

ОСНОВЫ

VerilogHDL/SystemVerilog

(синтез и моделирование)

Два типа RTL процессов

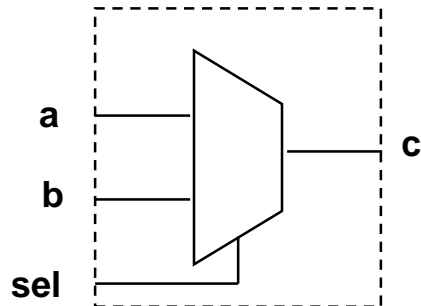
- **Комбинационный процесс**

- Чувствителен ко всем сигналам в процессе

```
always @ (a, b, sel)  
always @ *
```

Список чувствительности включает все входы комбинационной цепи

* - добавить все входы

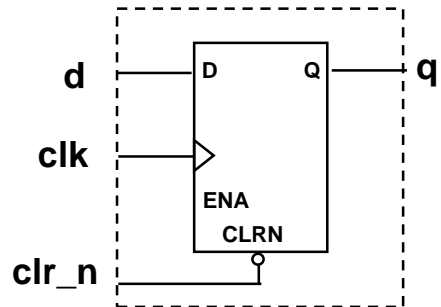


- **Тактовый (регистровый) процесс**

- Чувствителен к тактовым сигналам и сигналам управления

```
always @ (posedge clk, negedge clr_n)
```

Список чувствительности не включает d вход, а только тактовый сигнал и сигнал асинхронного сброса



Edge-Triggered Flipflop

```
module dff (  
    input d, clk,  
    output reg q);  
    always @(posedge clk)  
        q <= d ;  
  
endmodule
```

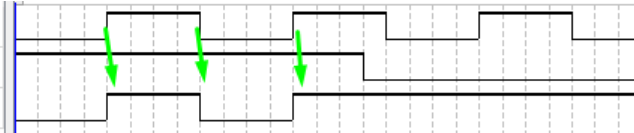
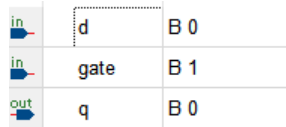


Latch и Flipflop

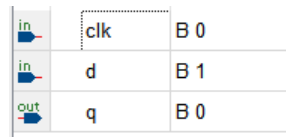
Level-Sensitive Latch

```
module latch (  
    input d, gate,  
    output reg q);  
    always @(d, gate)  
        if (gate)  
            q = d ;  
endmodule
```

Level-Sensitive Latch



Edge-Triggered Flipflop

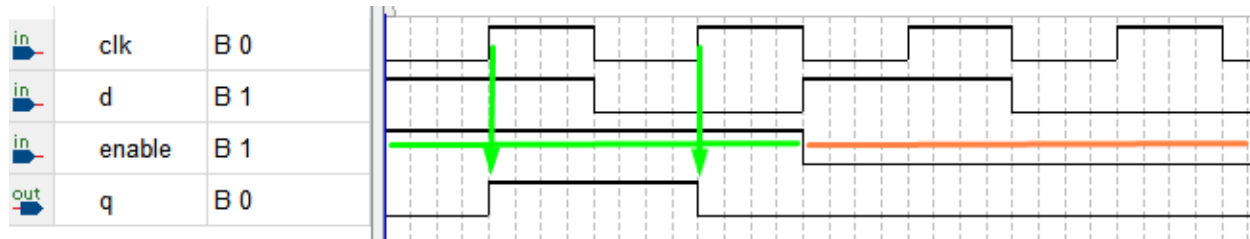
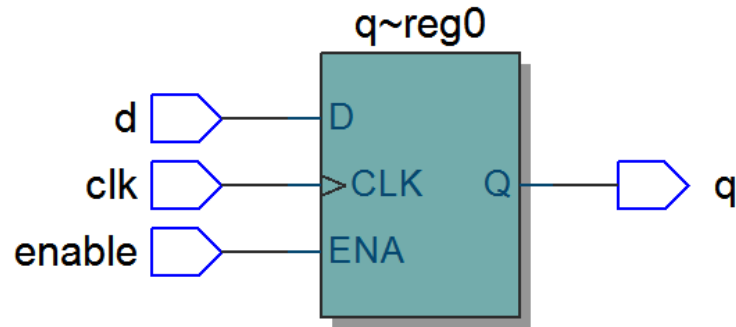


Edge-Triggered Flipflop

```
module dff (  
    input d, clk,  
    output reg q);  
    always @(posedge clk)  
        q <= d ;  
endmodule
```

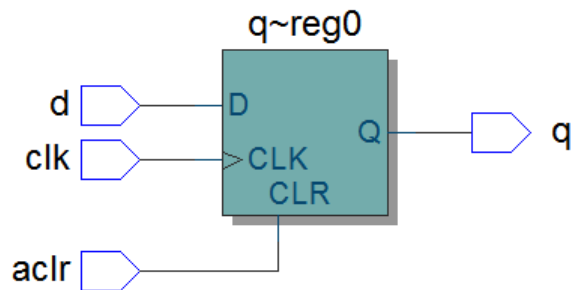
Clock Enable

```
module dff_ena (  
    input d, enable, clk,  
    output reg q);  
  
    always @( posedge clk )  
        if (enable)  
            q <= d;  
  
endmodule
```

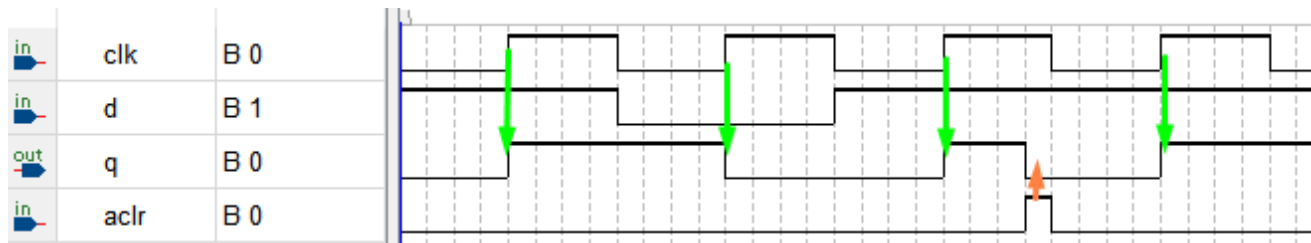


Asynchronous Clear

```
module dff_async (  
    input d, clk, aclr,  
    output reg q );  
  
    always @(posedge clk, posedge aclr)  
        if (aclr)  
            q <= 1'b0;  
        else  
            q <= d;  
  
endmodule
```

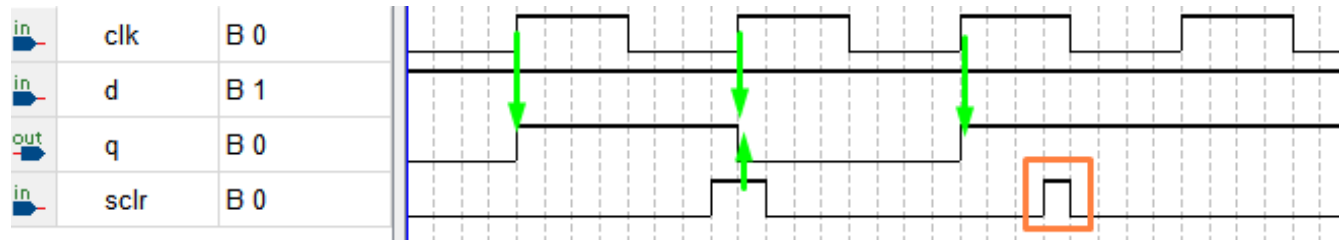
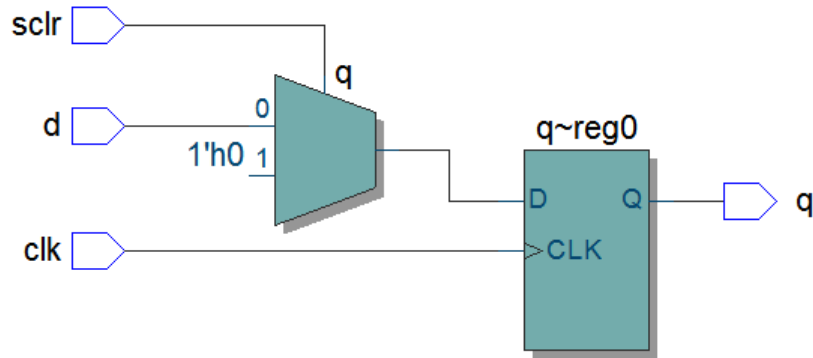


Using *posedge aclr*, must check that *aclr* equals '1'



Synchronous Clear

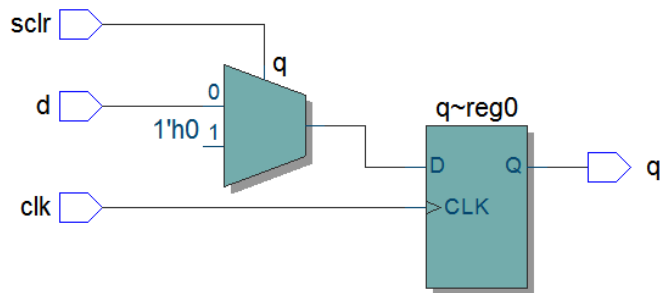
```
module dff_sync (  
    input d, clk, sclr,  
    output reg q);  
  
    always @(posedge clk)  
        if (sclr)  
            q <= 1'b0;  
        else  
            q <= d;  
  
endmodule
```



Asynchronous и Synchronous Clear

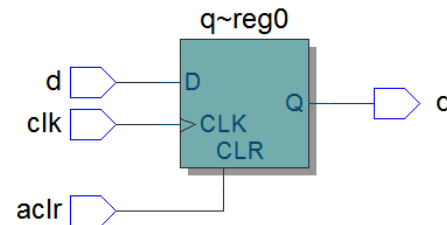
Synchronous Preset & Clear

```
module dff_sync (  
    input d, clk, sclr,  
    output reg q );  
always @(posedge clk) begin  
    if (sclr) q <= 1'b0;  
    else  
        q <= d;  
    end  
endmodule
```



Asynchronous Clear

```
module dff_async (  
    input d, clk, aclr,  
    output reg q);  
always @(posedge clk, posedge  
aclr)  
    if (aclr) q <= 1'b0;  
    else  
        q <= d;  
    end  
endmodule
```



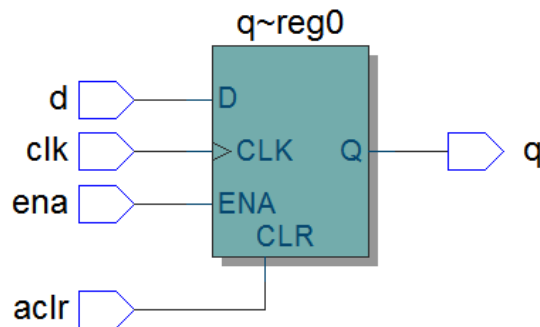
Приоритет управляющих сигналов

- ❑ Асинхронный сброс - Asynchronous clear (aclr)
- ❑ Асинхронная установка - Asynchronous preset (pre)
- ❑ Асинхронная загрузка - Asynchronous load (aload)
- ❑ Разрешение работы - Enable (ena)
- ❑ Синхронный сброс - Synchronous clear (sclr)
- ❑ Синхронная загрузка - Synchronous load (sload)

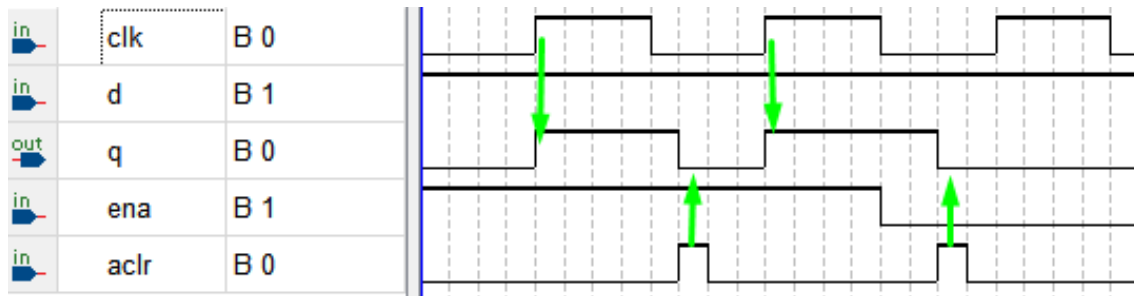
- ❑ Примечание:
 - ✓ Изменение порядка применения управляющих сигналов приводит к использованию дополнительных логических элементов

Asynchronous Clear и Enable

```
module dff_async (  
    input d, clk, aclr, ena,  
    output reg q );  
  
    always @(posedge clk, posedge aclr)  
        if (aclr)  
            q <= 1'b0;  
        else if (ena)  
            q <= d;  
endmodule
```

