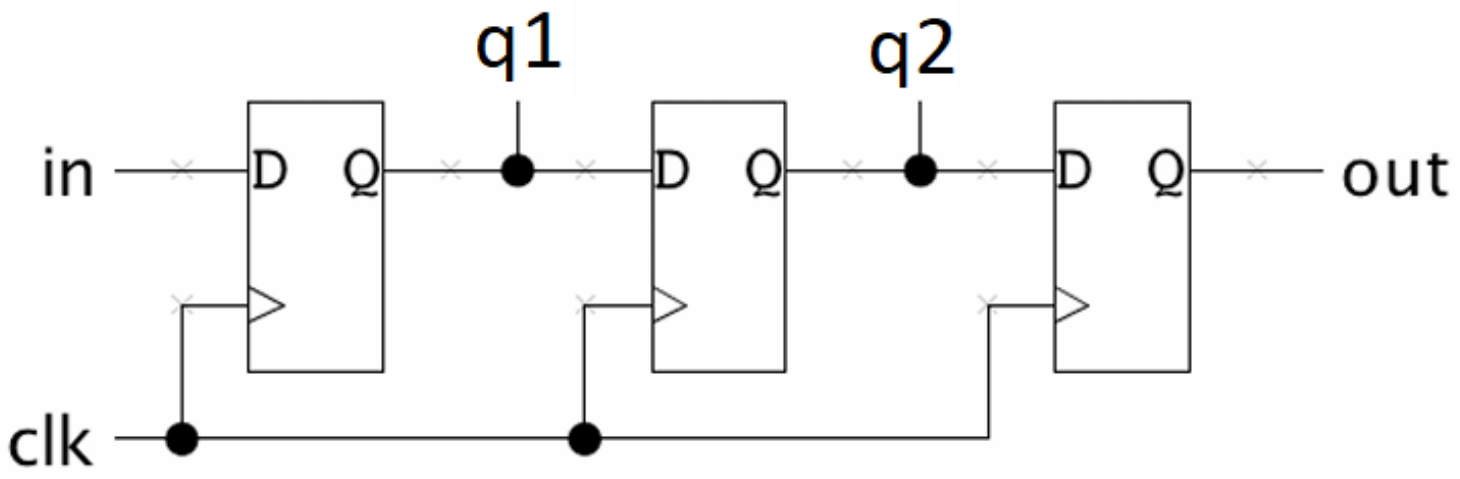
- **t**setup – время предустановки
- **t**hold – время удержания

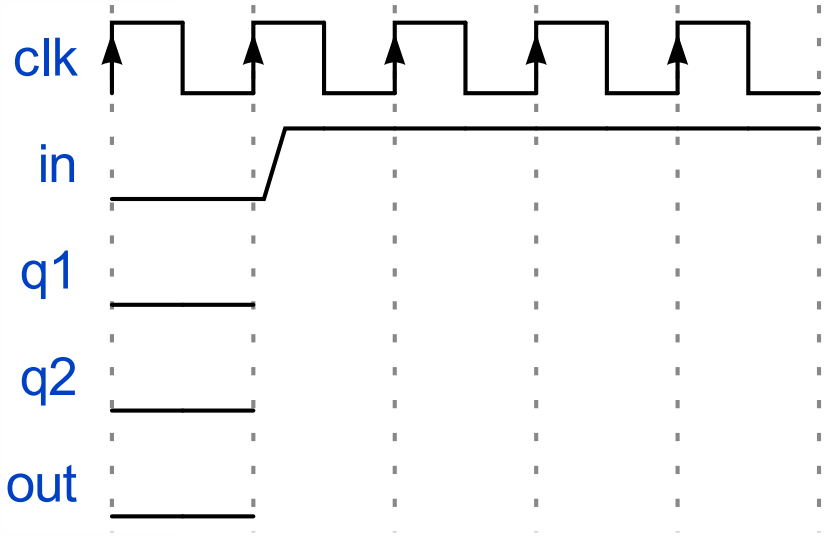
- ***t**ccq – задержка реакции (contamination) clock-to-Q*
- ***t**pcq – задержка распространения (propagation) clock-to-Q*

*Иллюстрация из книги Харрис и Харрис «Цифровая схемотехника и архитектура компьютера»

