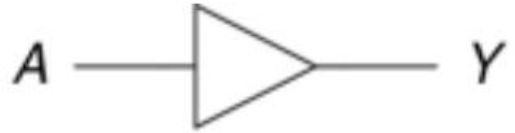
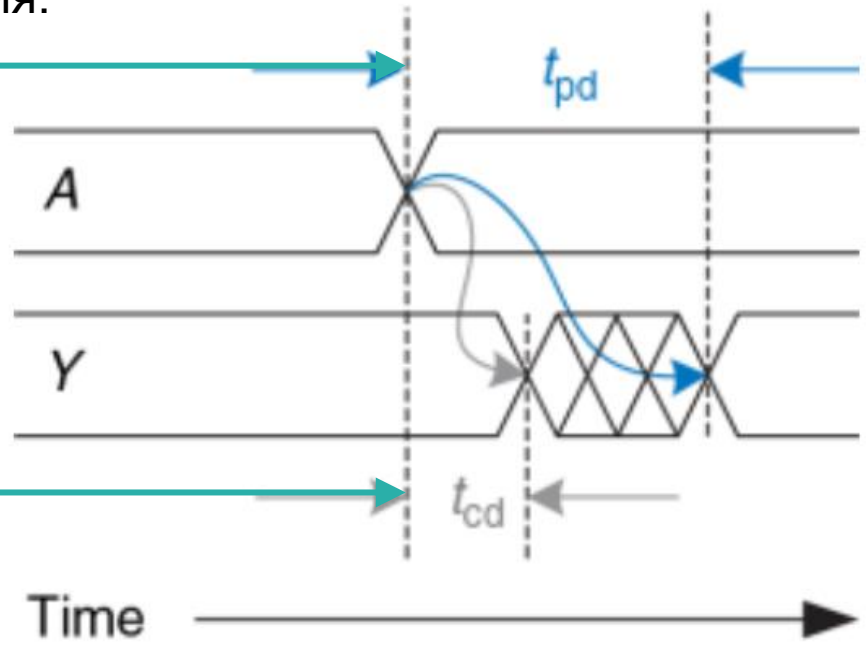
AQUARIUS

Основы логического проектирования микросхем. Последовательностная логика. Verilog.

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»

Синхронизация вычислений

- Использование тактового сигнала позволяет избежать распространение неопределенных результатов в устройстве и синхронизовать отдельные вычисления.

- Какое условие накладывается на период тактового сигнала?

  Период тактового сигнала должен быть больше совокупной задержки

- Из чего складывается совокупная задержка?

  Совокупная задержка:
  1. Время установки выходного сигнала R1
  2. Задержка на комбинационной логике
  3. Время предустановки входного сигнала R2

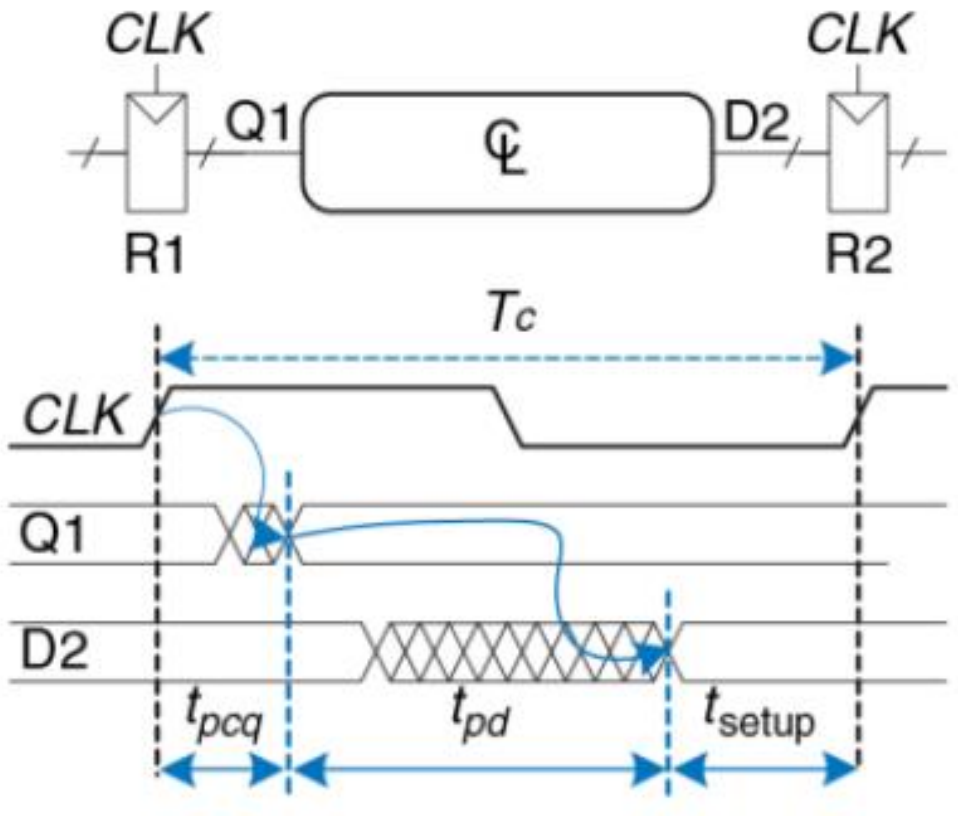
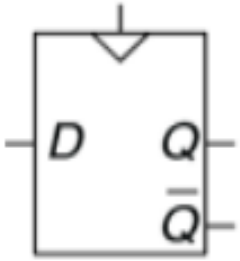