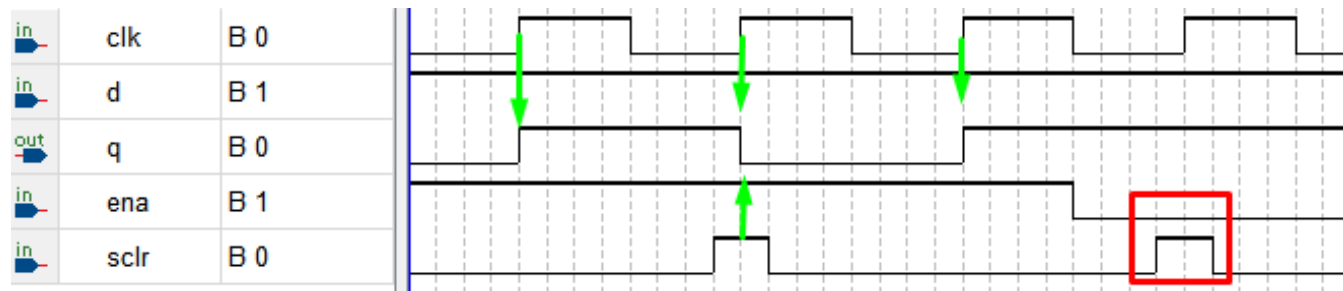
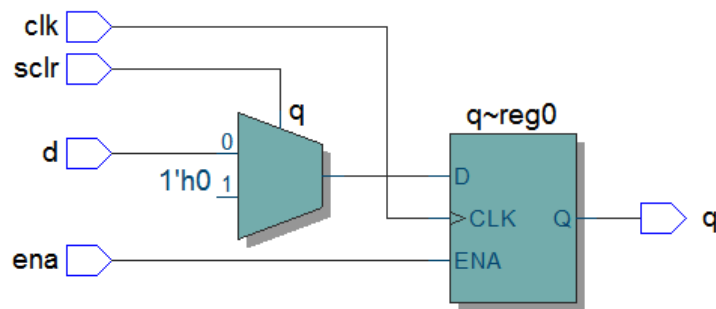


Using *posedge aclr*, must check that `aclr` equals '1'



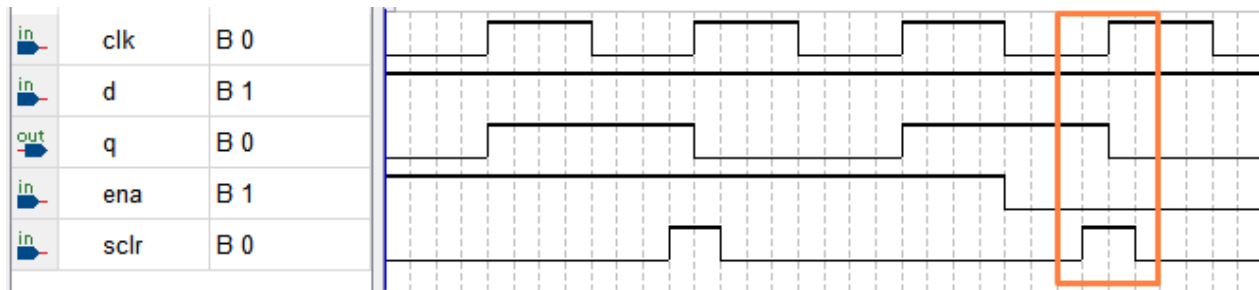
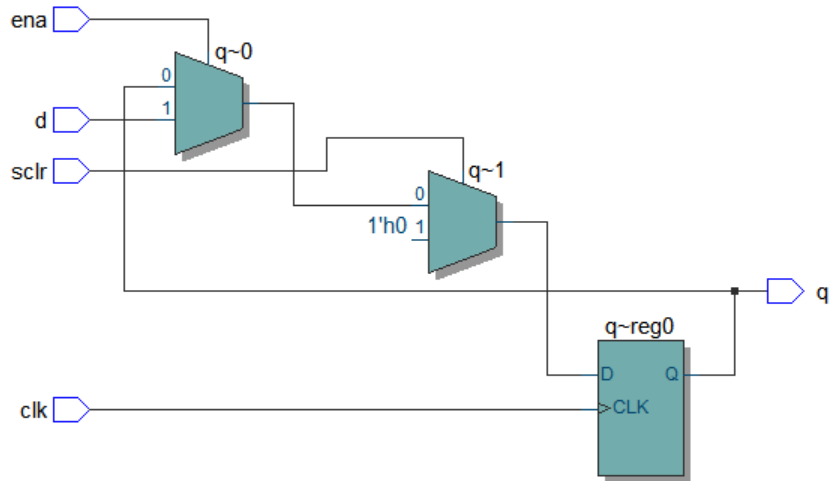
Synchronous Clear и Enable

```
module dff_sync (  
    input d, clk, sclr, ena,  
    output reg q);  
    always @(posedge clk)  
        if (ena)  
            if (sclr)  
                q <= 1'b0;  
            else  
                q <= d;  
endmodule
```



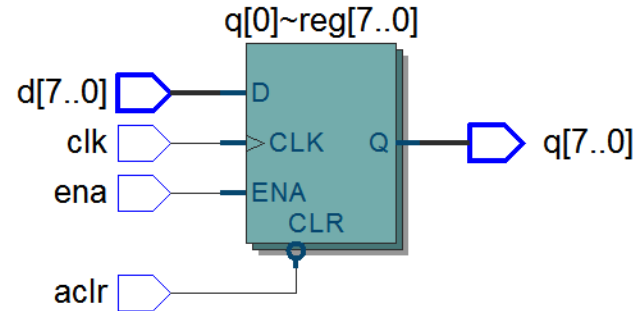
Synchronous Clear и Enable (неправильный приоритет)

```
module ex_1 (  
    input d, clk, sclr, ena,  
    output reg q);  
    always @(posedge clk)  
        if (sclr)  
            q <= 1'b0;  
        else if (ena)  
            q <= d;  
endmodule
```

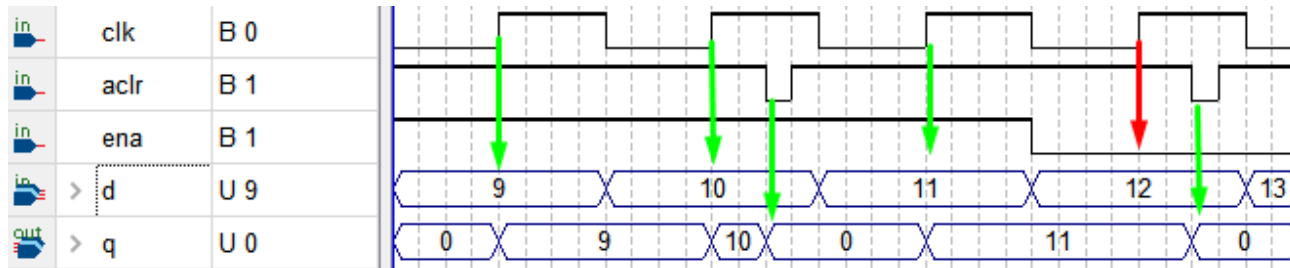


Регистр

```
module ex_1 (  
    input [7:0] d,  
    input clk, aclr, ena,  
    output reg [7:0] q;  
    always @(posedge clk, negedge aclr)  
        if (!aclr)  
            q <= 8'd0;  
        else if (ena)  
            q <= d;  
endmodule
```



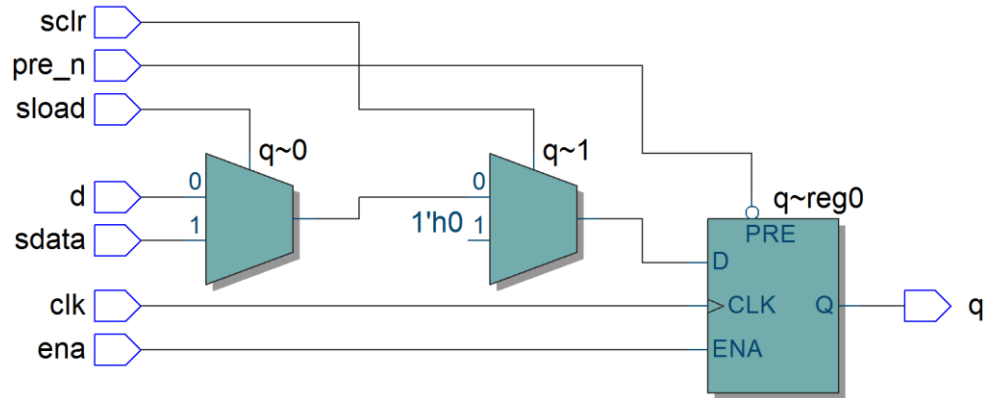
Using *negedge aclr*, must check that *aclr* equals '0'



Пример с другими управляющими сигналами

```
module ex_1 (  
    input clk, ena, d, sclr,  
    input sload, sdata, pre_n,  
    output reg q );
```

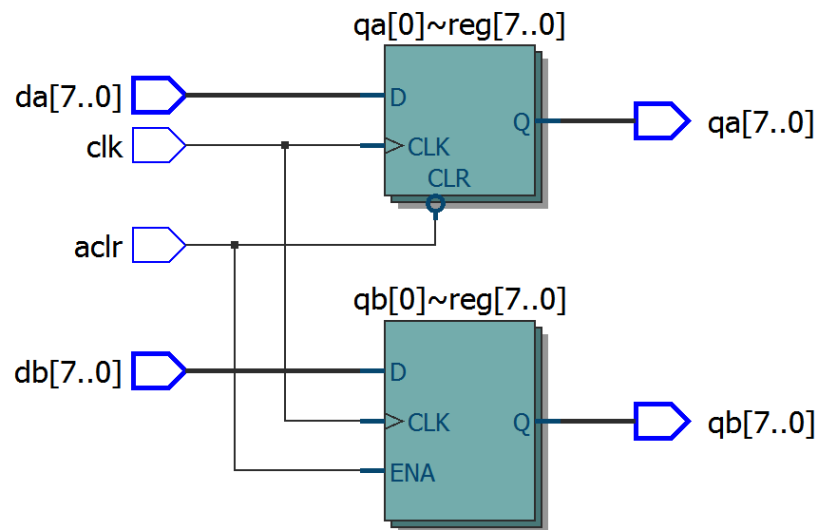
```
always @(posedge clk, negedge pre_n)  
    if (!pre_n)  
        q <= 1'b1;  
    else if (ena)  
        if (sclr)  
            q <= 1'b0;  
        else if (sload)  
            q <= sdata;  
        else  
            q <= d;  
endmodule
```



Правильно?

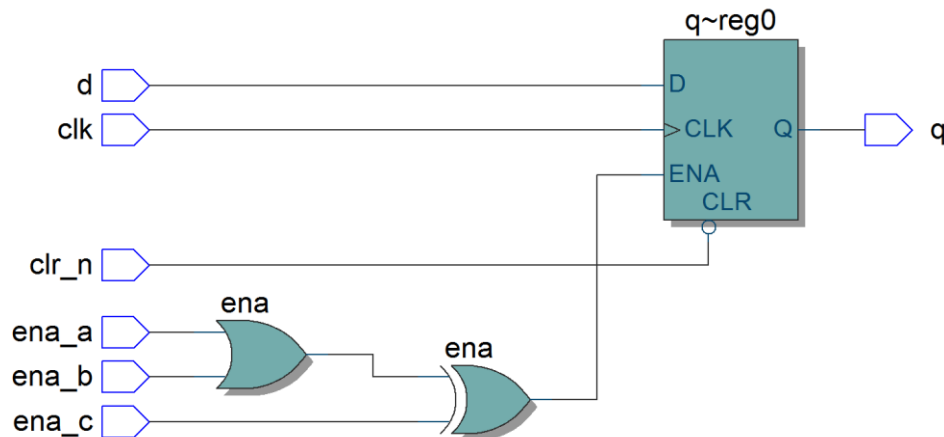
- ❑ Если qa надо сбрасывать, а qb – нет.

```
module rg_ex6 (da, db, aclr, clk, qa, qb);  
  
    input [7:0] da, db;  
    input aclr, clk;  
    output reg [7:0] qa, qb ;  
  
    always @( posedge clk or negedge aclr)  
        if (aclr == 0) qa <=0;  
        else  
            begin  
                qa <= da;  
                qb <= db;  
            end  
    endmodule
```



Триггер и сложный сигнал разрешения работы

```
module dff_ena (  
    input clk, clr_n, d, ena_a, ena_b, ena_c,  
    output reg q);  
always @ (posedge clk, negedge clr_n)  
    if (clr_n == 1'b0)  
        q <= 1'b0;  
    else if (ena == 1'b1)  
        q <= d;  
  
    assign ena = (ena_a | ena_b) ^ ena_c;  
  
endmodule
```