Последовательностная логика

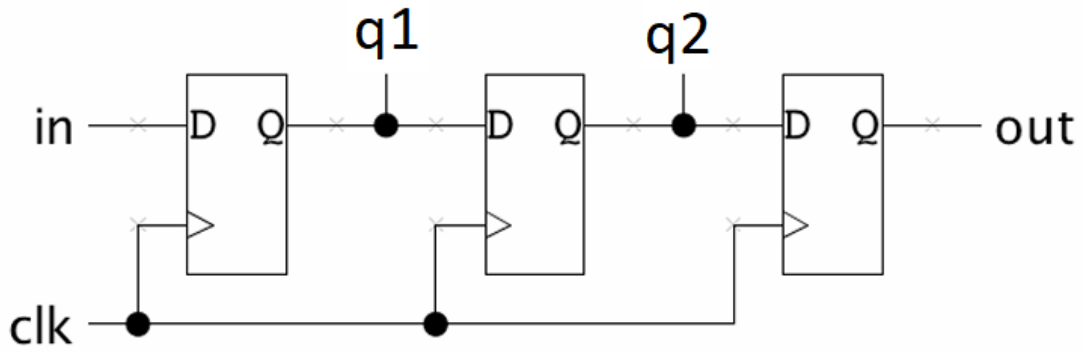
Нарисуйте временную диаграмму выходов q1, q2, out:



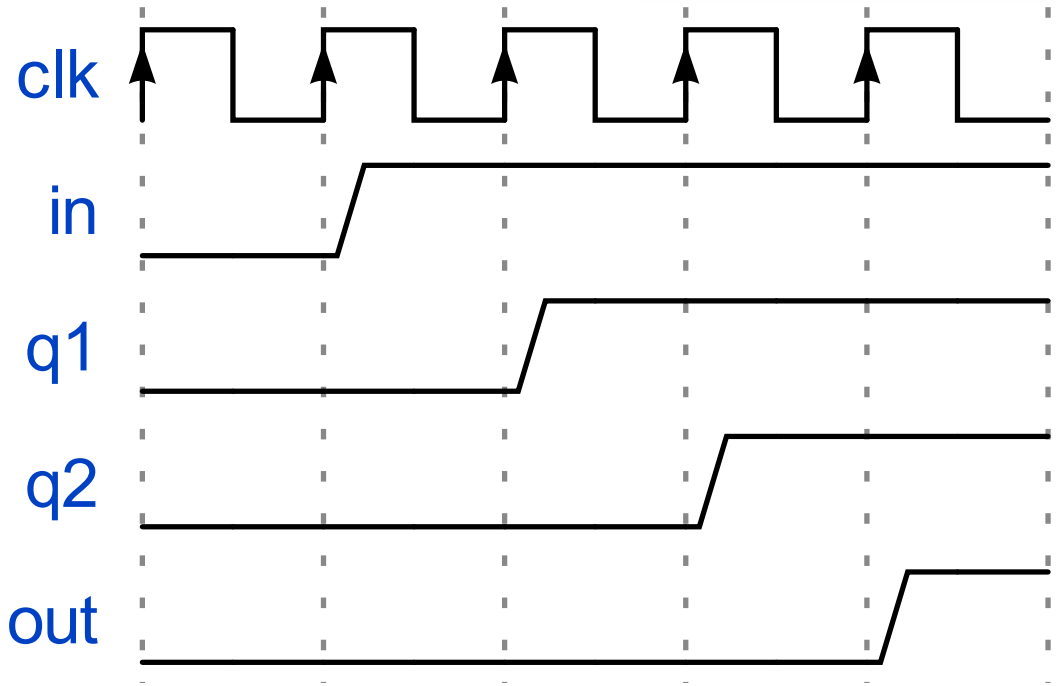


Последовательностная логика

Данные записываются только на следующий такт из-за наличия задержек между изменением входа и выхода триггеров

