

# Триггеры. Регистры в Verilog. Счётчики

# Цифровые схемы

$$F: \{0, 1\}^N \rightarrow \{0, 1\}^M$$



Комбинационная  
ЛОГИКА

$$\begin{pmatrix} \text{out}^1(t_r) \\ \text{out}^2(t_r) \\ \text{out}^3(t_r) \\ \dots \\ \text{out}^m(t_r) \end{pmatrix} = F \begin{pmatrix} \text{in}^1(t_r) \\ \text{in}^2(t_r) \\ \text{in}^3(t_r) \\ \dots \\ \text{in}^n(t_r) \end{pmatrix}$$

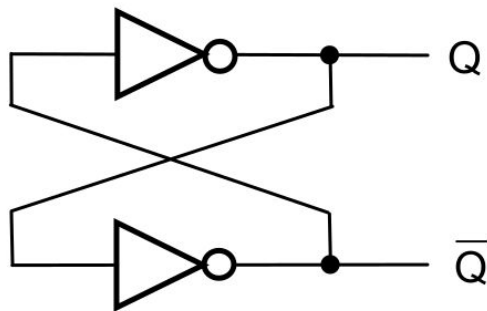


Последовательная  
ЛОГИКА

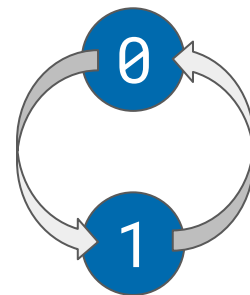
$$\begin{pmatrix} \text{out}^1(t_r) \\ \text{out}^2(t_r) \\ \text{out}^3(t_r) \\ \dots \\ \text{out}^m(t_r) \end{pmatrix} = F \begin{pmatrix} \text{in}^1(t_r) \\ \dots \\ \text{in}^n(t_r) \\ \text{in}^1(t_{r-1}) \\ \dots \\ \text{in}^k(t_{r-1}) \end{pmatrix}$$

# Последовательная логика

- Последовательностные схемы обладают памятью, в отличие от комбинационных
- Предыдущие значения входов определяют *состояние* системы

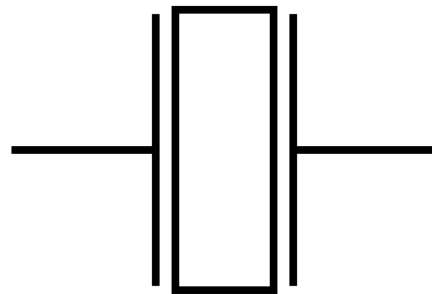


Бистабильный элемент — элемент с двумя устойчивыми состояниями. Способен хранить 1 бит информации.



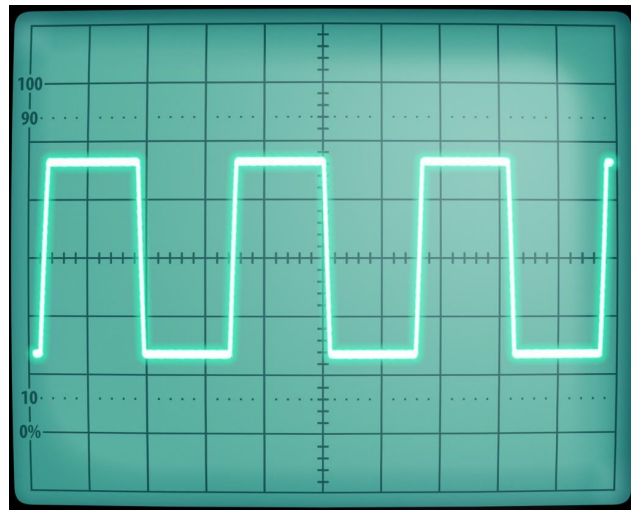
# Тактирование

- Комбинационная логика не работает мгновенно, поэтому вычисления нужно *синхронизировать*
- Кварцевый генератор генерирует *такты́ый сигнал* — clock
- Все комбинационные схемы должны успеть закончить вычисления за период тактового сигнала