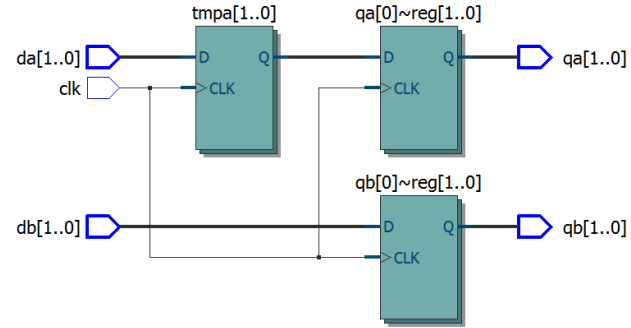
Блокирующие и неблокирующие назначения

```
module rg_ex6_1 (da, db, clk, qa, qb);
input [1:0] da, db;
input clk;
output reg [1:0] qa, qb ;

reg [1:0] tmpa, tmpb;

always @( posedge clk )
begin
    tmpa <= da;
    qa <= tmpa;

    tmpb = db;
    qb = tmpb;
end
endmodule
```

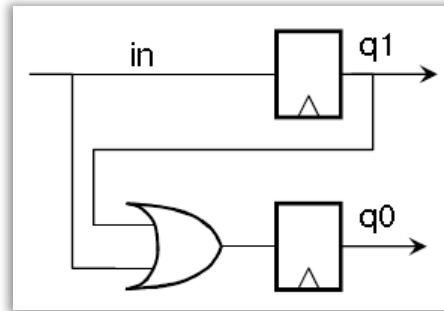
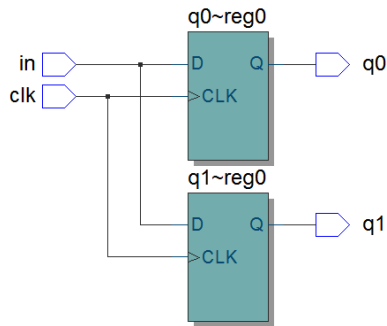


**При описании триггерных схем следует использовать
неблокирующие назначения**

Блокирующие и неблокирующие назначения

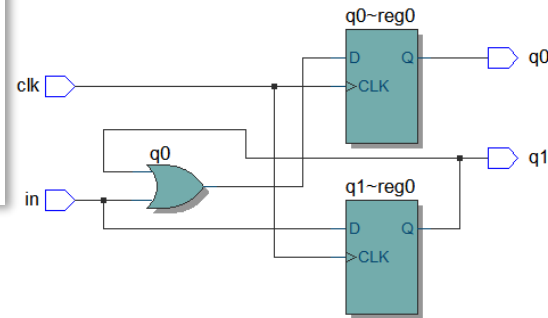
Blocking

```
module ex_1 (  
  input  clk, in,  
  output reg q1, q0);  
  
  always @(posedge clk) begin  
    q1 = in;  
    q0 = in | q1; end  
endmodule
```



Non-blocking

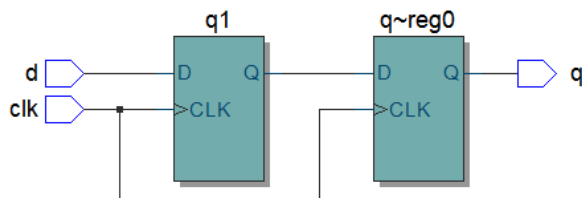
```
module ex_1 (  
  input  clk, in,  
  output reg q1, q0);  
  
  always @(posedge clk) begin  
    q1 <= in;  
    q0 <= in | q1; end  
endmodule
```



Порядок операторов для неблокирующих назначений

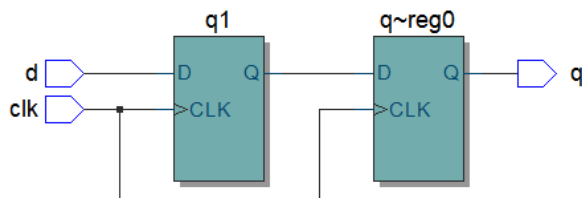
```
module ex_1(  
  output reg q,  
  input d, clk);  
  reg q1;  
  always @(posedge clk) begin  
    q1 <= d;  
    q <= q1; end  
endmodule
```

```
module ex_1(  
  output reg q,  
  input d, clk);  
  reg q1;  
  always @(posedge clk) begin  
    q <= q1;  
    q1 <= d; end  
endmodule
```

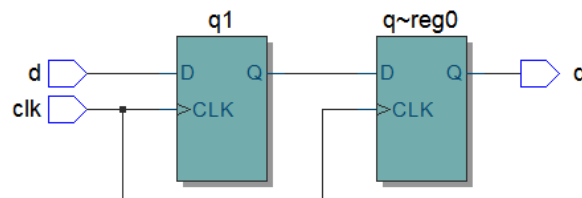


Порядок операторов для неблокирующих назначений

```
module ex_1(  
  output reg q,  
  input d, clk);  
  reg q1;  
  always @(posedge clk) begin  
    q1 <= d;  
    q <= q1; end  
endmodule
```



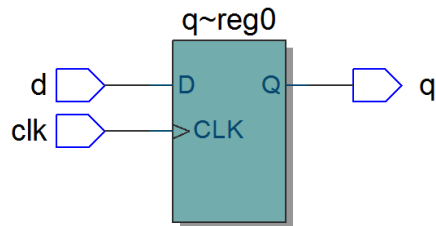
```
module ex_1(  
  output reg q,  
  input d, clk);  
  reg q1;  
  always @(posedge clk) begin  
    q <= q1;  
    q1 <= d; end  
endmodule
```



Порядок операторов для блокирующих назначений

```
module ex_1(  
  output reg q,  
  input d, clk);  
  reg q1;  
  always @(posedge clk) begin  
    q1 = d;  
    q = q1; end  
endmodule
```

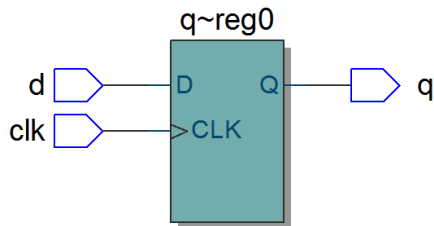
```
module ex_1(  
  output reg q,  
  input d, clk);  
  reg q1;  
  always @(posedge clk) begin  
    q = q1;  
    q1 = d; end  
endmodule
```



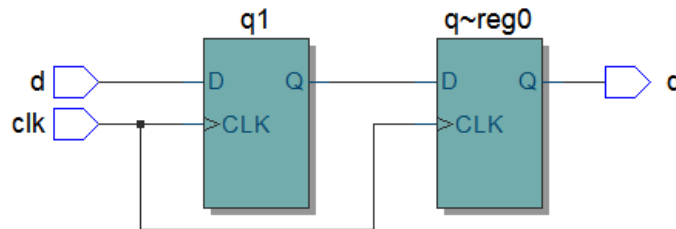
Порядок операторов для блокирующих назначений

□ schematic?

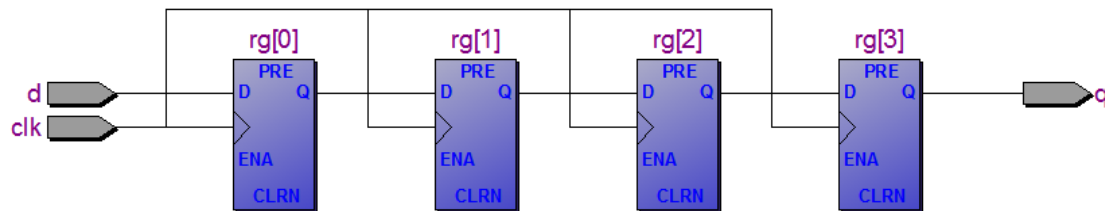
```
module ex_1(  
  output reg q,  
  input d, clk);  
  reg q1;  
  always @(posedge clk) begin  
    q1 = d;  
    q = q1; end  
endmodule
```



```
module ex_1(  
  output reg q,  
  input d, clk);  
  reg q1;  
  always @(posedge clk) begin  
    q = q1;  
    q1 = d; end  
endmodule
```

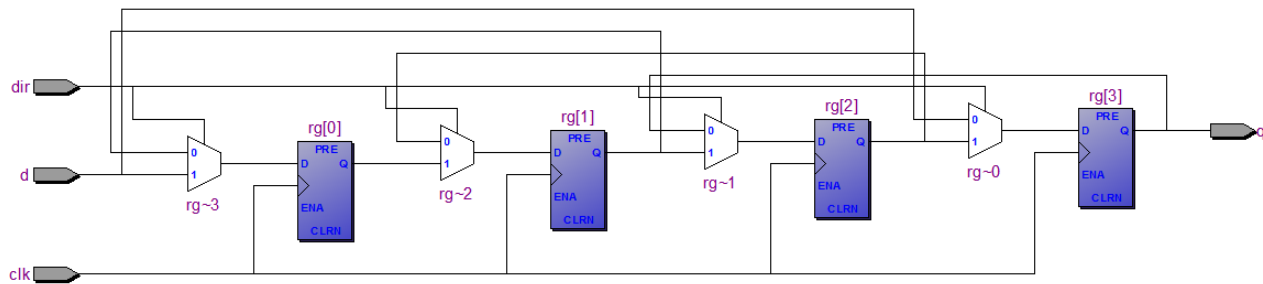


Простейший сдвигающий регистр



```
module rg_ex9 (d, clk, q);  
  input d;  
  input clk;  
  output q ;  
  
  reg [3:0] tmp;  
  
  always @( posedge clk ) tmp <= {tmp[2:0], d};  
  assign q = tmp[3];  
  
endmodule
```

Сдвигающий регистр с выбором направления сдвига



```
module rg_ex10 (d, dir, clk, q);
input  d;
input  clk, dir;
output q ;

reg [3:0] tmp;

always @( posedge clk )
tmp <= (dir)? {tmp[2:0], d} : {d, tmp[3:1]} ;

assign q = tmp[3];

endmodule
```


Сдвигающий регистр с выбором направления сдвига и загрузкой данных

```
module rg_ex11 (d, dir, d_in, load, clk, q);
input  d;
input  [3:0] d_in;
input  clk, dir, load;
output q ;

reg [3:0] tmp;

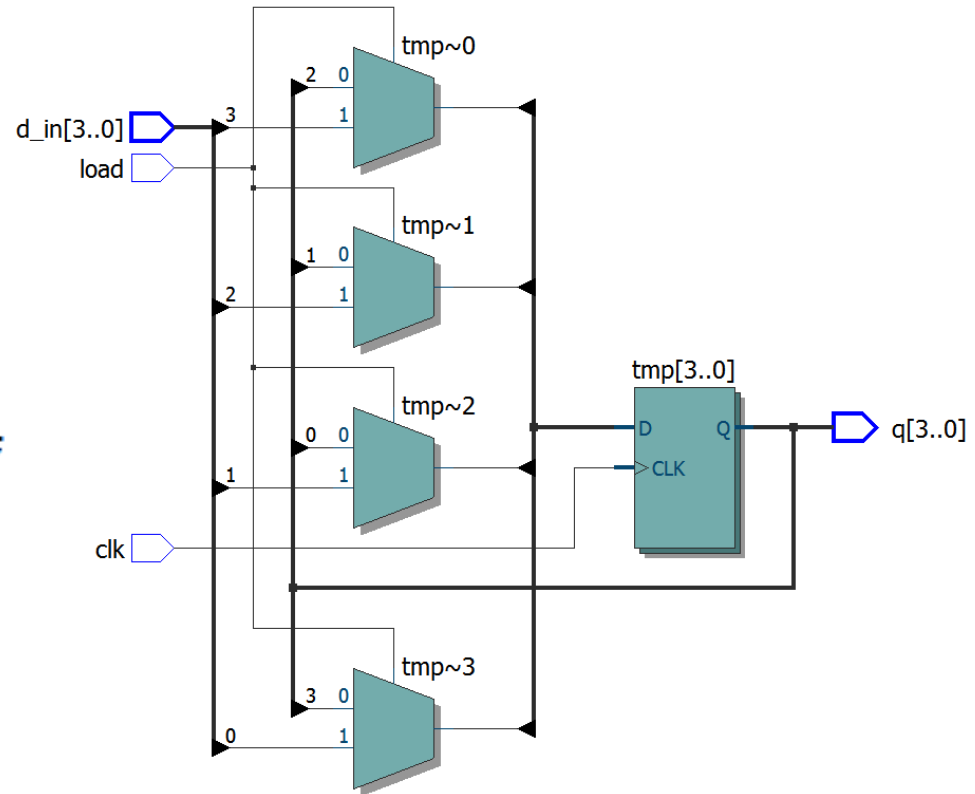
always @( posedge clk )
if (load) tmp <= d_in;
else      tmp <= (dir)? {tmp[2:0], d} : {d, tmp[3:1]} ;

assign q = tmp[3];

endmodule
```