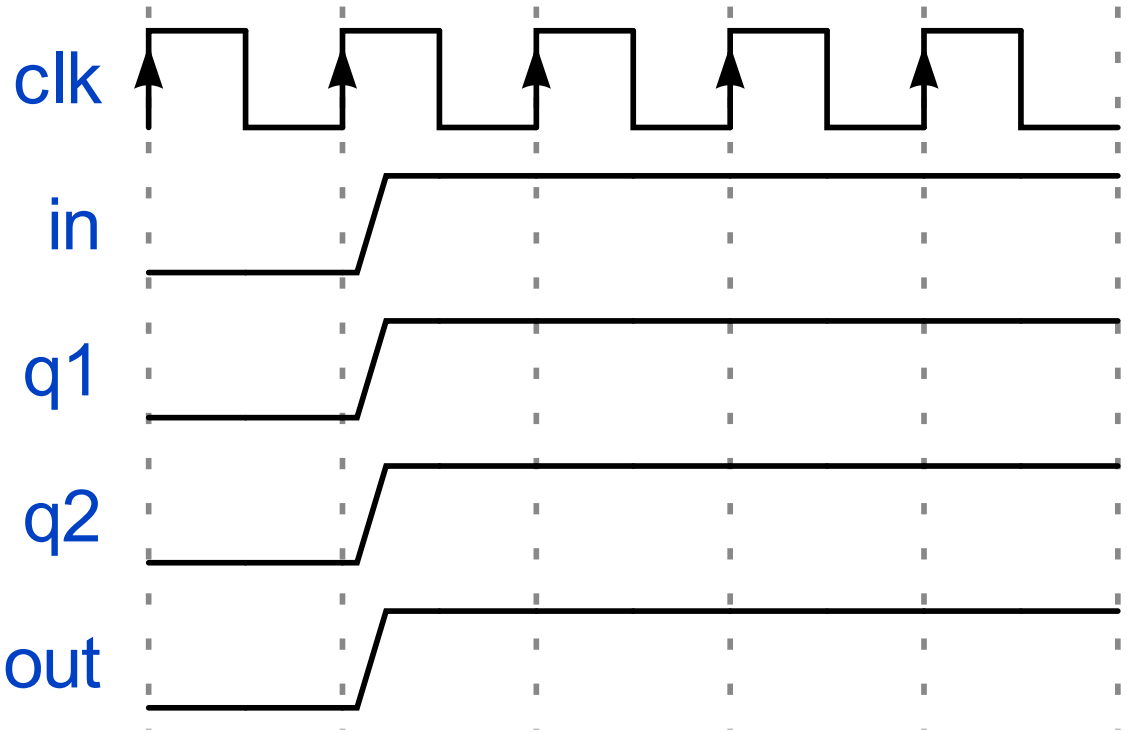


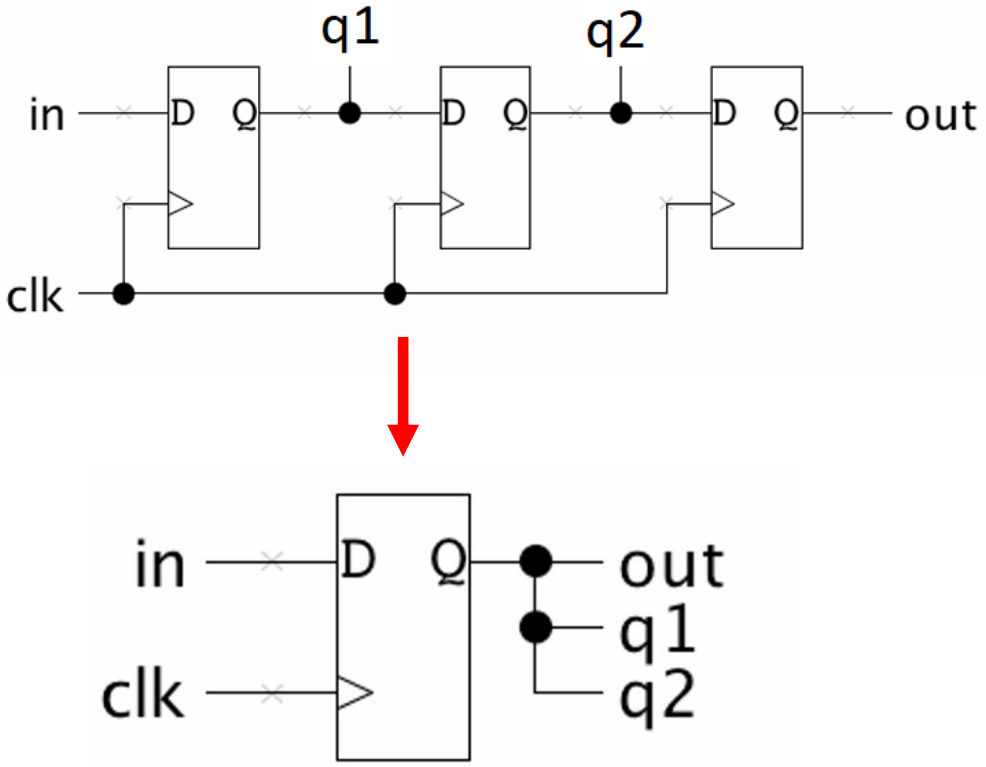
Т.е. в текущем такте записывается «старое» значение входа



Последовательностная логика

А что будет, если в Последовательностной логике вовсе не учитывать задержку?



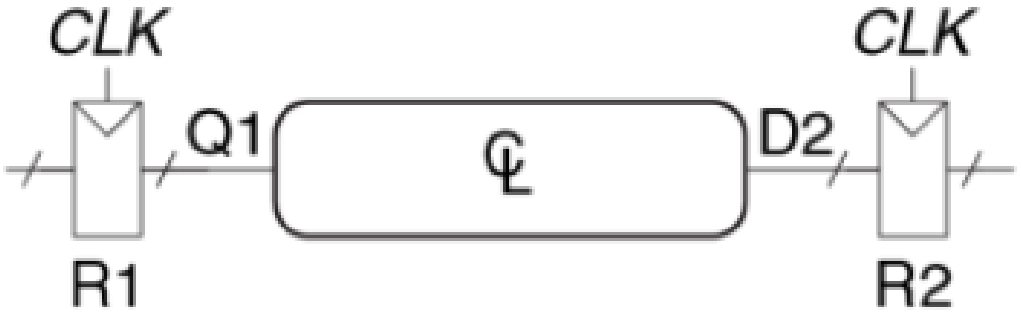


Временная диаграмма изменится и не будет соответствовать действительности

Смешанная логика

Вывод

Большинство схем представляют собой смешение Последовательностной и комбинационной логики, как представлено на рисунке.