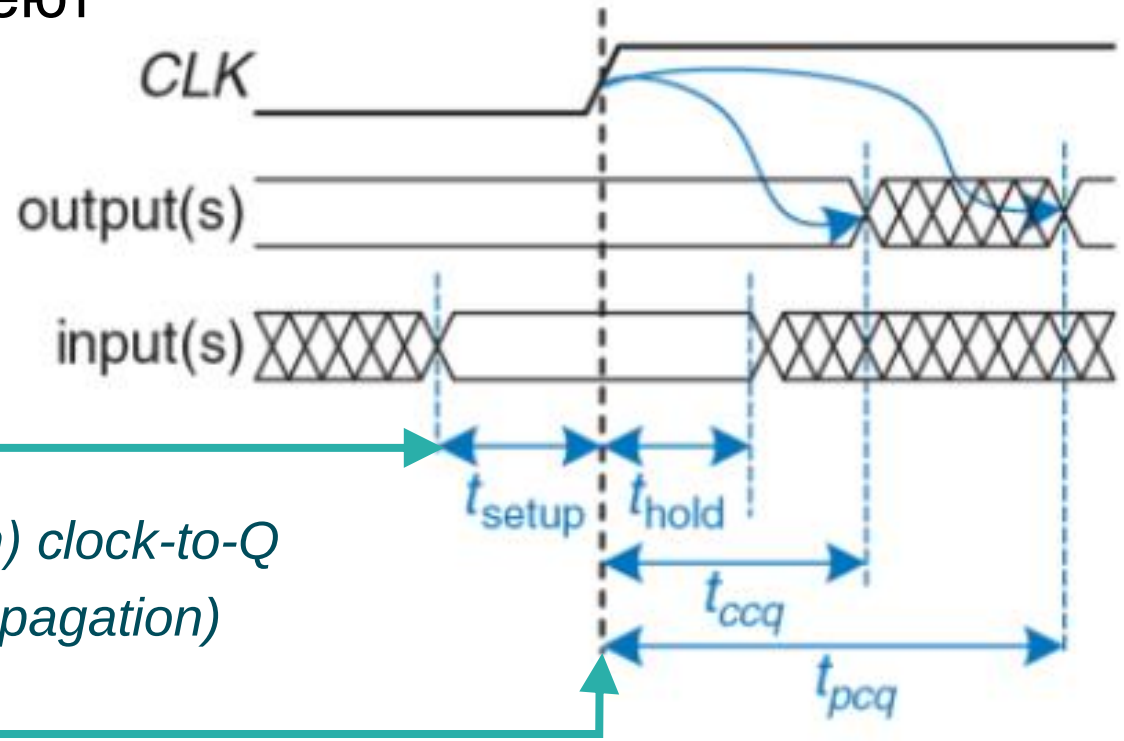
*Иллюстрация из книги Харрис и Харрис «Цифровая схемотехника и архитектура компьютера»

Динамические характеристики



Последовательностные схемы имеют временные ограничения:

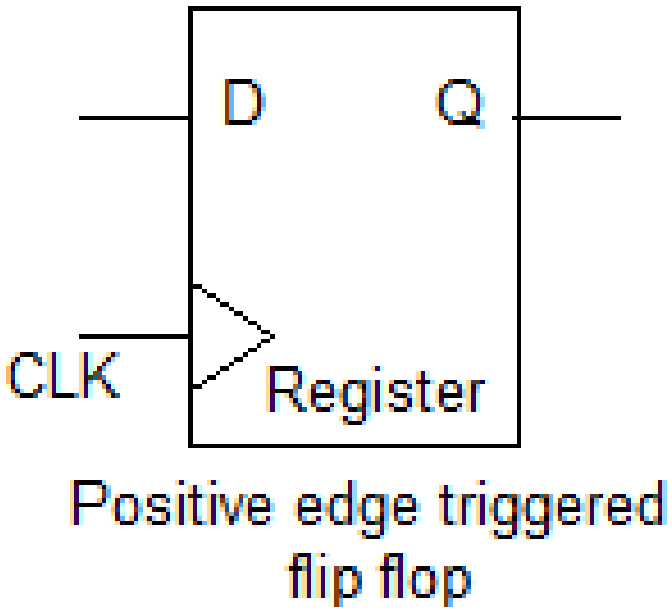
- **t**setup – время предустановки
- **t**hold – время удержания