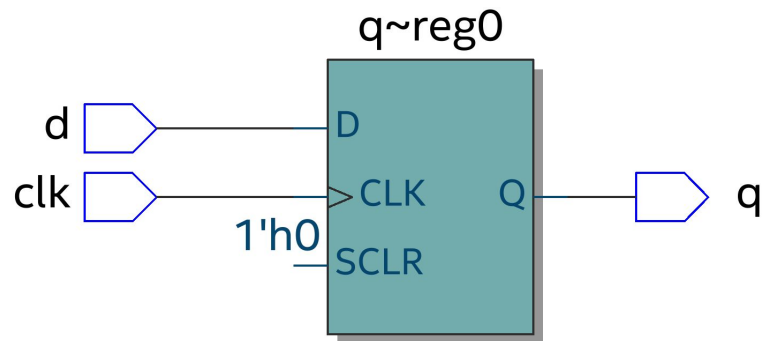
# Тактирование



- Тактовый сигнал — clock
- Фронт — positive edge
- Спад — negative edge



# D-триггер

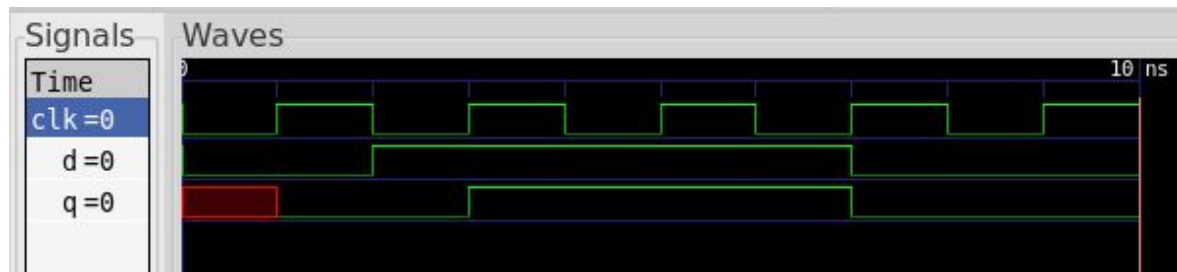
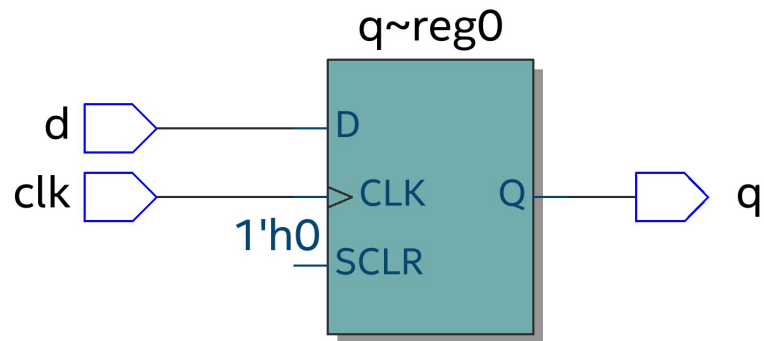
```
reg q;  
  
always @(posedge clk) begin  
    q <= d; // Non-blocking assignment  
end
```



D	CLK	$Q_n$
0		0
1		1
*	0	$Q_{n-1}$
*	1	$Q_{n-1}$

# D-триггер

```
reg q;  
  
always @(posedge clk) begin  
    q <= d;  
end
```

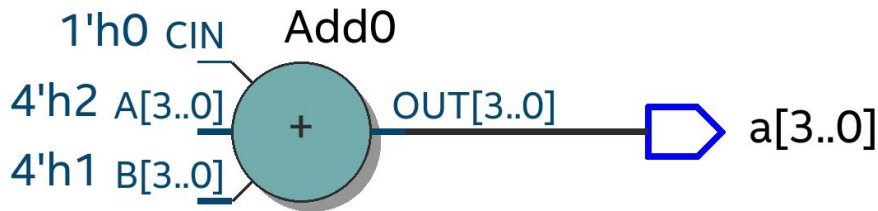


D	CLK	$Q_n$
0		0
1		1
*	0	$Q_{n-1}$
*	1	$Q_{n-1}$

# Присваивание

- Непрерывное — continuous assignment
  - Используется для описания простой комбинационной логики
  - Вместе с оператором `assign` или при определении `wire`
  - Слева — `wire`, справа — любые выходы/операции

```
wire [3:0]a;  
wire [3:0]b = 4'b10;  
  
assign a = b + 4'b1;
```