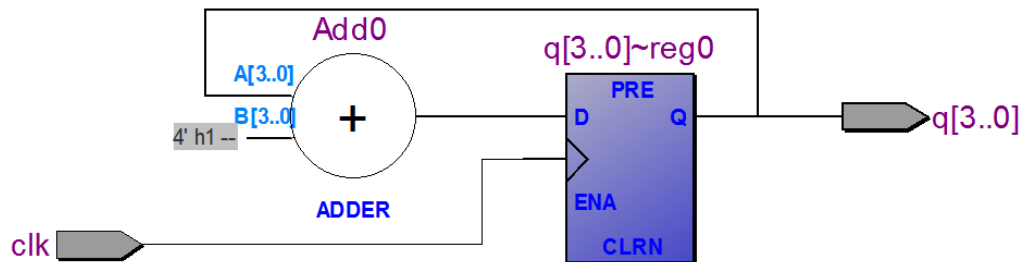
Кольцевой сдвигающий регистр с загрузкой данных

```
module rg_ex12 (d_in, load, clk, q);  
input [3:0] d_in;  
input clk, load;  
output [3:0] q ;  
  
reg [3:0] tmp;  
  
always @( posedge clk )  
if (load) tmp <= d_in;  
else      tmp <= {tmp[2:0], tmp[3]} ;  
  
assign q = tmp;  
  
endmodule
```



Двоичный счетчик

```
module cnt_ex1 (clk, q);  
  input clk;  
  output reg [3:0] q;  
  
  always @(posedge clk)  
    q <= q+4'h1;  
  
endmodule
```

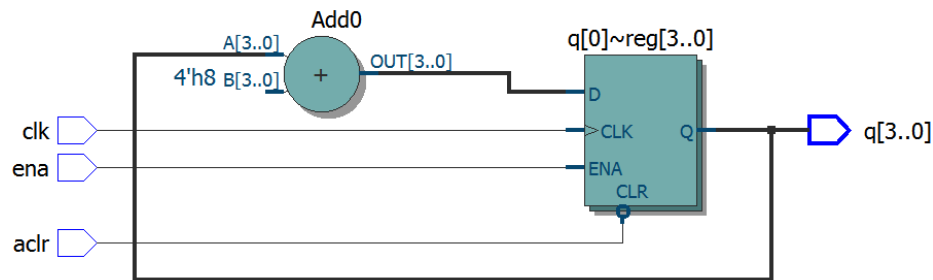


Двоичный счетчик

с входами разрешения работы и асинхронного сброса

aclr	ena	q
0	x	Асинхронный сброс
1	0	хранение
1	1	Счет +

```
module cnt_ex2 (clk, aclr, ena, q);  
input clk, aclr, ena;  
output reg [3:0] q;  
  
always @(posedge clk, negedge aclr)  
if (aclr == 0) q <= 0;  
else if (ena) q <= q+4'h1;  
  
endmodule
```



4-разрядный реверсивный счетчик с параллельной загрузкой

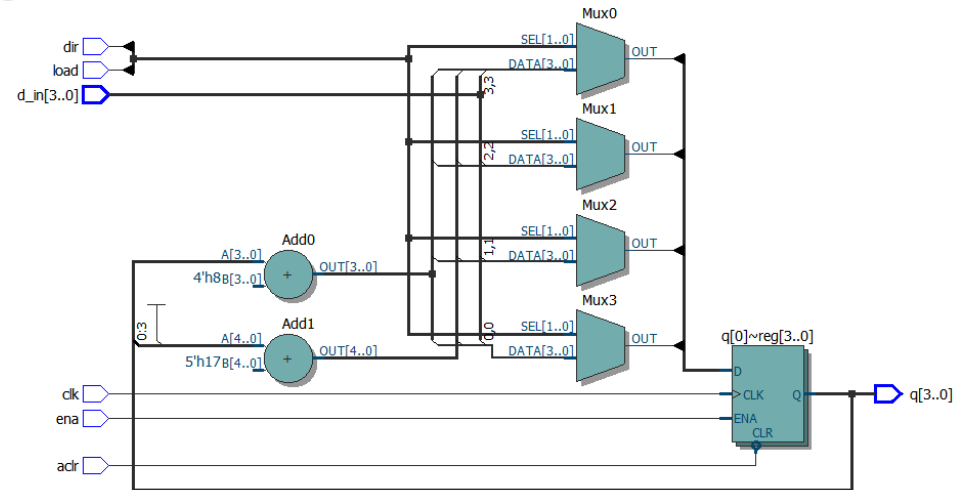
aclr	ena	load	din	dir	q
0	x	x	x	x	асинхронный сброс
1	0	x	x	x	хранение
1	1	0	din	x	Запись din
1	1	1	x	1	Счет +
1	1	1	x	0	Счет -

```

module cnt_ex3 (clk, aclr, dir, load, d_in, ena, q);
input clk, aclr, ena, dir, load;
input [3:0] d_in;
output reg [3:0] q;

always @(posedge clk, negedge aclr)
if (aclr == 0) q <= 0;
else if (ena)
    casex ({dir, load})
        2'bx0 : q <= d_in;
        2'b11 : q <= q+4'h1;
        2'b01 : q <= q-4'h1;
    endcase
endmodule

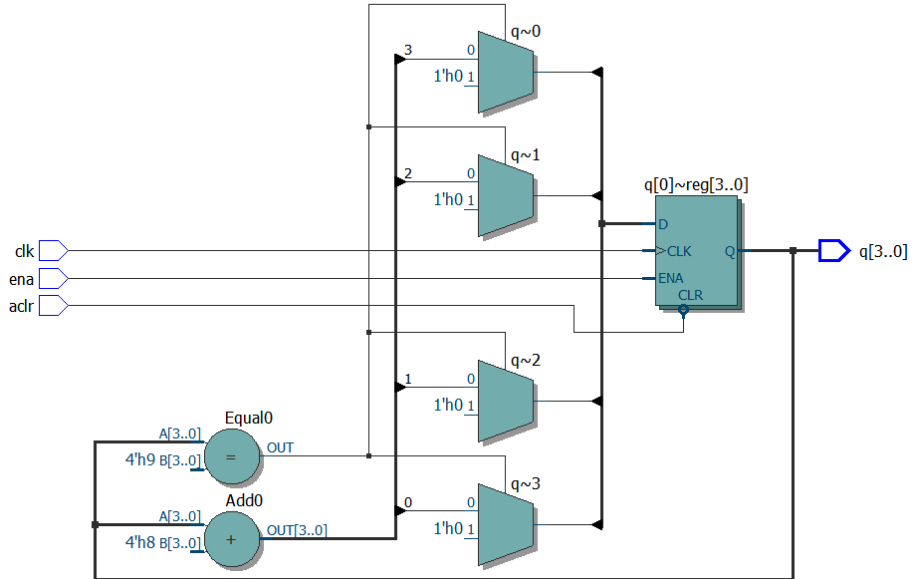
```



Двоично-десятичный счетчик

❑ Счетчик считает от 0 до 9, далее 0 и т.д.

```
module cnt_ex5 (clk, aclr, ena, q);  
input clk, aclr, ena;  
output reg [3:0] q;  
  
always @(posedge clk, negedge aclr)  
if (aclr == 0) q <= 0;  
else if (ena)  
    if (q == 4'h9) q <= 4'h0;  
    else  
        q <= q+4'h1;  
endmodule
```



Реверсивный счетчик с параллельной загрузкой и программируемым модулем счета

aclr	ena	load	din	dir	q
0	x	x	x	x	асинхронный сброс
1	0	x	x	x	хранение
1	1	0	din	x	Запись din
1	1	1	x	1	Счет +
1	1	1	x	0	Счет -

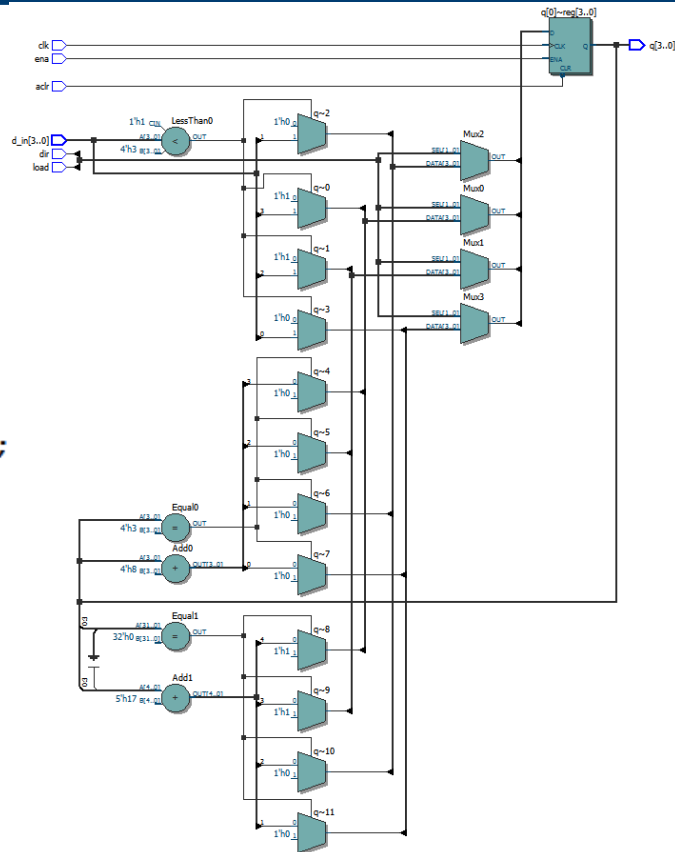
*Если $din > (module-1)$,
то записывается $(module-1)$

(часть 2)

```
module cnt_ex4 (clk, aclr, dir, load, d_in, ena, q);
input clk, aclr, ena, dir, load;
input [3:0] d_in;
output reg [3:0] q;

wire [3:0] cnt_mod = 12; //= mod-1

always @(posedge clk, negedge aclr)
if (aclr == 0) q <= 0;
else if (ena)
  casex ({dir, load})
    2'bx0 : if (d_in <= cnt_mod) q <= d_in;      else q <= cnt_mod;
    2'b11 : if (q == cnt_mod) q <= 0;           else q <= q+4'h1;
    2'b01 : if (q == 0) q <= cnt_mod;           else q <= q-4'h1;
  endcase
endmodule
```



Счетчик с выходом переноса

- ❑ Двоичный счетчик с выходом сигнала переноса (перенос = 1 при достижении счетчиком своего максимального значения)
- ❑ Использование:
 - ✓ Для каскадного соединения счетчиков
 - ✓ Для реализации счетчиков-делителей

reset	ena	q	cout
0	x	Асинхронный сброс	0
1	0	хранение	хранение
1	1	Счет +	0

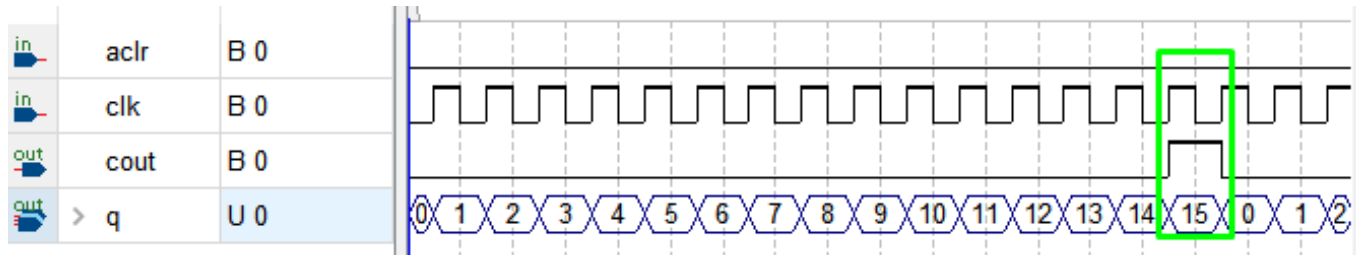
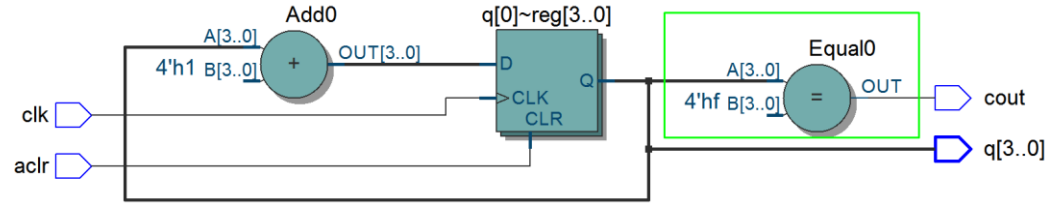
Счетчик с выходом переноса (асинхронным)

```
module ex_1
(input  clk, aclr,
 output cout,
 output reg [3:0] q);

assign cout = ( q == 4'hf);

always @(posedge clk, posedge aclr)
    if (aclr) q <= 4'd0;
    else     q <= q + 4'd1;

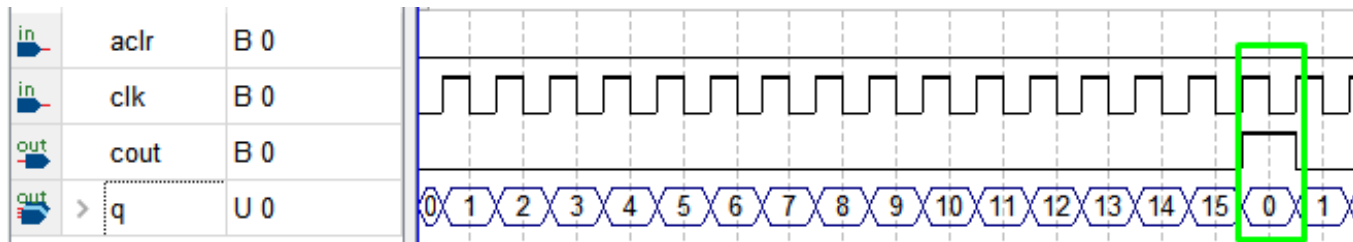
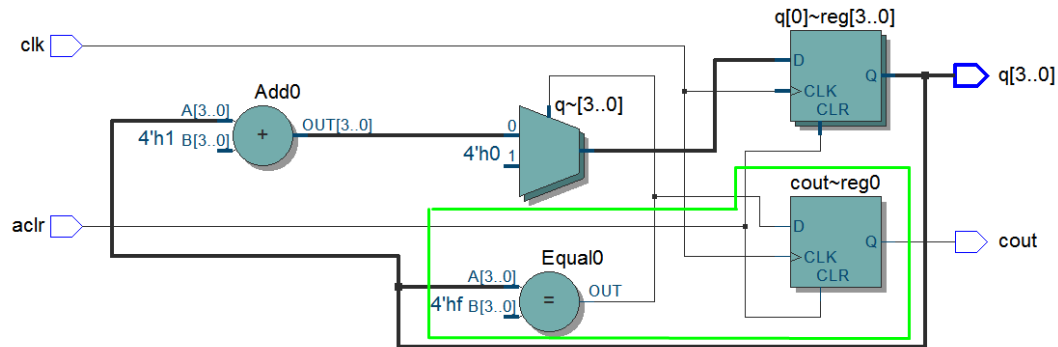
endmodule
```