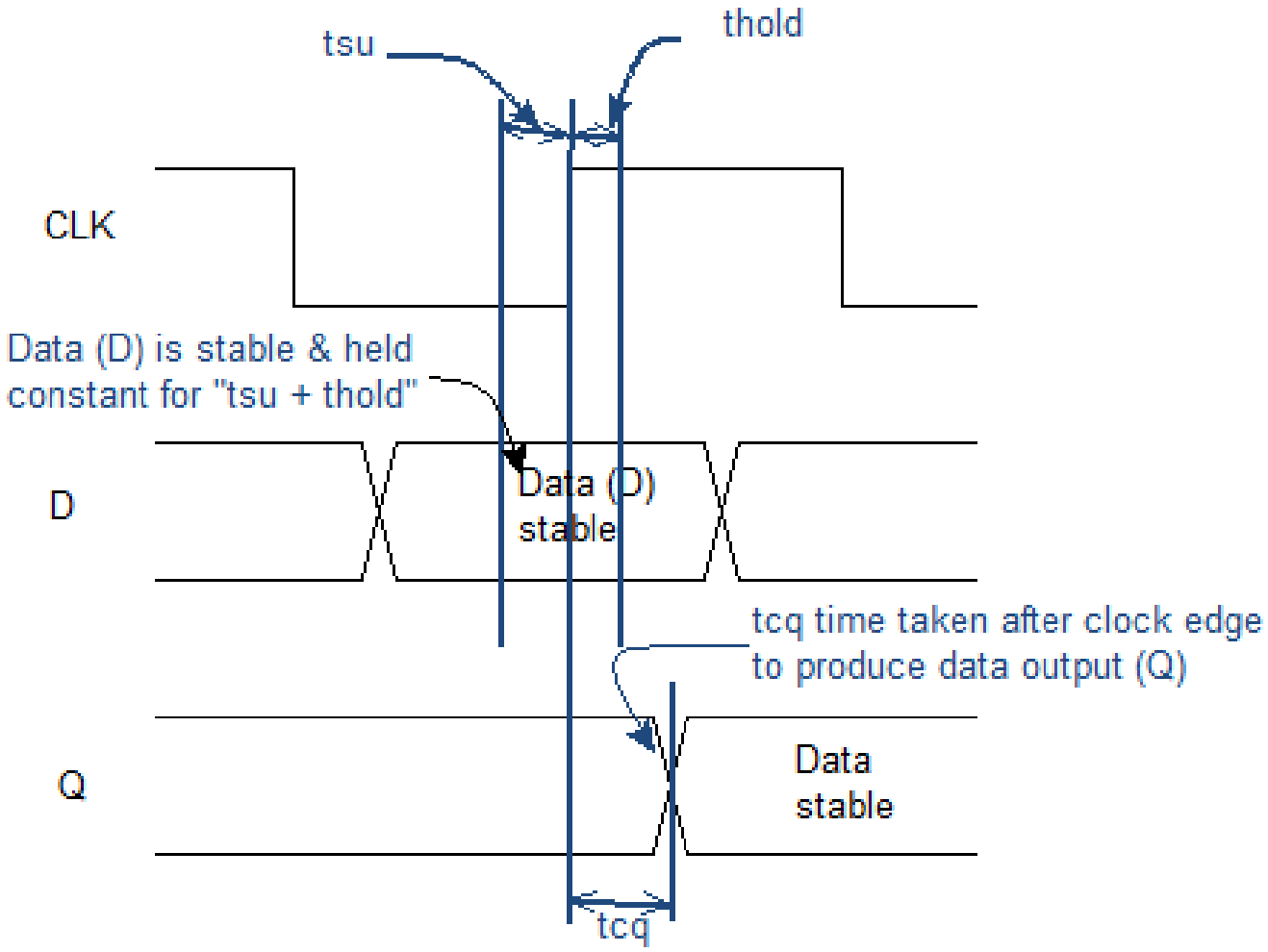
- ***t**ccq – задержка реакции (contamination) clock-to-Q*
- ***t**pcq – задержка распространения (propagation) clock-to-Q*