При симуляции без учета физических особенностей использованных элементов:

1. Для **комбинационной логики допустимо мгновенное изменение выходов** в соответствии с изменениями её входов.

2. Для **Последовательностной логики недопустимы** подобные упрощения.

AQUARIUS

Регионы симуляции

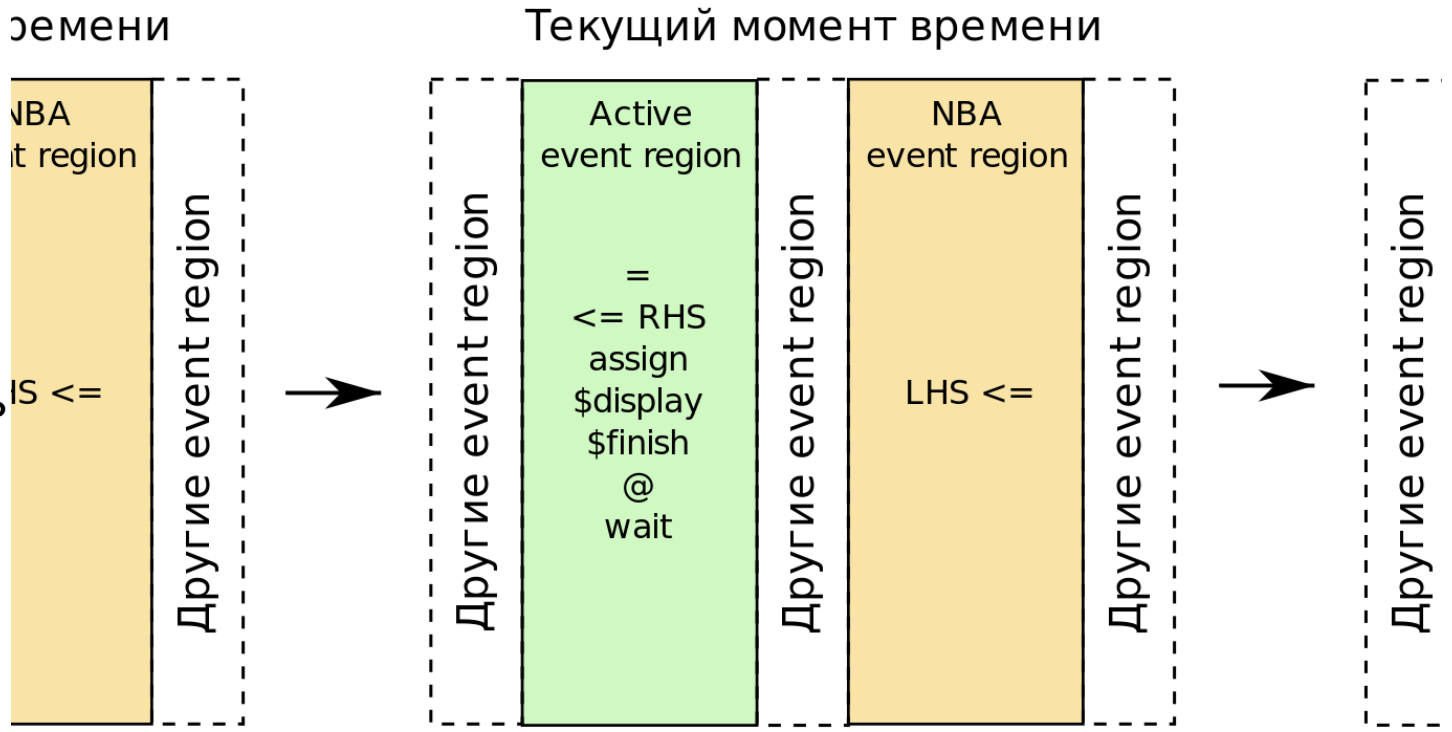
Регионы симуляции

Вывод

Для решения проблемы симуляции Последовательностной логики в SystemVerilog **каждый такт симуляции (дельта-цикл) разбивается на набор регионов**.