Счетчик с выходом переноса (синхронным) вар.1

```
module ex_1
(input  clk, aclr,
output reg cout,
output reg [3:0] q);

always @(posedge clk, posedge aclr)
    if (aclr) begin
        q <= 4'd0;
        cout <= 1'd0; end
    else if (q == 4'hf) begin
        q <= 4'd0;
        cout <= 1'd1; end
    else begin
        q <= q + 4'd1;
        cout <= 1'd0; end
endmodule
```



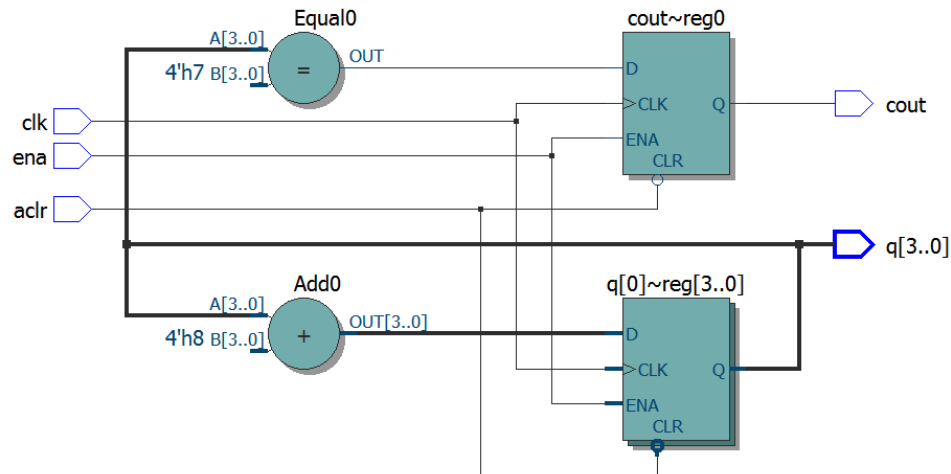
Счетчик с выходом переноса (синхронным) вар.2

```
module cnt_ex6 (clk, aclr, ena, q, cout);  
input clk, aclr, ena;  
output reg [3:0] q;  
output reg cout;
```

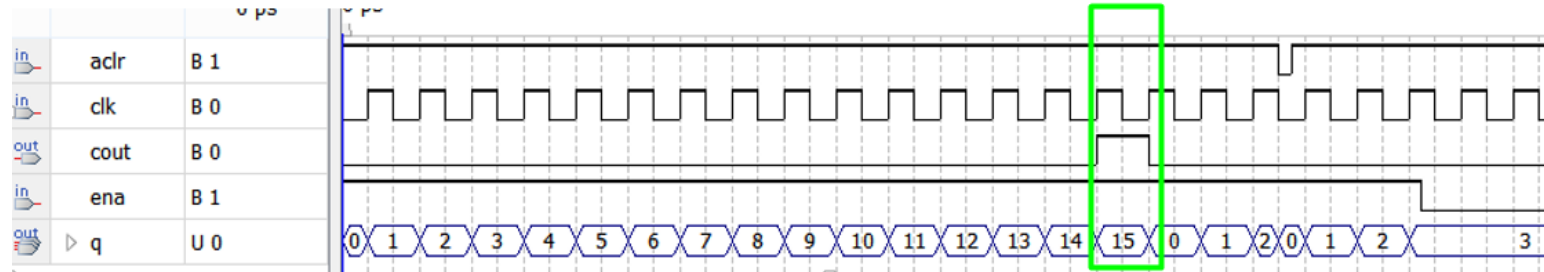
```
always @(posedge clk, negedge aclr)  
if (aclr == 0) q <= 4'h0;  
else if (ena) q <= q+4'h1;
```

```
always @(posedge clk, negedge aclr)  
if (aclr == 0) cout <= 1'b0;  
else if (ena)  
    if (q==4'he) cout <= 1'b1;  
    else cout <= 1'b0;
```

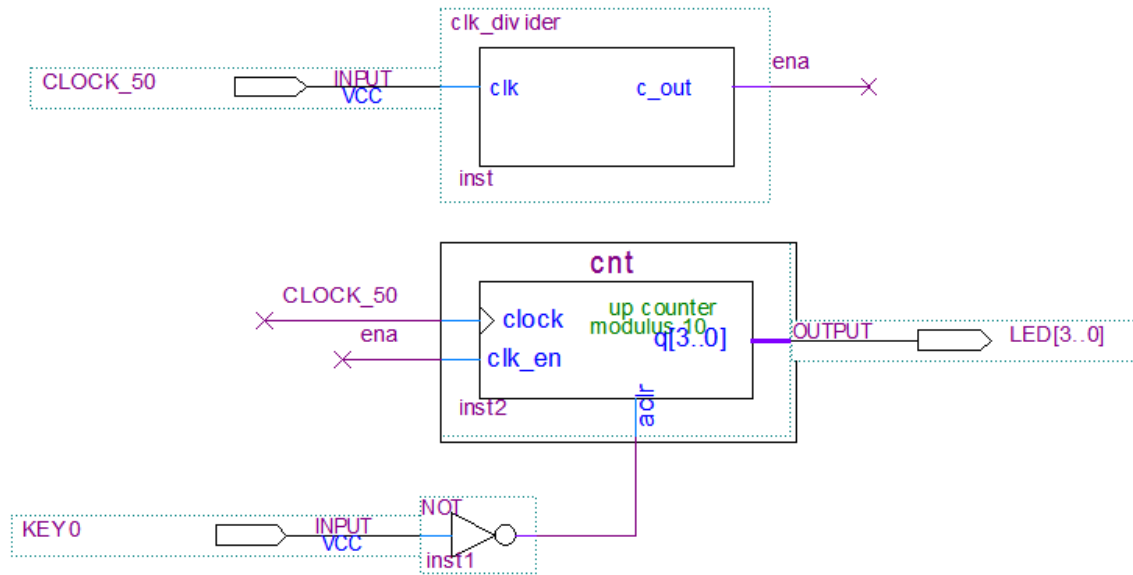
```
endmodule
```



50,0 ns 240,0 ns 320,0 ns 400,0 ns 480,0 ns 560,0 ns 640,0 ns 720,0 ns 800,0 ns 880,0 ns



Каскадное соединение счетчиков



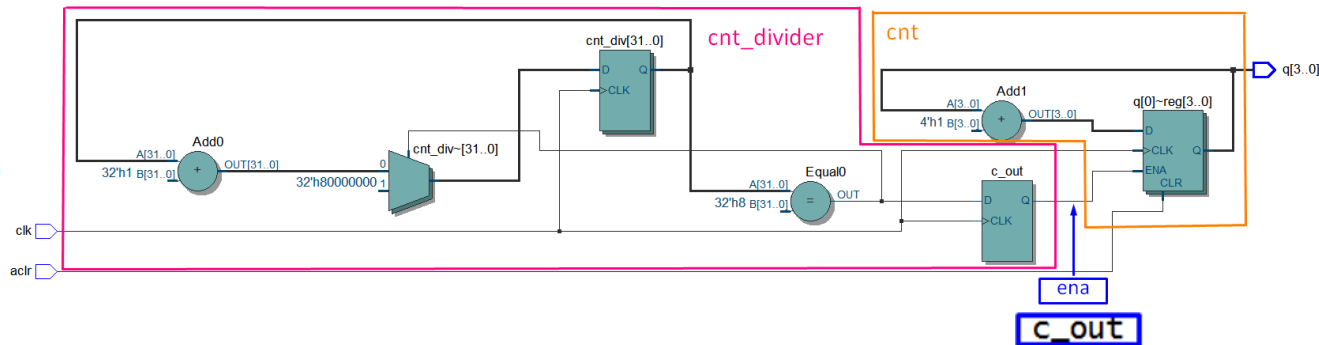
Каскадное соединение счетчиков

```
module ex_1
# (parameter div_by = 8)
(input clk, aclr,
 output reg [3:0] q);
  reg c_out;
  reg [31:0] cnt_div;
```

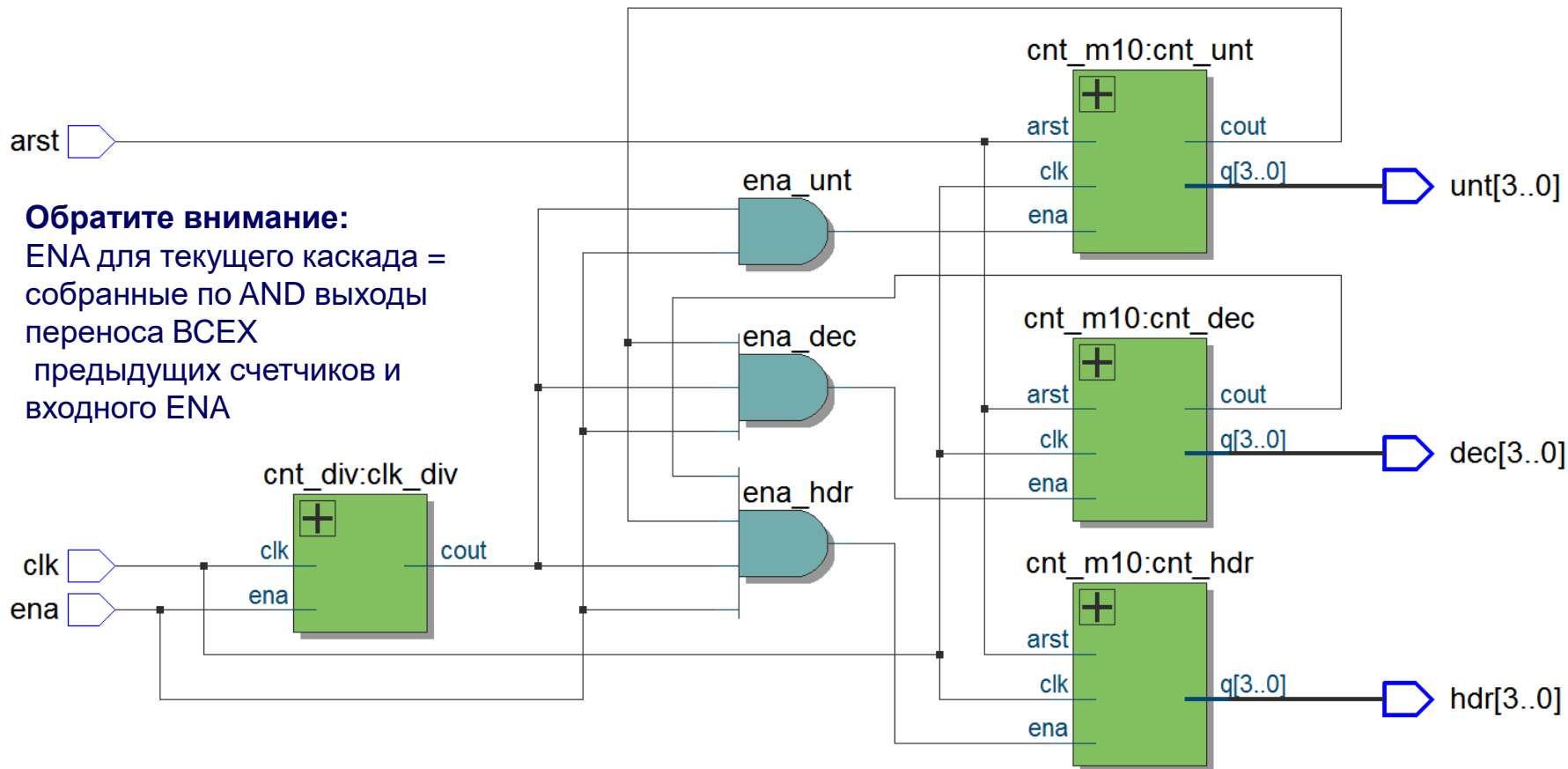
```
always @(posedge clk )
if (cnt_div==div_by) begin
  cnt_div  <= 1;
  c_out    <= 1'b1; end
else begin
  cnt_div  <= cnt_div+1;
  c_out    <= 1'b0; end
```

```
always @(posedge clk, posedge aclr)
  if (aclr)    q <= 4'd0;
  else if (c_out) q <= q + 1'b1;
```

```
endmodule
```