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

Временная диаграмма D-триггера

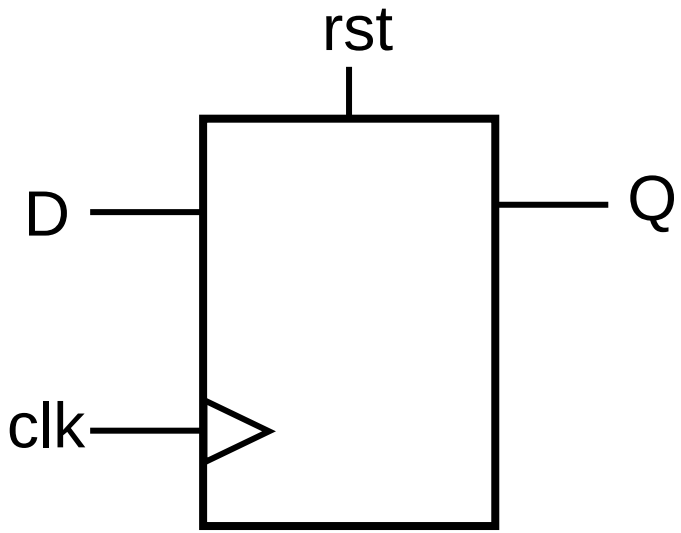


Positive edge triggered
flip flop



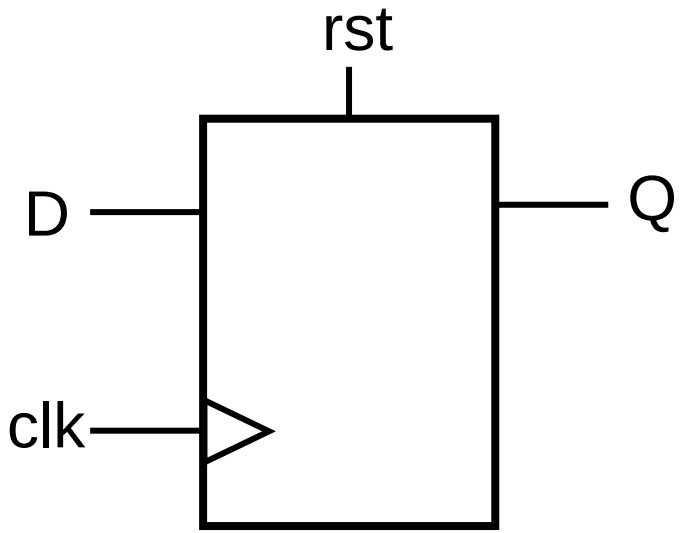
D-триггер с синхронным сбросом.

Для описания триггеров используются неблокирующее присваивание (<=)




```
module flip_flop(
  input clk,
  input rst,
  input D,
  output Q
);

logic out;
  always_ff @(posedge clk) begin
    if (rst) out <=1'b0;
    else out <= D;
  end

  assign Q=out;

endmodule
```

D-триггер с асинхронным сбросом.




```
module flip_flop(
  input clk,
  input rst,
  input D,
  output Q
);

logic out;
  always_ff @(posedge clk or posedge rst) begin
    if (rst) out <=1'b0;
    else out <= D;
  end

  assign Q=out;

endmodule
```