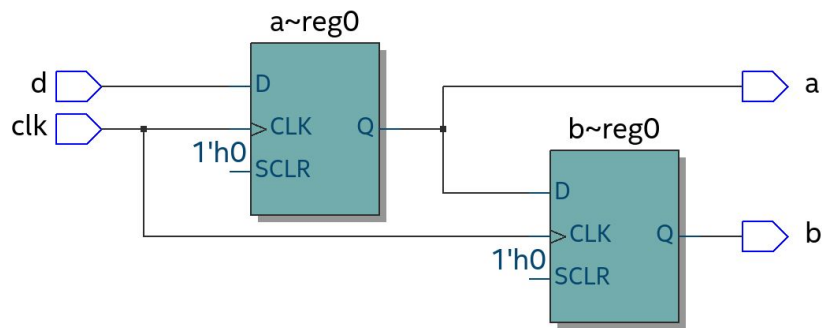
- Неблокирующее — nonblocking assignment
- Блокирующее — blocking assignment



# Присваивание

- Непрерывное — continuous assignment
- Неблокирующее — nonblocking assignment
  - Используется для описания последовательностной логики
  - Только внутри `always`-блока, порядок присваиваний не важен
  - Слева — `reg`, справа — любые выходы/операции

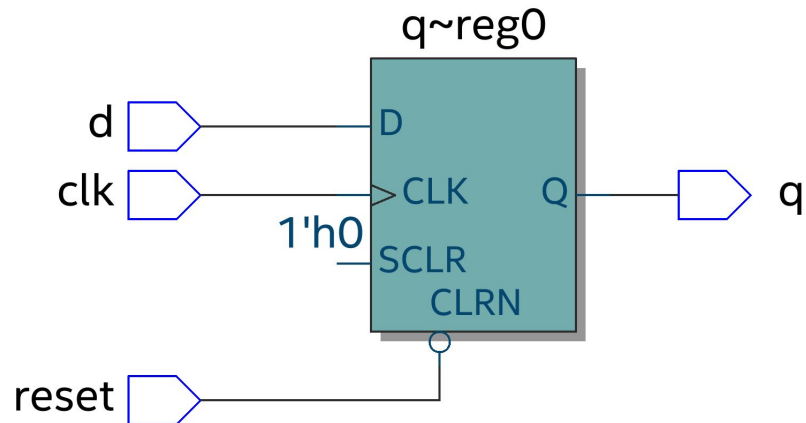
```
reg a;  
reg b;  
  
always @(posedge clk) begin  
    a <= d;  
    b <= a;  
end
```



- Блокирующее — blocking assignment

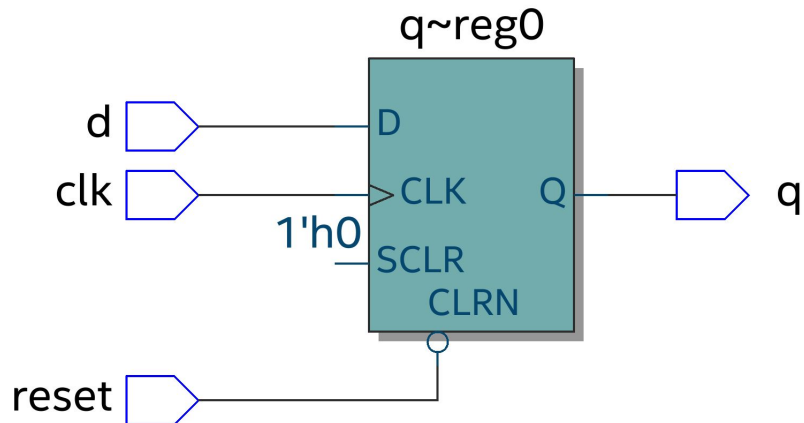
# D-триггер с асинхронным сбросом

```
reg q;  
  
always @(posedge clk or negedge reset) begin  
    if (!reset)  
        q <= 0;  
    else  
        q <= d;  
end
```