Для упрощения мы будем рассматривать только **активный регион** и **регион неблокирующих присваиваний (NBA).**

*Размер такта можно задать через `timescale или передать в симулятор через командную строку



*Изображение Сергея Чусова
https://github.com/serge0699

Активный регион vs NBA

Активный регион [a = b]	NBA (non-blocking assignment) [a <= b]
Симуляция комбинационной логики* **блокирует** переход из активного региона в NBA, пока не произойдут все необходимые вычисления и присвоения. Поэтому этот тип присвоения называется **блокирующим**	NBA исполняется после того, как все процессы в активном регионе завершены.
Вычисление **только правой части** неблокирующих присваиваний. (**входов** Последовательностной логики)	Присваивает выражению с **левой части** значение выражения с правой части неблокирующего равенства. Т.е. устанавливает значения **выходов*** Последовательностной логики
Триггер исполнения блоков событий @	

*Выходы **комбинационной** логики являются входами Последовательностной логики, поэтому **в активном регионе** производятся вычисления для комбинационной логики и для **входов Последовательностной** логики

*Выходы **Последовательностной** логики могут быть установлены только после установки значений на её входах.

*Важно отметить, что вычисления производятся для всех узлов схемы одновременно.

Пример схемы

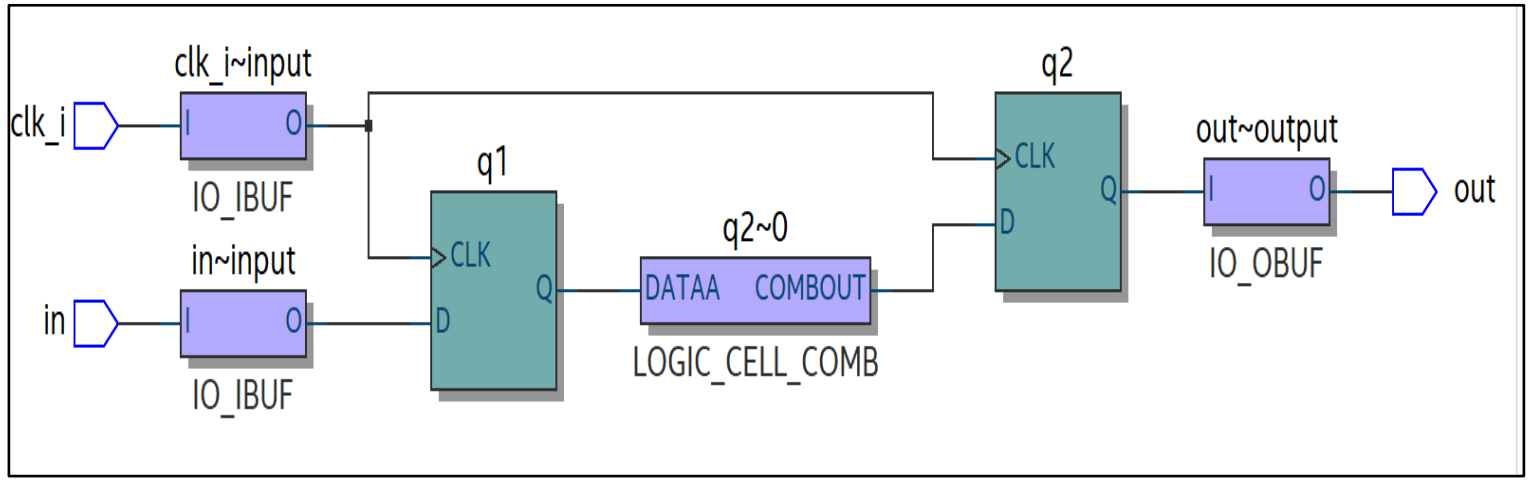
```
module mixed_logic(
  input clk_i
  ,input in
  ,output out
);
  wire comb_in, comb_out, d1, d2;
  reg q1;
  reg q2;

  assign d1 = in;
  assign comb_in = q1;
  assign d2 = comb_out;
  assign out = q2;

  always_comb begin
    comb_out = comb_in^1;
  end

  always @(posedge clk_i) begin
    q1 <= d1;
  end

  always @(posedge clk_i) begin
    q2 <= d2;
  end
endmodule
```

