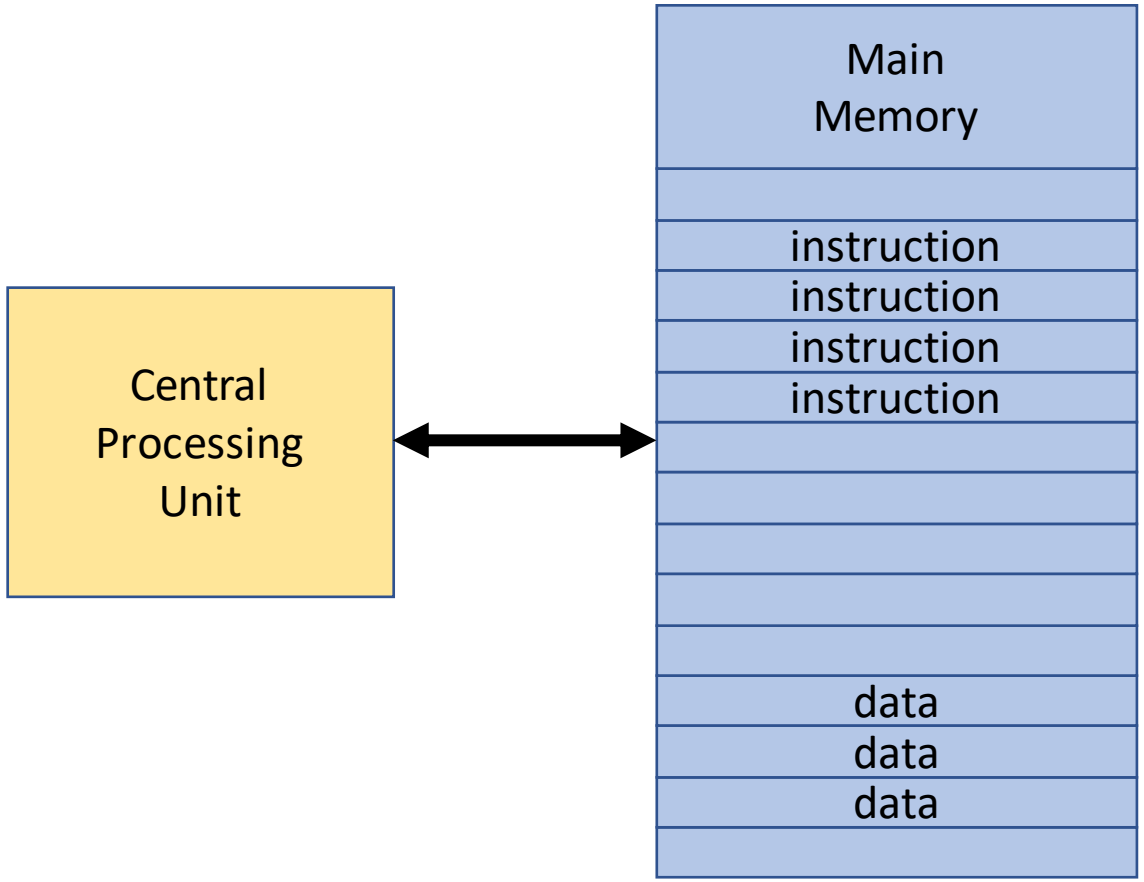
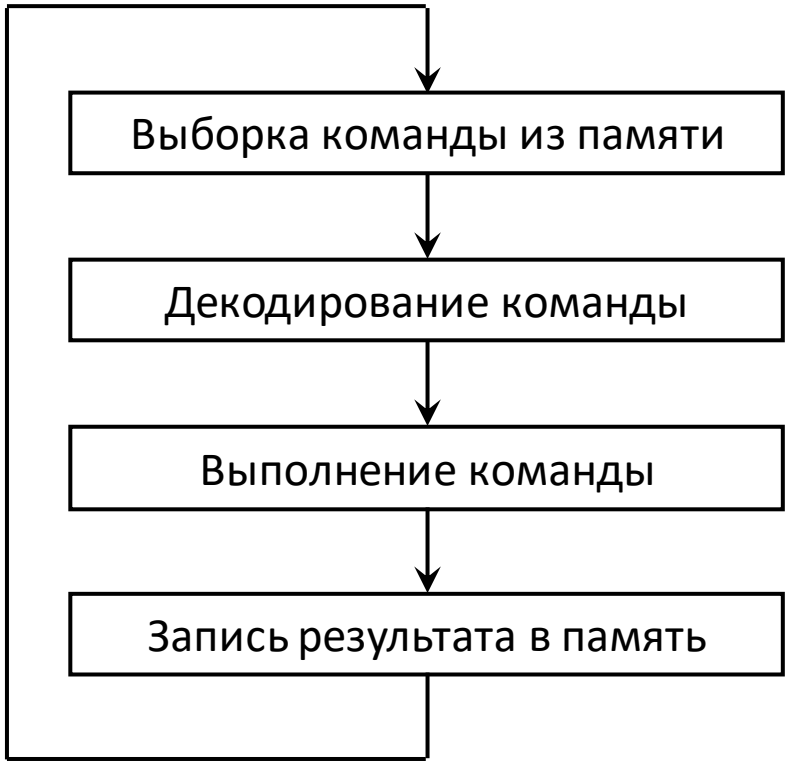
Лекция 2. Основные концепции