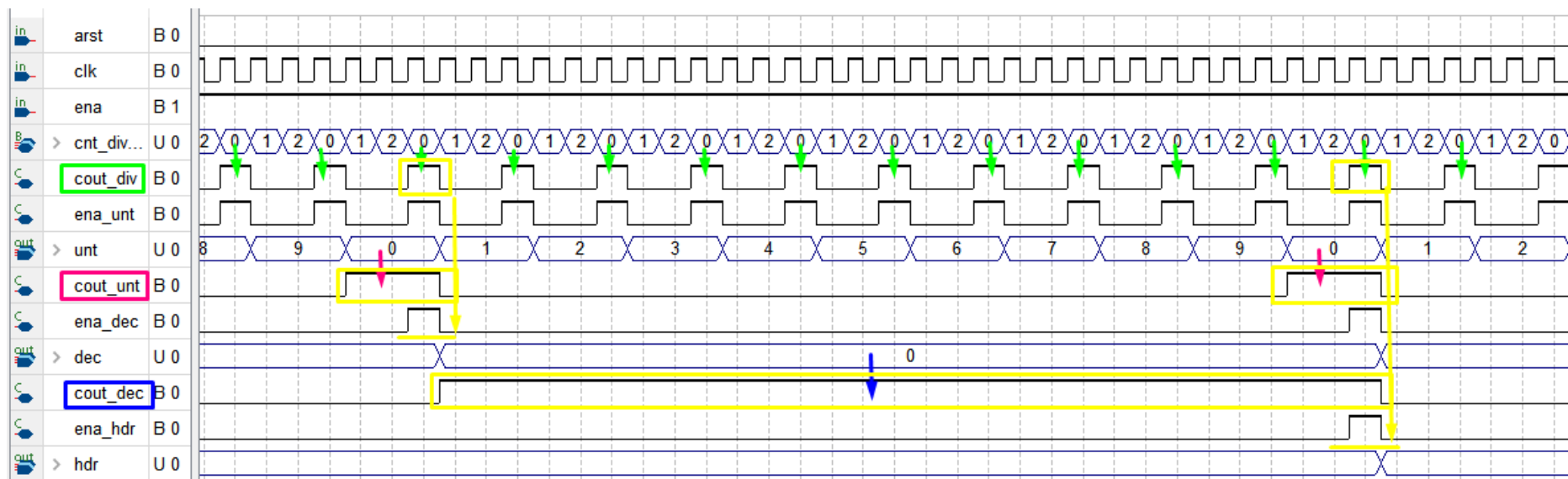
Каскадное соединение счетчиков



Каскадное соединение счетчиков

Обратите внимание:

ENA для текущего каскада = собранные по AND выходы переноса BCEX предыдущих счетчиков и входного ENA



altera_attributes (pin name & I/O standard)

НАЗНАЧЕНИЕ ВЫВОДОВ

```
module ex_1 (  
    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVTTL\"", chip_pin = "AA14" *) //KEY[0]  
    input rst,  
    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVTTL\"", chip_pin = "AA15" *) //clock  
    input clk,  
    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVTTL\"", chip_pin = "AF14" *) //KEY[1]  
    input updn,  
  
    (* altera_attribute = "-name IO_STANDARD \"3.3-V LVTTL\"", chip_pin = "V17, W16, V16" *)  
    //LEDR[2] LEDR[1] LEDR[0]  
    output reg [2:0] q);  
  
always@(posedge clk, negedge rst)  
begin  
    if (!rst) q <= 3'd0;  
    else      q <= q + (updn ? 1'd1 : -1'd1);  
end  
endmodule
```

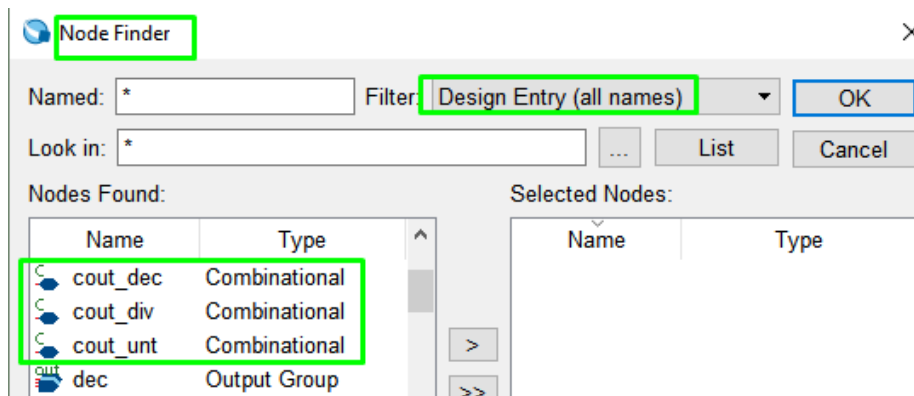


Signal Name	FPGA Pin No.	Description	I/O Standard
KEY[0]	PIN_AA14	Push-button[0]	3.3V
KEY[1]	PIN_AA15	Push-button[1]	3.3V
KEY[2]	PIN_W15	Push-button[2]	3.3V
KEY[3]	PIN_Y16	Push-button[3]	3.3V

Node Name	Direction	I/O Bank	Location	I/O Standard
in clk	Input	3B	PIN AA15	3.3-V LVTTL
out q[2]	Output	4A	PIN V17	3.3-V LVTTL
out q[1]	Output	4A	PIN W16	3.3-V LVTTL
out q[0]	Output	4A	PIN V16	3.3-V LVTTL
in rst	Input	3B	PIN AA14	3.3-V LVTTL
in updn	Input	3B	PIN AF14	3.3-V LVTTL

Как сохранить цепи для моделирования

- ❑ wire/reg могут быть минимизированы системой синтеза (Analysis & Synthesis).
 - ✓ При этом минимизированные wire/reg будут не доступны при моделировании.
- ❑ Для сохранения wire/reg для моделирования:
 - ✓ Используйте атрибут **keep**
 - *(*keep*) wire cout_div, cout_unt, cout_dec;*



Описание комбинационных и триггерных схем

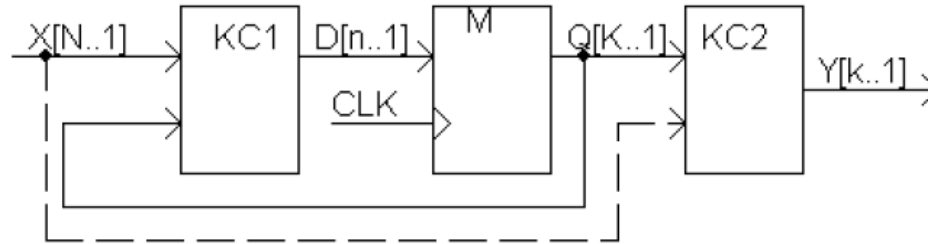
- ❑ RTL описания используют процедурные блоки для описания комбинационных и триггерных схем
 - ✓ Комбинационный - Combinatorial Process
 - ✓ Тактовый (регистровый) - Clocked Process
- ❑ Для описания комбинационных схем используются **блокирующие** назначения
- ❑ Для описания триггерных схем используются **не блокирующие** назначения



Конечный автомат

Конечный автомат

- ❑ Конечный автомат – устройство с памятью, выходные сигналы которого зависят от истории поступления входных сигналов.

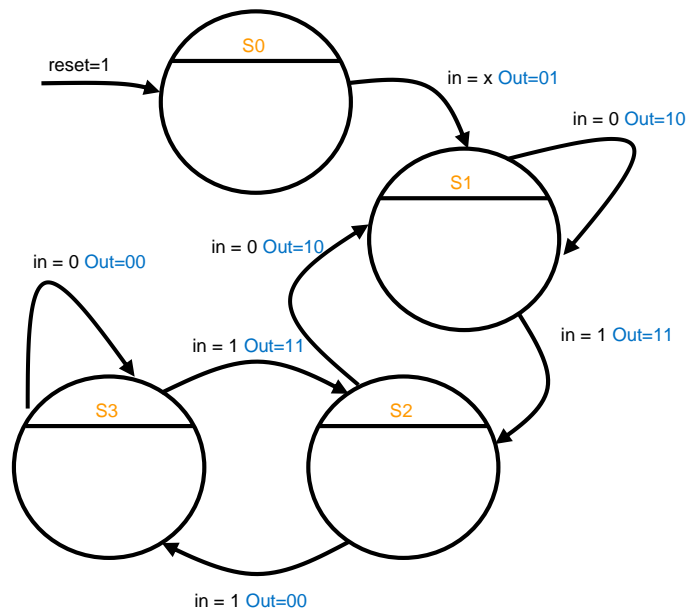
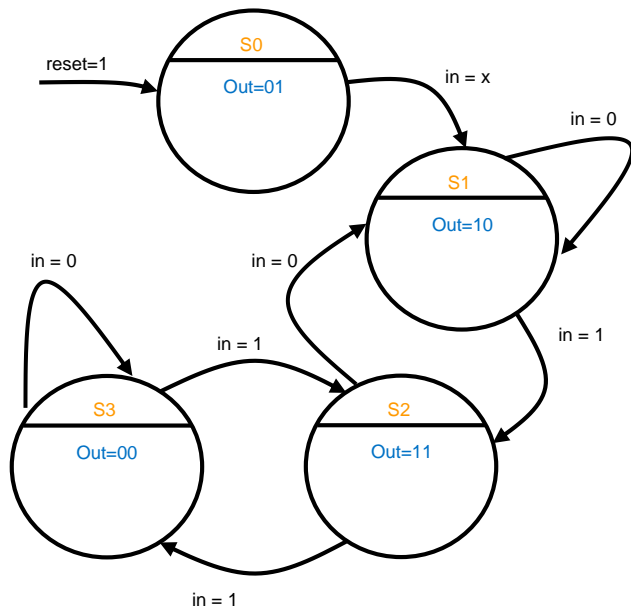


- ❑ Обобщенная структурная схема конечного автомата.
 - ✓ $X[N..1]$ –входные сигналы; $Y[k..1]$ – выходные сигналы;
 - ✓ $Q[K..1]$ –разряды памяти, определяющие состояние автомата; $D[n..1]$ –данные для записи в память;
 - ✓ М-память автомата (набор триггеров);
 - ✓ КС1 –комбинационная схема, обеспечивающая формирование данных для записи в память; КС2 –комбинационная схема, формирующая выходные сигналы.

Задание Конечного Автомата (КА)

□ В зависимости от способа формирования выходных сигналов выделяют два класса конечных автоматов:

- ✓ Автомат Мура – выходные сигналы зависят только от текущего состояния автомата;
- ✓ Автомат Мили – выходные сигналы зависят от текущего состояния автомата и от текущих входных сигналов



Конечный автомат

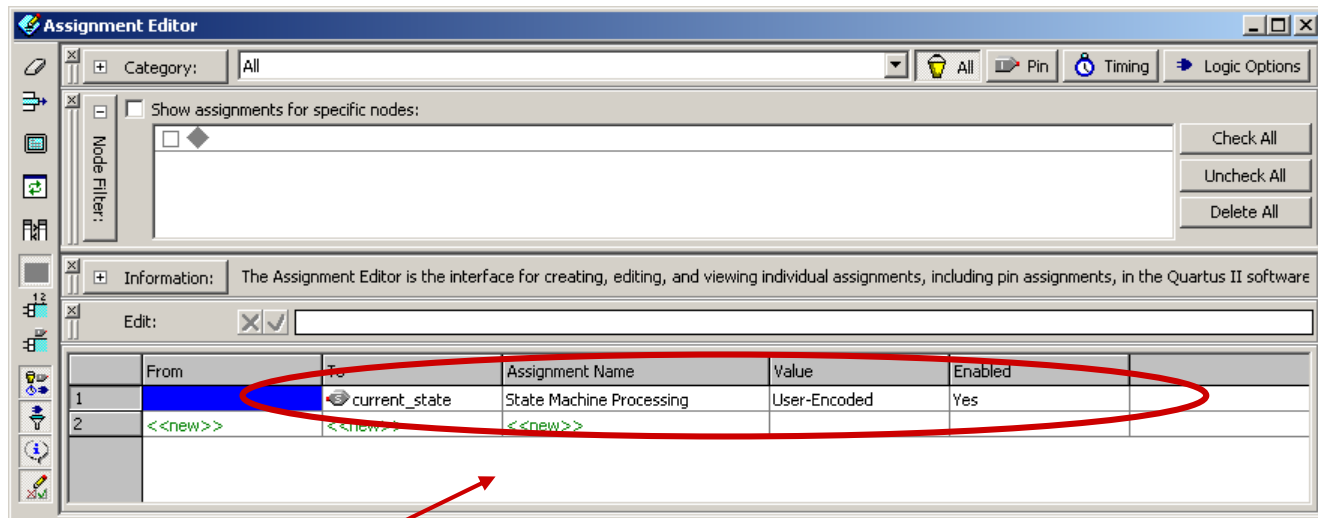
- ❑ Число триггеров (n_{tr}), используемых для реализации модуля памяти автомата, определяется числом состояний автомата (N_c) и способом их кодирования:
 - ✓ Двоичное кодирование (Binary coding). При этом $n_{tr} = \lceil \log_2 N_c \rceil$
 - ✓ Кодирование по принципу: одно состояние—один триггер (One Hot State). При этом $n_{tr} = N_c$.
- ❑ Для задания состояний конечного автомата используется
 - ✓ parameter
 - ✓ localparam

Кодирование состояний КА

State	Binary Encoding	Grey-Code Encoding	One-Hot Encoding	Custom Encoding
Idle	000	000	00001	?
Fill	001	001	00010	?
Heat_w	010	011	00100	?
Wash	011	010	01000	?
Drain	100	110	10000	?