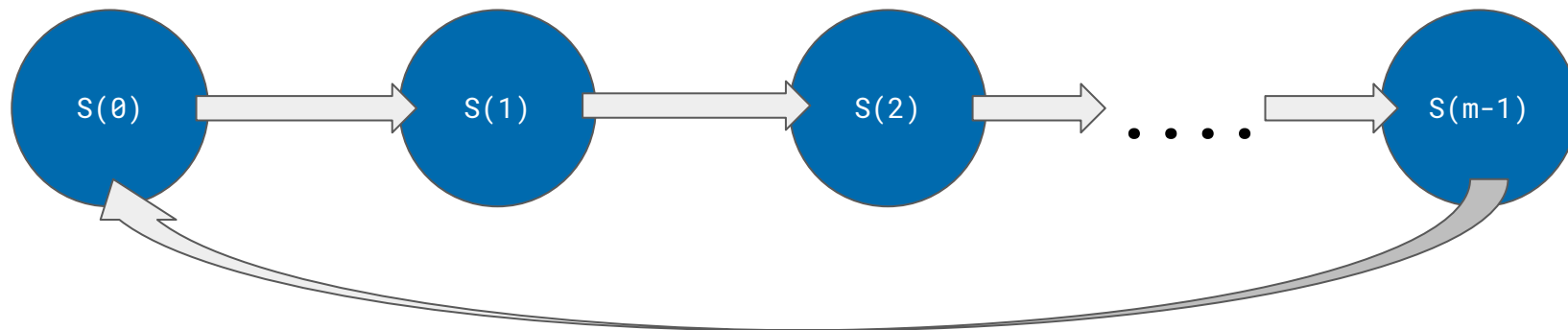
# D-триггер с асинхронным сбросом

```
reg q;  
  
always @(posedge clk or negedge reset) begin  
    if (!reset)  
        q <= 0;  
    else  
        q <= d;  
end
```



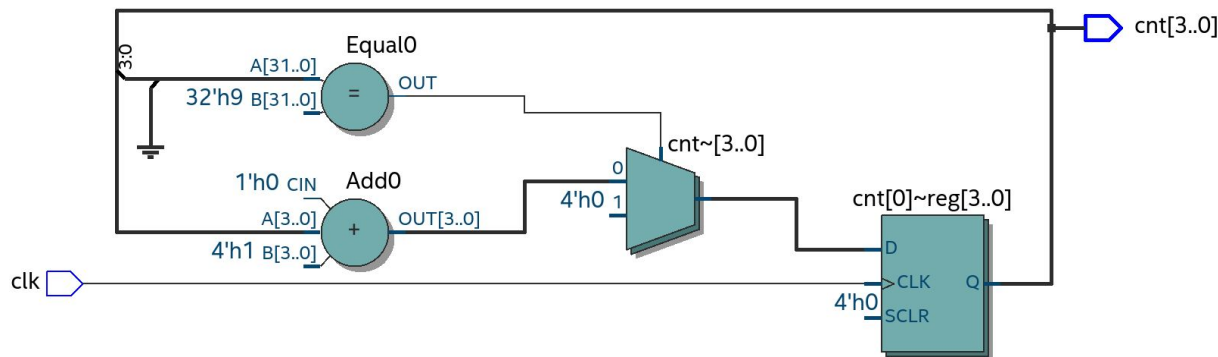
# Счетчик

- Тактируемая последовательностная схема
- Диаграмма состояний — единственное кольцо
- Счетчик от 0 до  $m-1$  имеет  $m$  состояний
- Счетчик от 0 до  $2^n-1$  требует  $n$  бит и называется двоичным  $n$ -разрядным счетчиком
- Может использоваться как делитель частоты на  $m$