и инструменты

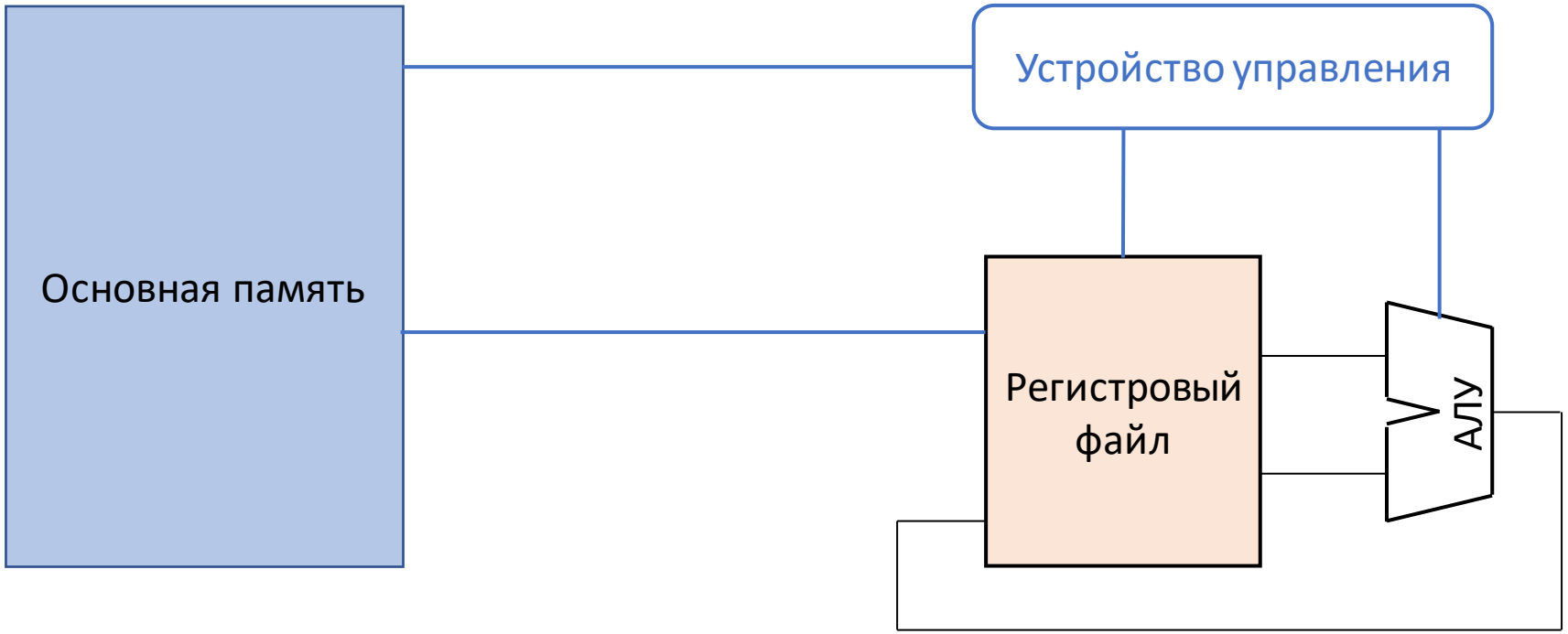
План лекции

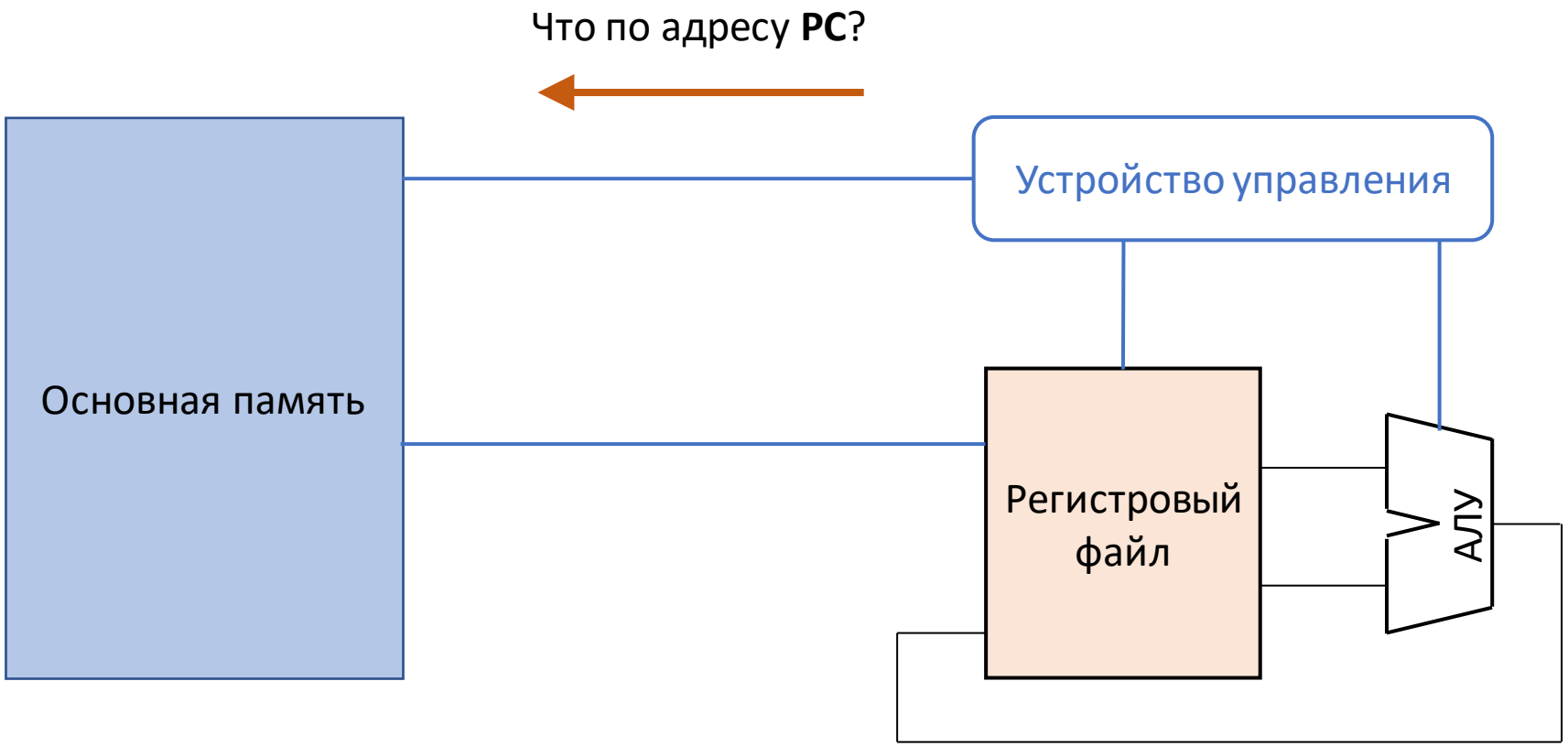