- По умолчанию в пакете Quartus II
 - One-hot кодирование для СБИС с look-up table (LUT)
 - СБИС содержит много триггеров
 - Binary (minimal bit) для СБИС с product-term архитектурой (MAX)
 - СБИС содержит мало регистров, но многовходовые логические матрицы

Задание способа кодирования в Quartus II



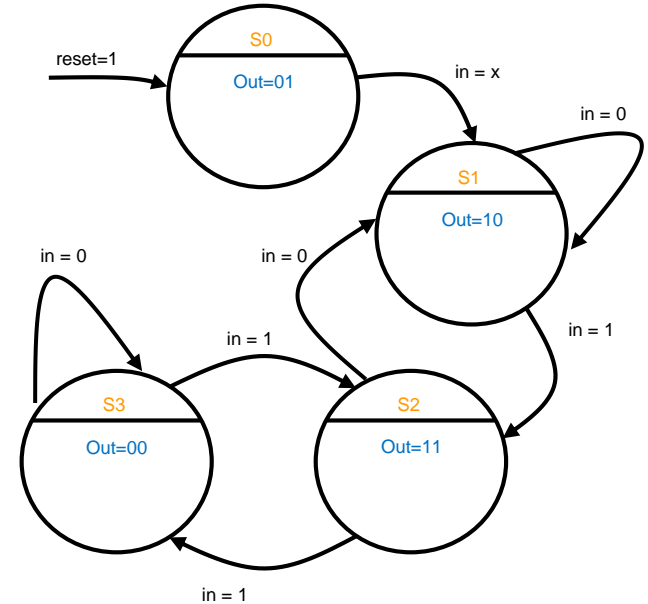
**Назначения выполняются для
сигнала, определяющего
состояние КА**

Варианты:

- One-Hot
- Gray
- Minimal Bits
- Sequential
- User-Encoded

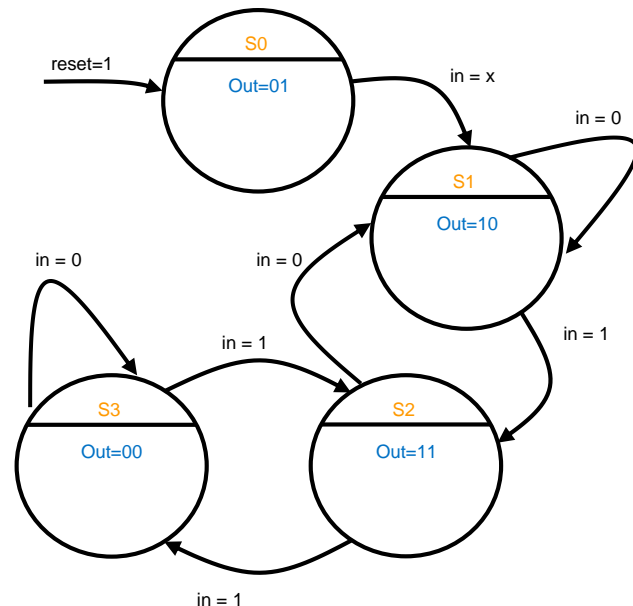
Объявление состояний КА

```
1 module FSM_Moore
2   (input clk, in, reset,
3    output reg [1:0] out
4   );
5
6   // Declare state register
7   reg [1:0] state;
8
9   // Declare states
10  parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3;
```



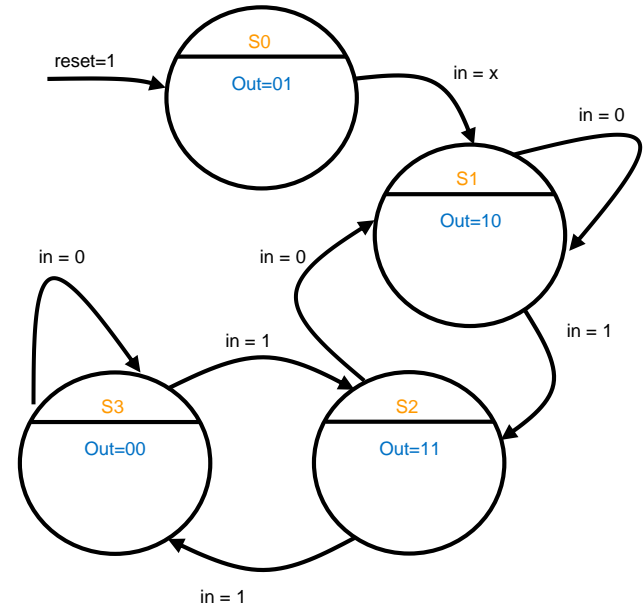
Описание таблицы переходов КА

```
12 // Next state logic
13 always @ (posedge clk or posedge reset) begin
14     if (reset)
15         state <= S0;
16     else
17         case (state)
18             S0: state <= S1;
19             S1: if (in) state <= S2;
20                 //else state <= S1;
21             S2: if (in) state <= S3;
22                 else state <= S1;
23
24             S3: if (in) state <= S2;
25                 //else state <= S3;
26         endcase
27     end
28
```

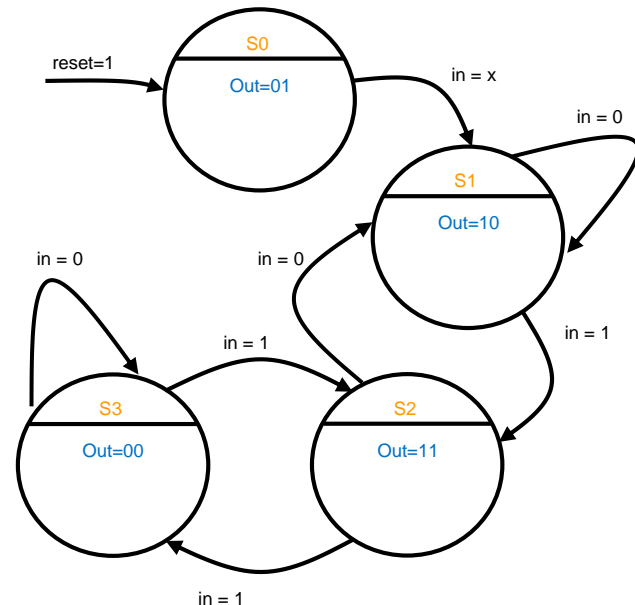
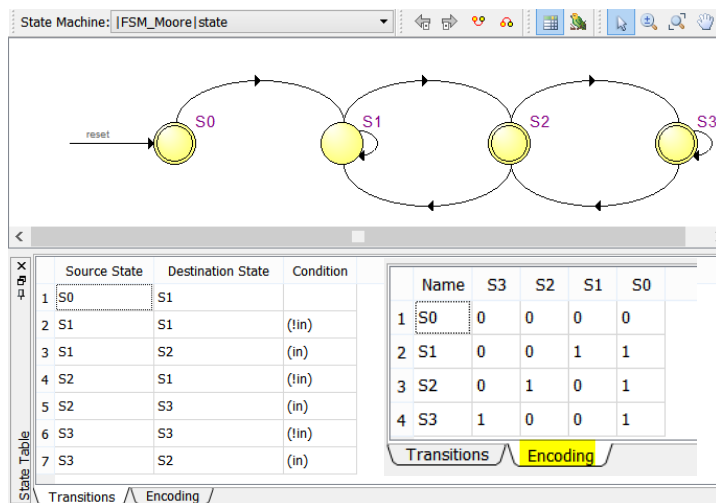
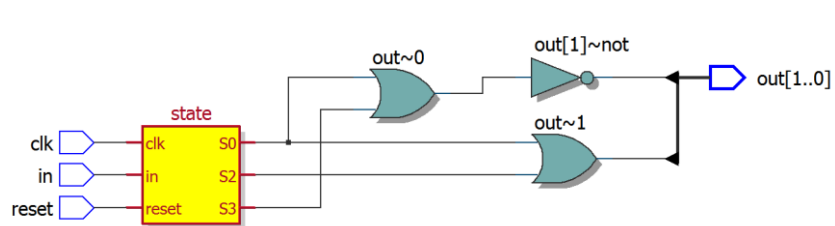


Описание таблицы выходов КА

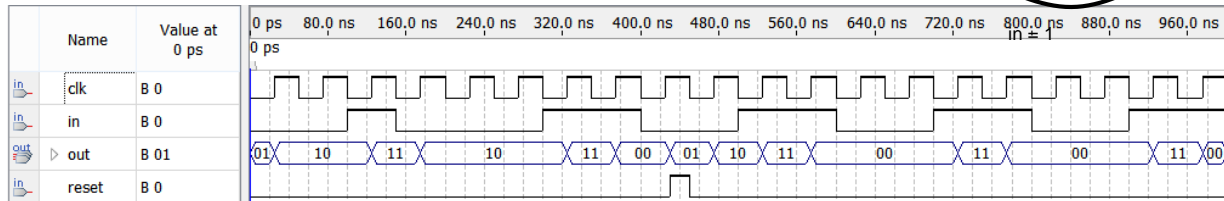
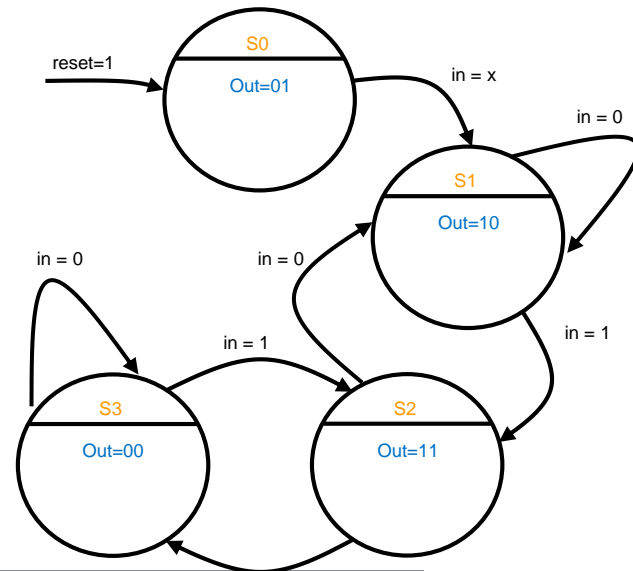
```
29 // Output depends only on the state
30 always @ (state) begin
31     case (state)
32         S0: out = 2'b01;
33         S1: out = 2'b10;
34         S2: out = 2'b11;
35         S3: out = 2'b00;
36         default:
37             out = 2'b00;
38     endcase
39 end
40
41 endmodule
```