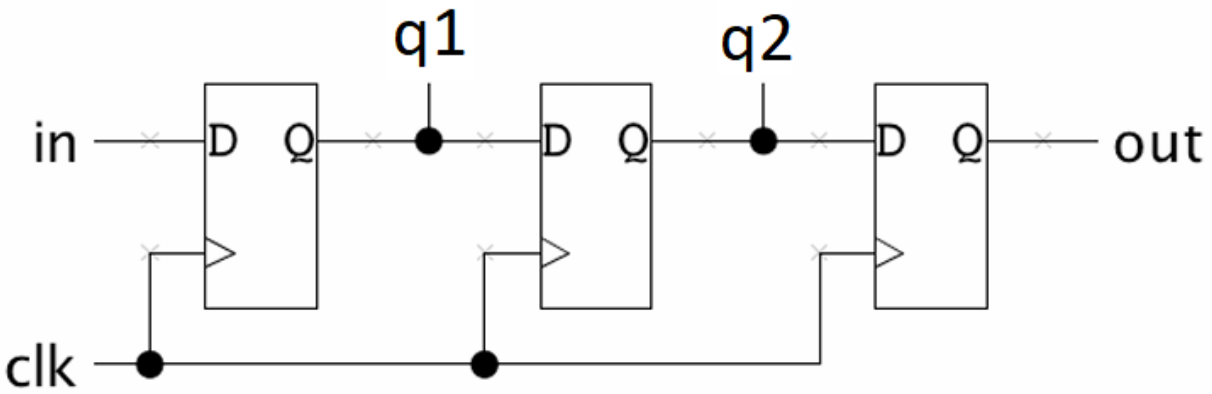


Активный регион

Non-blocking assignment region (NBA)

Пример схемы

```
module nonblocking(
  input in, clk,
  output reg out
);
  reg q1, q2;
  always @(posedge clk) begin
    q1  <= in;
    q2  <= q1;
    out <= q2;
  end
endmodule
```



Активный регион

Non-blocking assignment region (NBA)

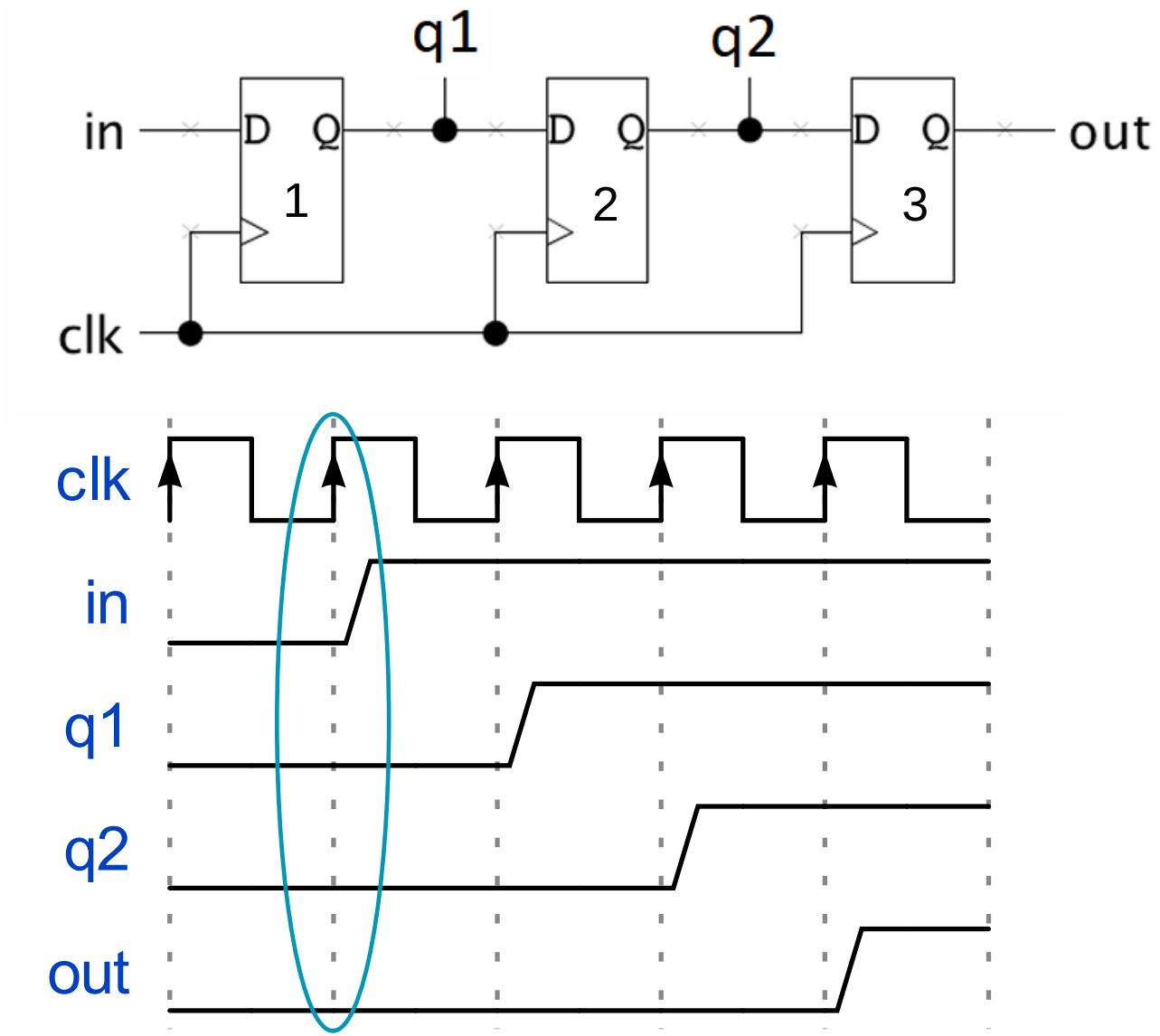
Пример

Активный регион:

```
always @(posedge clk) begin
  q1  <= in;
  q2  <= q1;
  out <= q2;
end
```