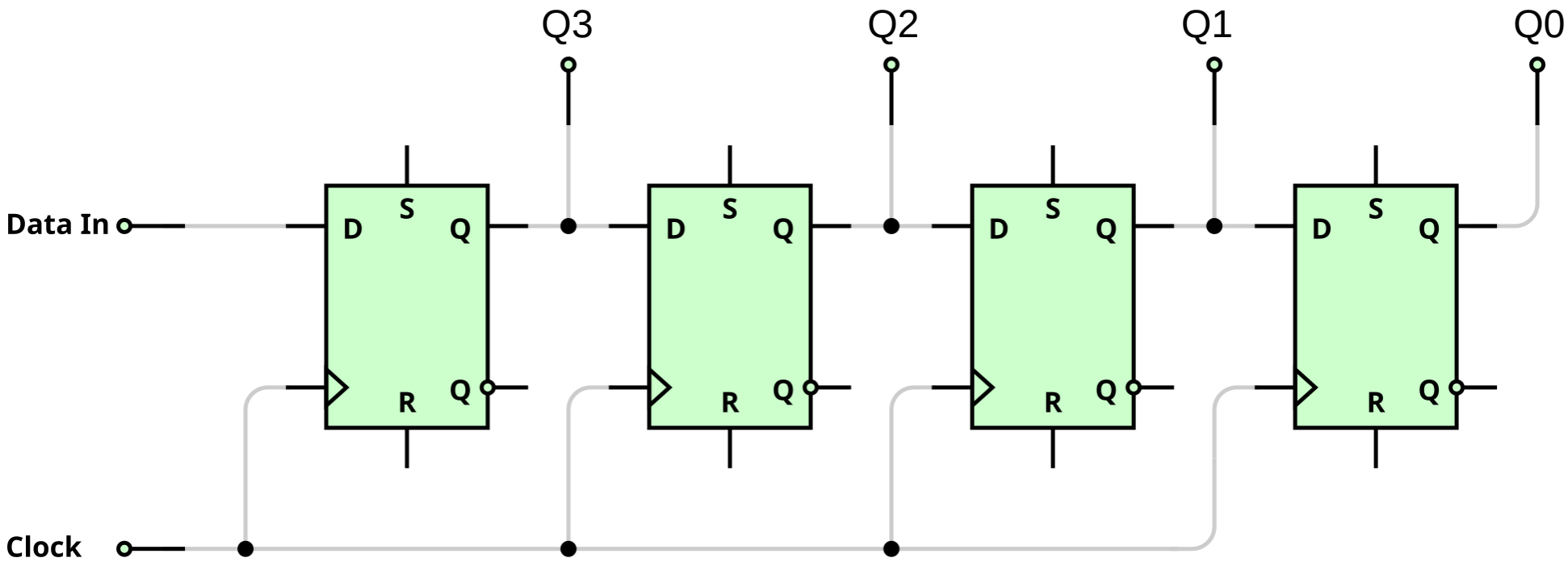
Особенности описания сигнала сброса

- Сигнал сброса должен в своем имени содержать буквенное обозначение ***rst*** или ***reset***
- В разработке ASIC (Application-specific integrated circuit) используется сброс с активным низким уровнем. («Активный ноль»)
- Описание сброса в блоке always **всегда** используется перед описанием логики триггера

Выполнение упражнений hdlbits

Защелки и триггеры

Сдвиговый регистр




```
always @(posedge clk) begin
    q[3:0] <= {data, q[3:1]};
end
```