Результаты синтеза

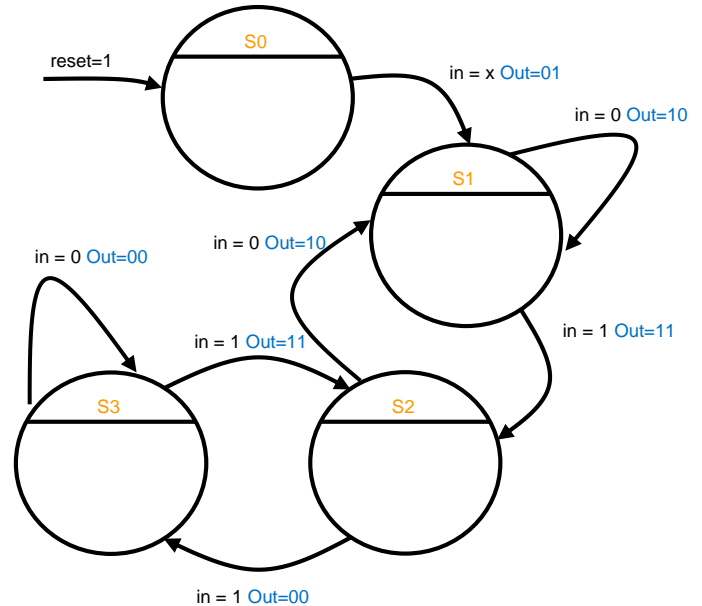


Моделирование



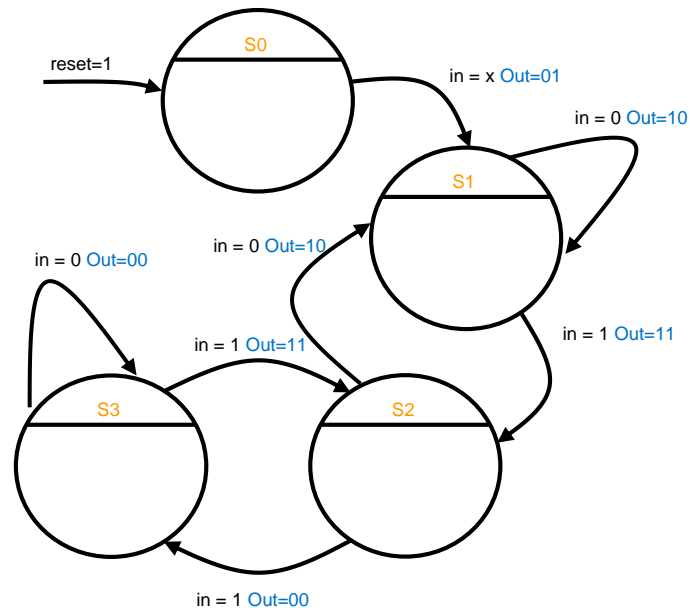
Объявление состояний КА (Мили)

```
1 module FSM_Mealy
2   ( input clk, in, reset,
3     output reg [1:0] out
4   );
5
6   // Declare state register
7   reg [1:0] state;
8
9   // Declare states
10  parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3;
```



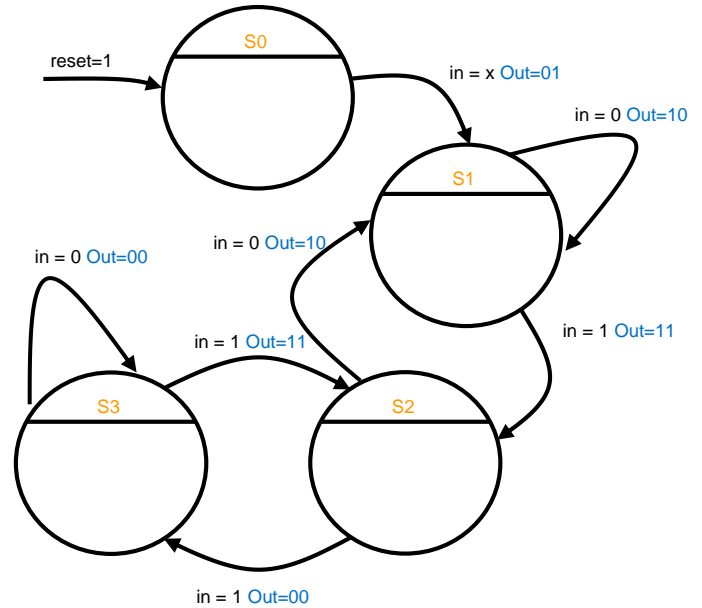
Описание таблицы переходов КА

```
12 // Next state logic
13 always @ (posedge clk or posedge reset) begin
14     if (reset)
15         state <= S0;
16     else
17         case (state)
18             S0: state <= S1;
19             S1: if (in) state <= S2;
20                 //else state <= S1;
21             S2: if (in) state <= S3;
22                 else state <= S1;
23             S3: if (in) state <= S2;
24                 //else state <= S3;
25         endcase
26     end
27
28
```

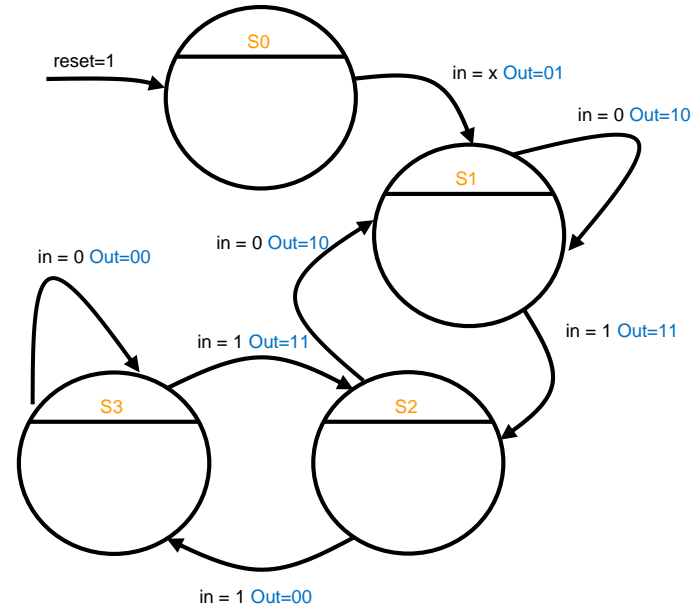
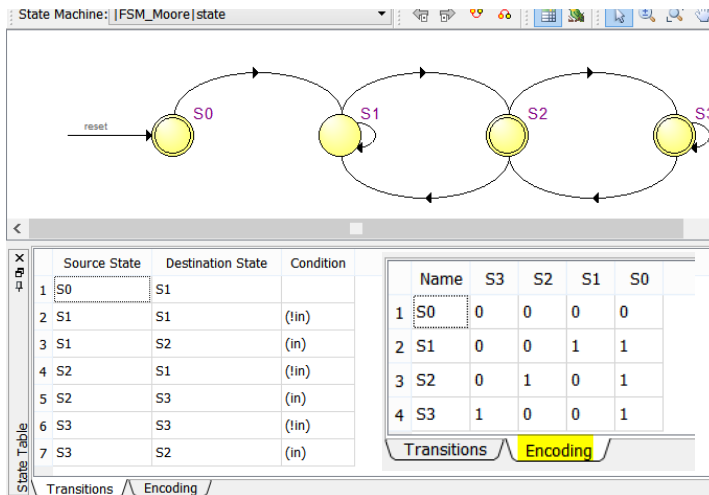
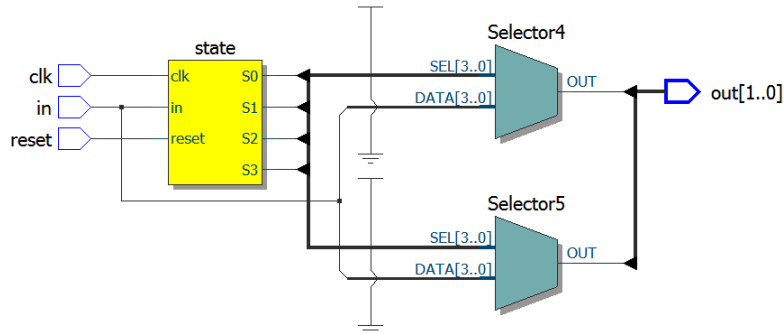


Описание таблицы выходов КА

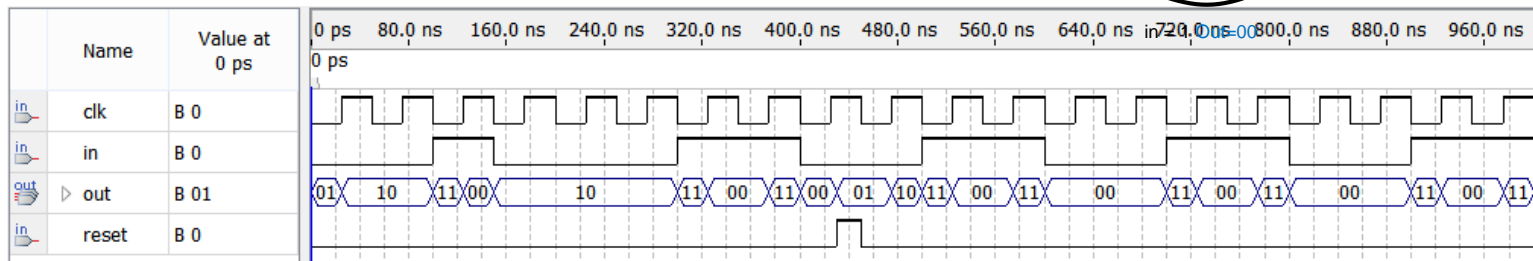
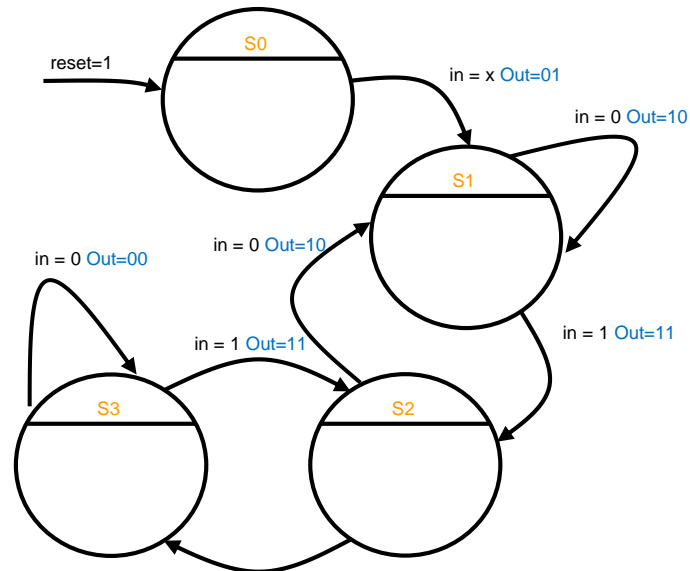
```
31 always @ (state or in)
32 begin
33     case (state)
34         S0:         out = 2'b01;
35
36         S1:
37             if (in) out = 2'b11;
38             else   out = 2'b10;
39
40         S2:
41             if (in) out = 2'b00;
42             else   out = 2'b10;
43
44         S3:
45             if (in) out = 2'b11;
46             else   out = 2'b00;
47
48     endcase
49 end
```



Результаты синтеза



Моделирование



Неопределенные состояния КА


State	Binary Encoding	Grey-Code Encoding	One-Hot Encoding	Custom Encoding
Idle	000	000	00001	?
Fill	001	001	00010	?
Heat_w	010	011	00100	?
Wash	011	010	01000	?
Drain	100	110	10000	?
Неопределенные состояния	101; 110; 111	100;101; 111	32-5 состояний	

- Любые сбои в аппаратуре могут служить причиной перехода КА в неопределенные состояния (не описанные в алгоритме работы)
- Переход в неопределенное состояние приводит к «зависанию» КА

Создание «надежных» КА

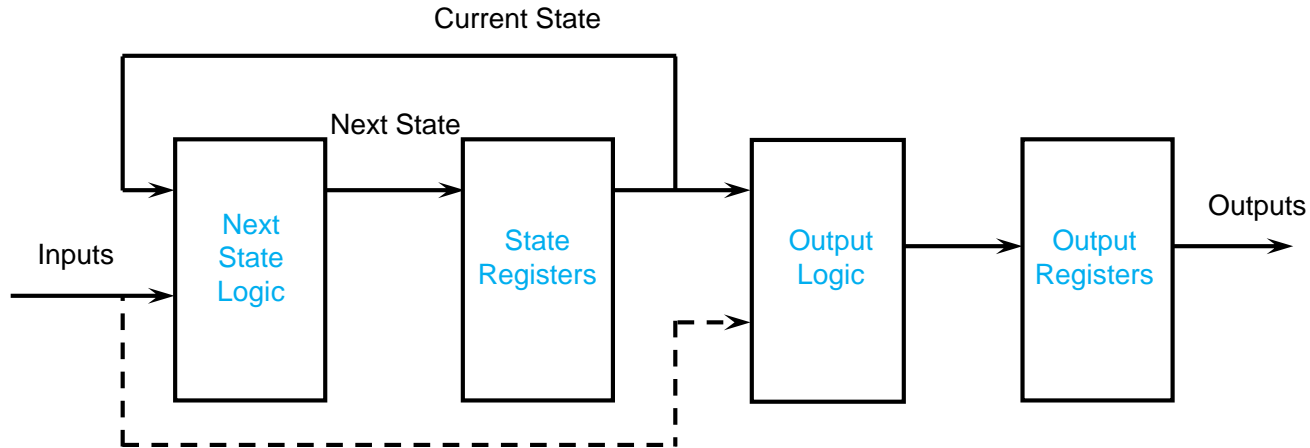
❑ Необходимо использовать:

- ✓ Назначение SAFE STATE MACHINE пакета QuartusII (для всего проекта или отдельного КА), что может потребовать дополнительных логических элементов

	From	To	Assignment Name	Value	Enabled
1		 current_state	Safe State Machine	On	Yes
2	<<new>>	<<new>>	<<new>>		

Синхронизация выходов КС2

- ❑ Синхронизация выходов КС2 (output logic) приводит к появлению дополнительной задержки



Синхронизация выходов КС2 без дополнительной задержки

- Использовать для КС2 next state а не current state