In = 0 out = 0
D1 = 0 Q1 = 0
D2 = 0 Q2 = 0
D3 = 0 Q3 = 0

*in = 0, так как он подается с синхронной схемы через «<= » в тестбенче

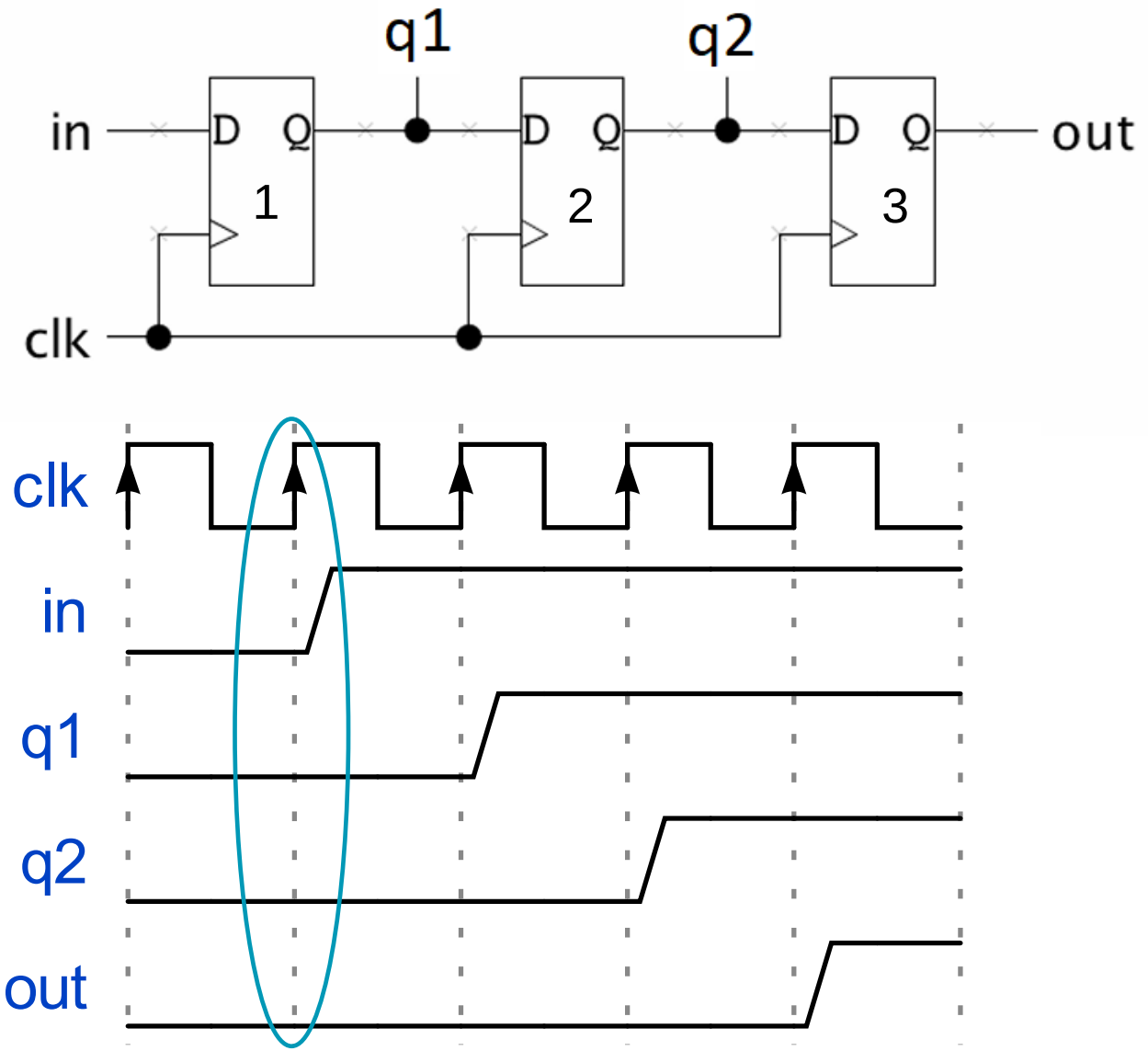


Пример

NBA регион:

```
always @(posedge clk) begin
  q1  <= in;
  q2  <= q1;
  out <= q2;
end
```