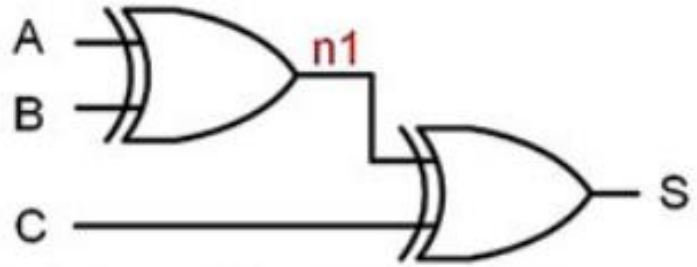
Выполнение упражнений hdlbits

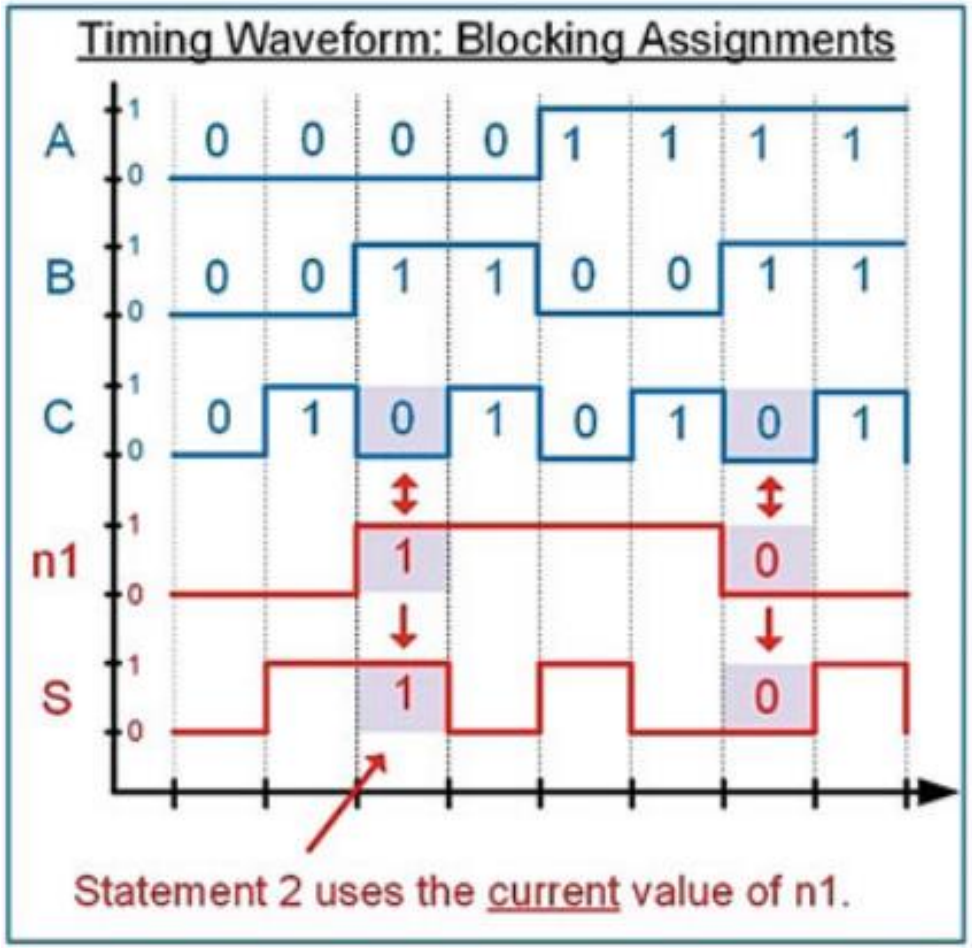
Счетчики и сдвиговые регистры

Неблокирующее vs блокирующее присваивание

Блокирующее присваивание блокирует симуляцию выражения до момента присваивания значения переменной.

```
module BlockingEx1 (output reg  S,
                    input  wire A, B, C);

  reg  n1;

  always @ (A, B, C)
   begin
     n1 = A ^ B;     // statement 1
     S  = n1 ^ C;    // statement 2
   end

endmodule
```