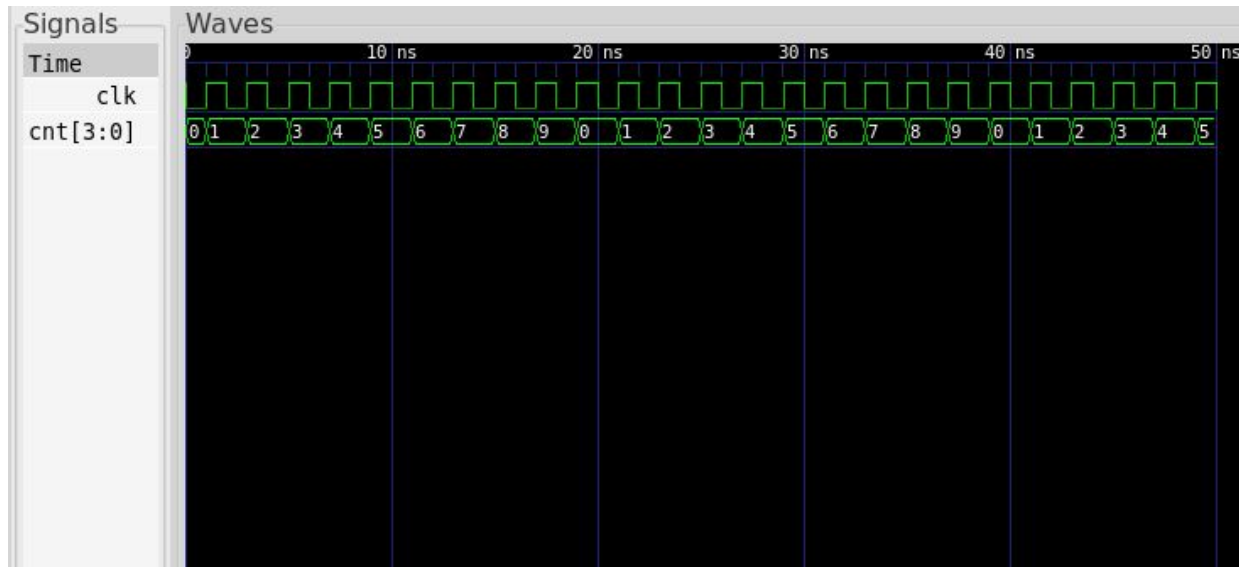
# Счетчик

```
module counter(  
    input clk  
);  
  
reg [3:0]cnt = 0;  
  
always @(posedge clk) begin  
    if (cnt == 9)  
        cnt <= 0;  
    else  
        cnt <= cnt + 1;  
    end  
  
endmodule
```



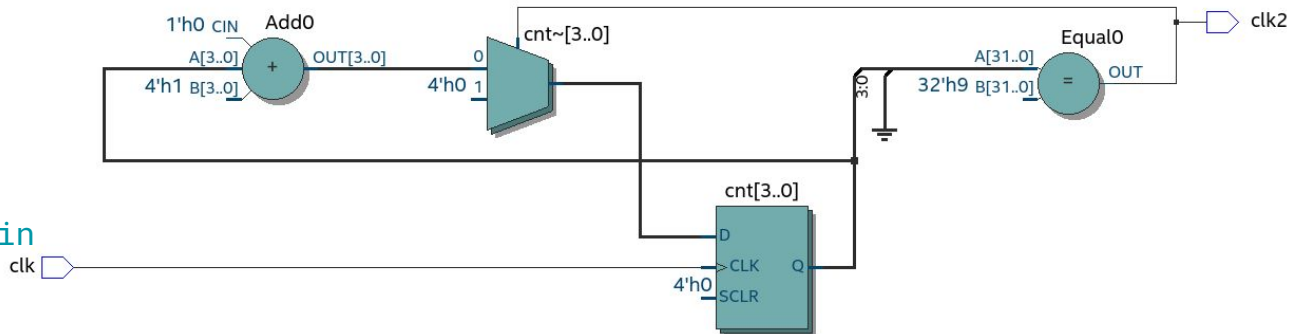
# Счетчик

```
module counter(  
    input clk  
);  
  
    reg [3:0]cnt = 0;  
  
    always @(posedge clk) begin  
        if (cnt == 9)  
            cnt <= 0;  
        else  
            cnt <= cnt + 1;  
    end  
  
endmodule
```



# Счетчик

```
module counter(  
    input clk,  
  
    output clk2  
);  
  
assign clk2 = (cnt == 9);  
  
reg [3:0]cnt = 0;  
  
always @(posedge clk) begin  
    if (clk2)  
        cnt <= 0;  
    else  
        cnt <= cnt + 1;  
    end  
  
endmodule
```



# Счетчик

```
module counter(  
    input clk,  
  
    output clk2  
);  
  
assign clk2 = (cnt == 9);  
  
reg [3:0]cnt = 0;  
  
always @(posedge clk) begin  
    if (clk2)  
        cnt <= 0;  
    else  
        cnt <= cnt + 1;  
    end  
  
endmodule
```





# GitHub

[github.com/viktor-prutyanov/drec-fpga-intro](https://github.com/viktor-prutyanov/drec-fpga-intro)