In = 1 out = 0
D1 = 0 Q1 = 0
D2 = 0 Q2 = 0
D3 = 0 Q3 = 0

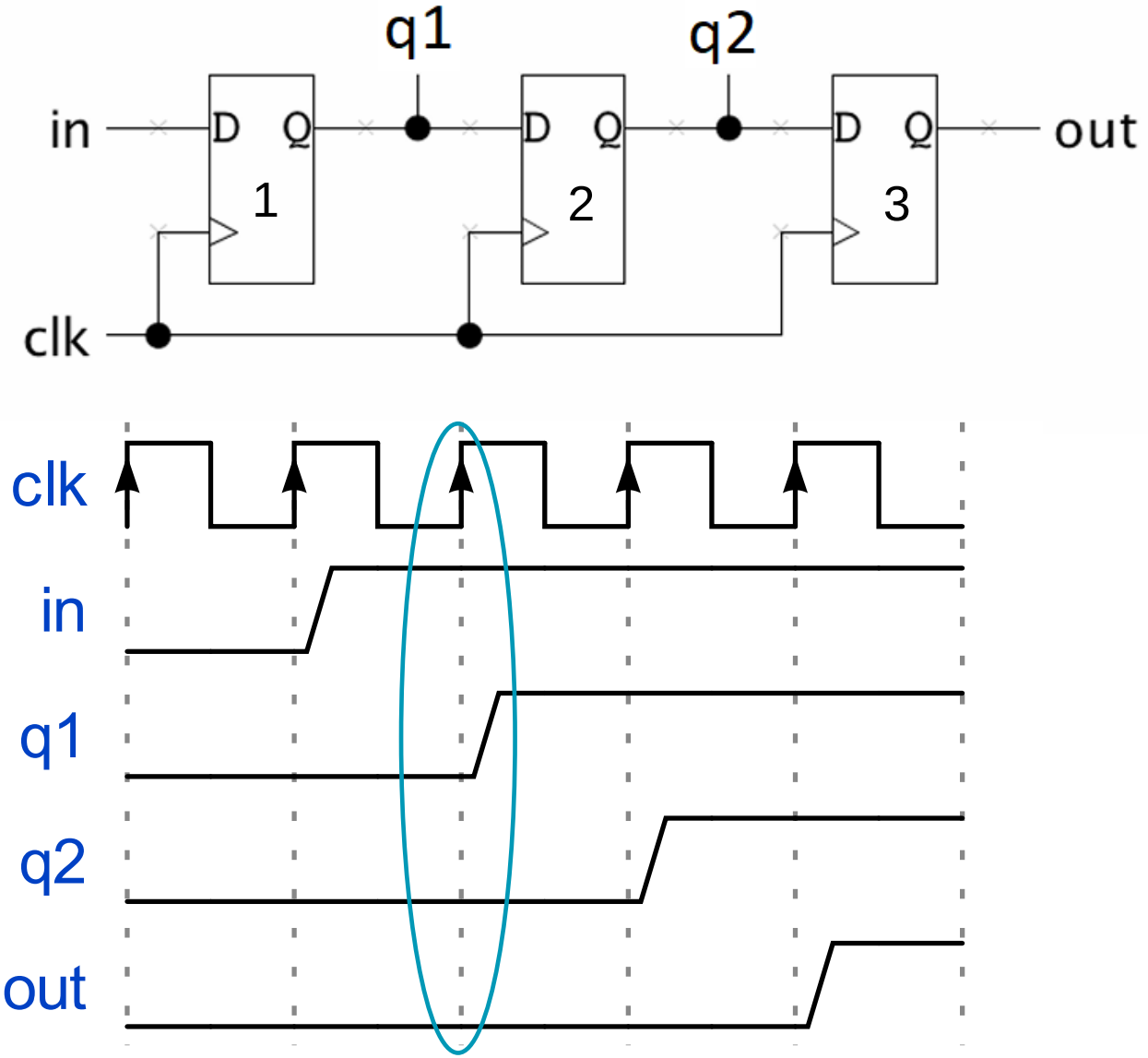


Пример

Активный регион:

```
always @(posedge clk) begin
  q1  <= in;
  q2  <= q1;
  out <= q2;
end
```

In = 1 out = 0
D1 = 1 Q1 = 0
D2 = 0 Q2 = 0
D3 = 0 Q3 = 0



Пример

NBA регион:

```
always @(posedge clk) begin
  q1  <= in;
  q2  <= q1;
  out <= q2;
end
```

In = 1 out = 0
D1 = 1 Q1 = 1
D2 = 0 Q2 = 0
D3 = 0 Q3 = 0