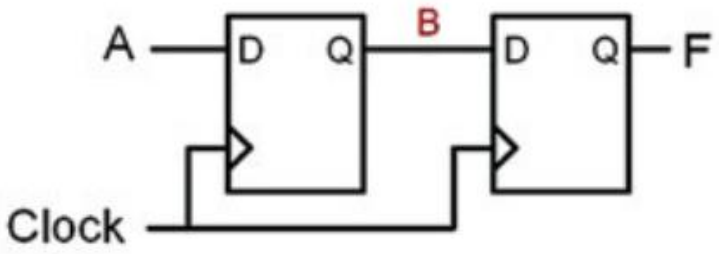


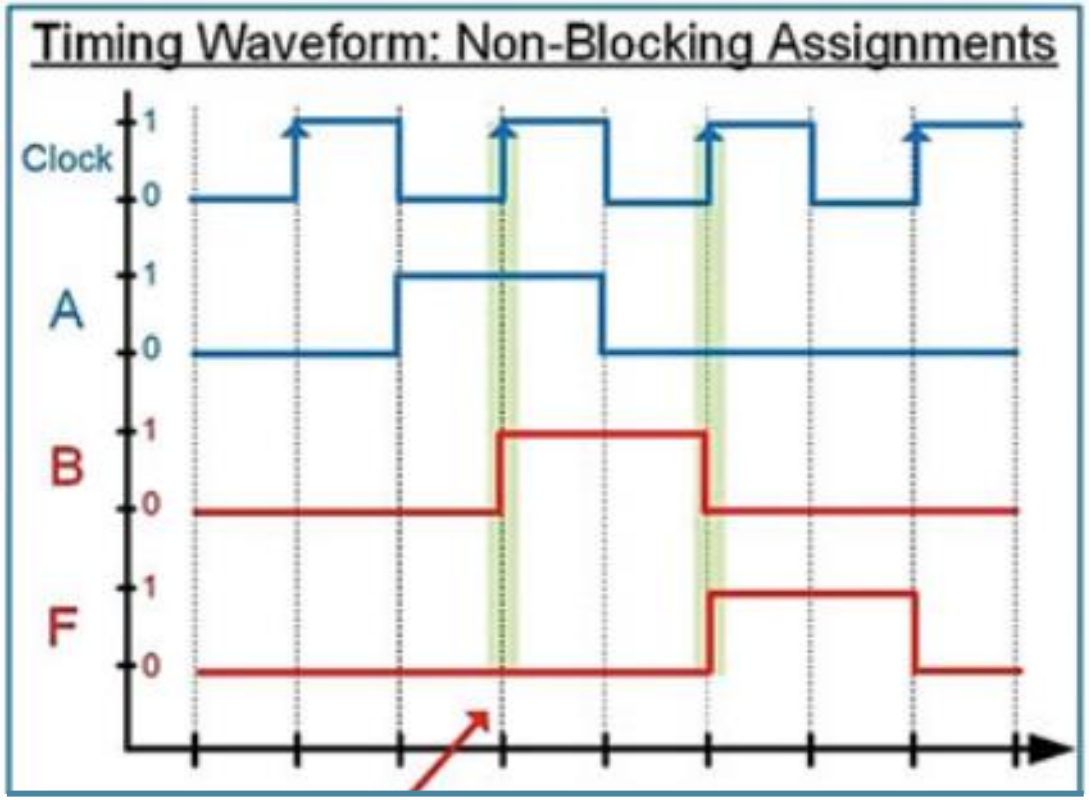


Неблокирующее vs блокирующее присваивание

Неблокирующее присваивание не блокирует присвоение значения и выполняет следующие процедуры блока always. То есть присвоения симулируются параллельно.

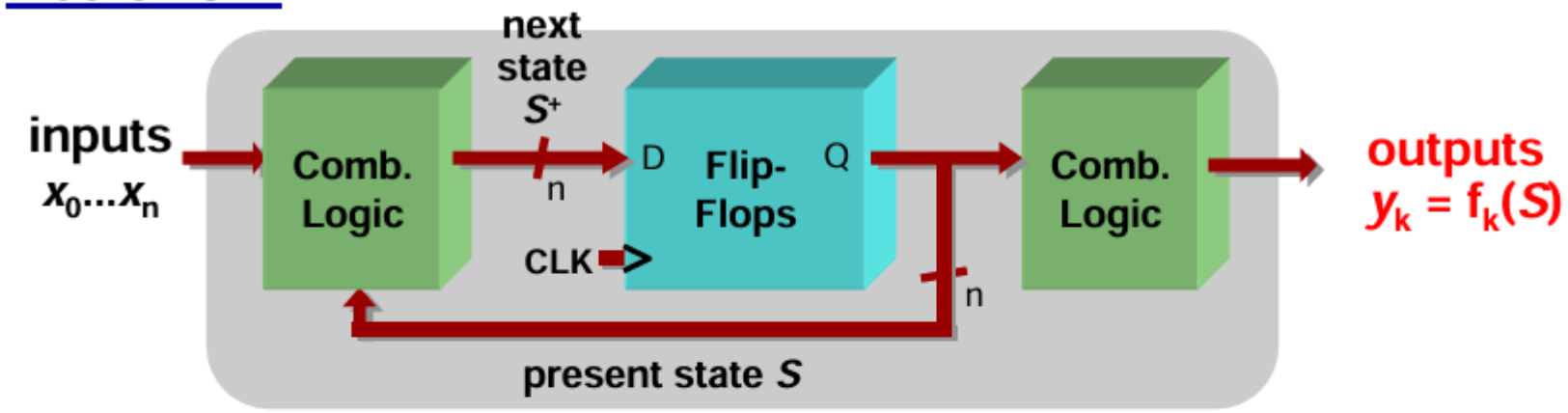
```
module NonBlockingEx1 (output reg  F,
                       input  wire A,
                       input  wire Clock);
  reg  B;

  always @ (posedge Clock)
    begin
      B <= A;    // statement 1
      F <= B;    // statement 2
    end

endmodule
```



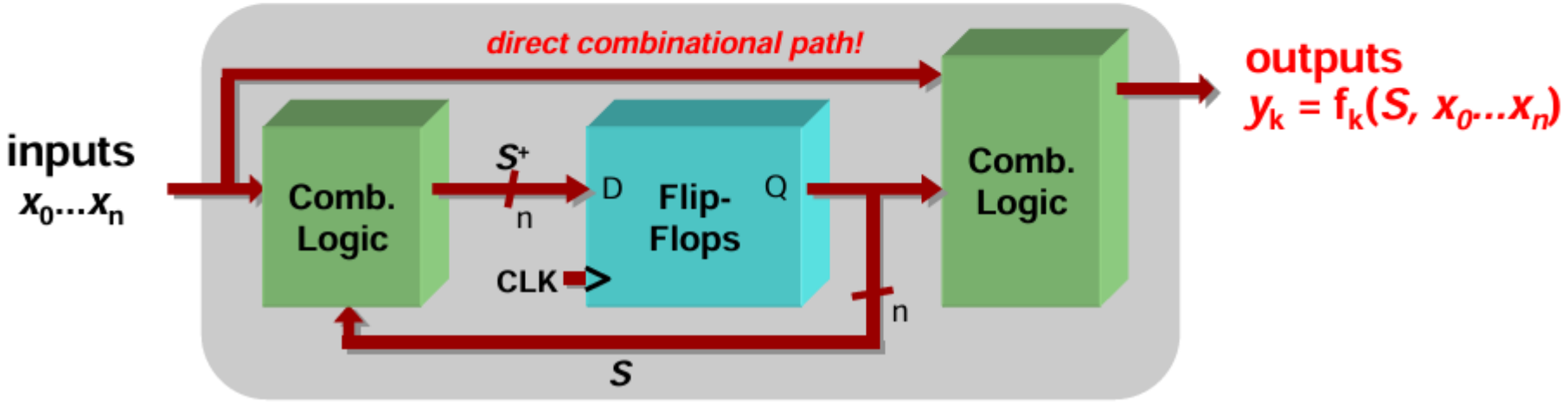


Конечные автоматы

Moore FSM:

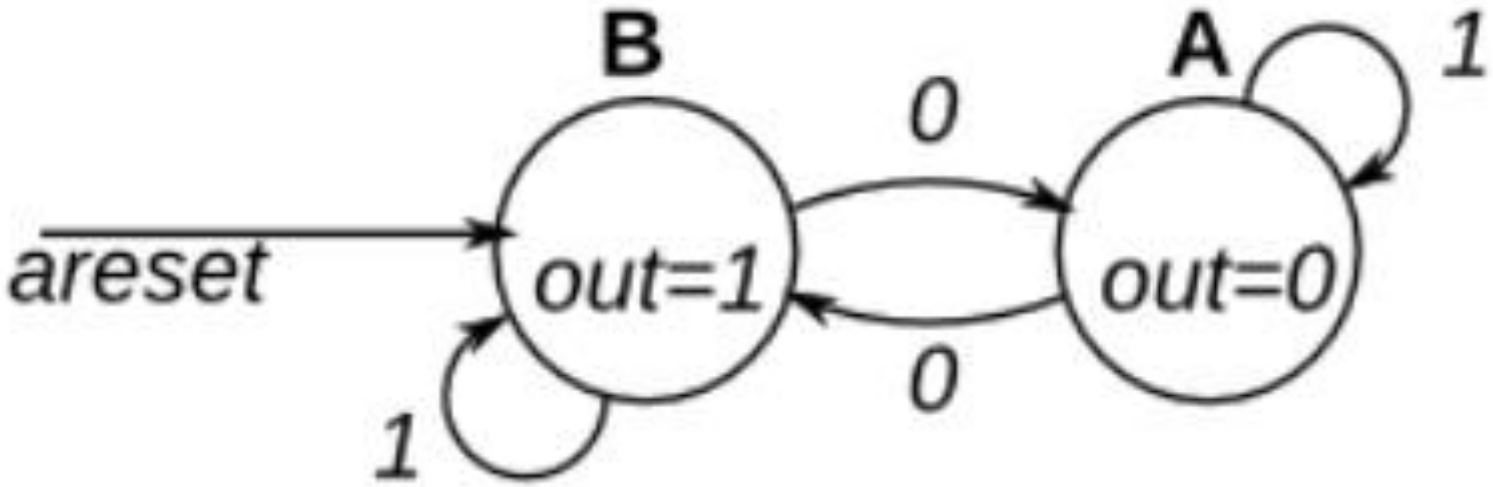


Mealy FSM:



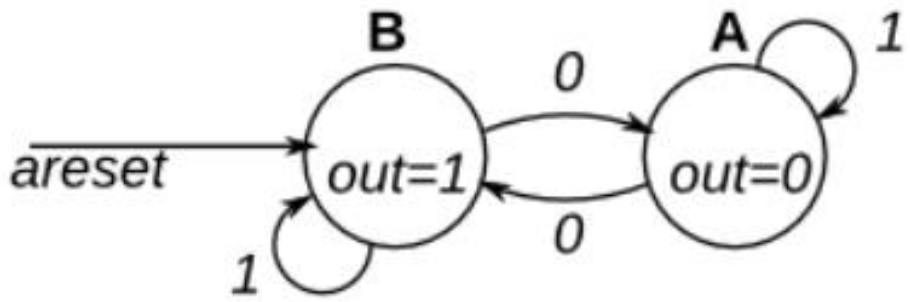
(c) Introductory Digital Systems Laboratory

Конечные автоматы



Пример конечного автомата

```
always @(*) begin
    case (state)
        A:
        begin
            out = 0;
            if (in == 1) next_state = A;
            else next_state = B;
        end
        B:
        begin
            out = 1;
            if (in == 1) next_state = B;
            else next_state = A;
        end
    endcase
end
```




```
always @(posedge clk, posedge areset) begin
    if (areset) begin
        state <= B;
    end
    else begin
        state <= next_state;
    end
end
```

Выполнение упражнений hdlbits

Конечные автоматы (раздел FSM)