- Обобщенная структура процессора
- Классификация цифровых интегральных схем
- Программируемая логическая интегральная схема (FPGA)
- Основы языка описания аппаратуры Verilog HDL
- Критический путь

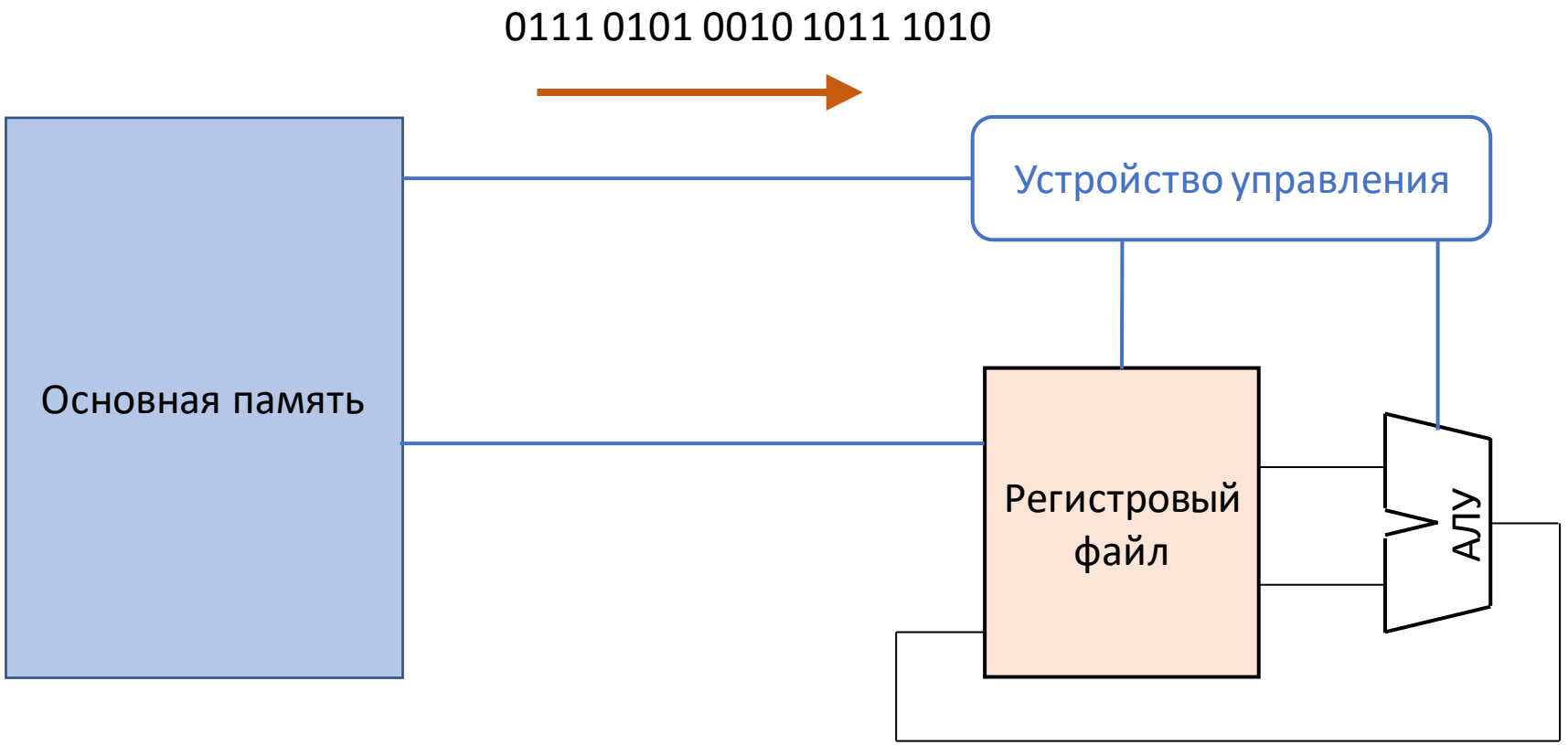
Машина с хранимой в памяти программой

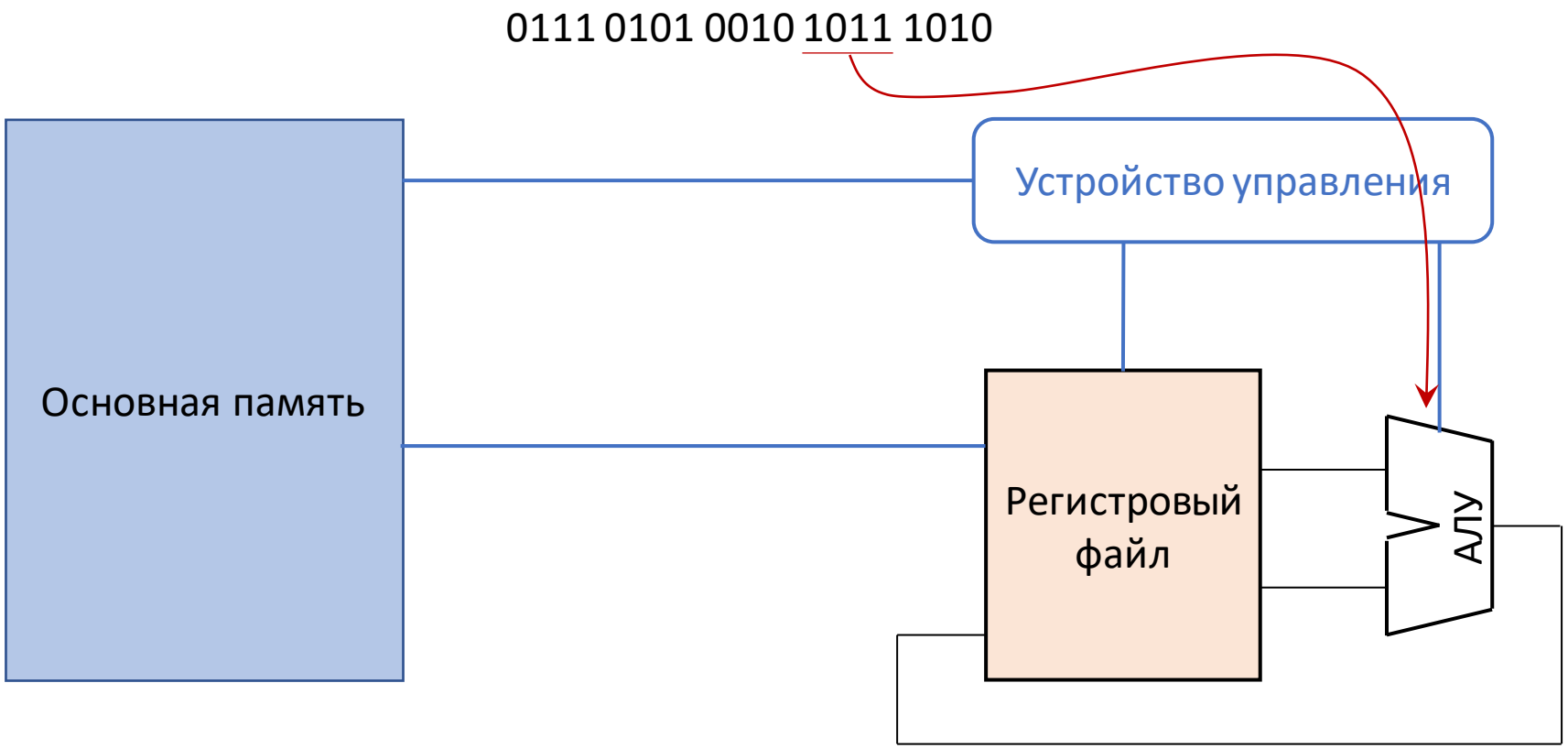


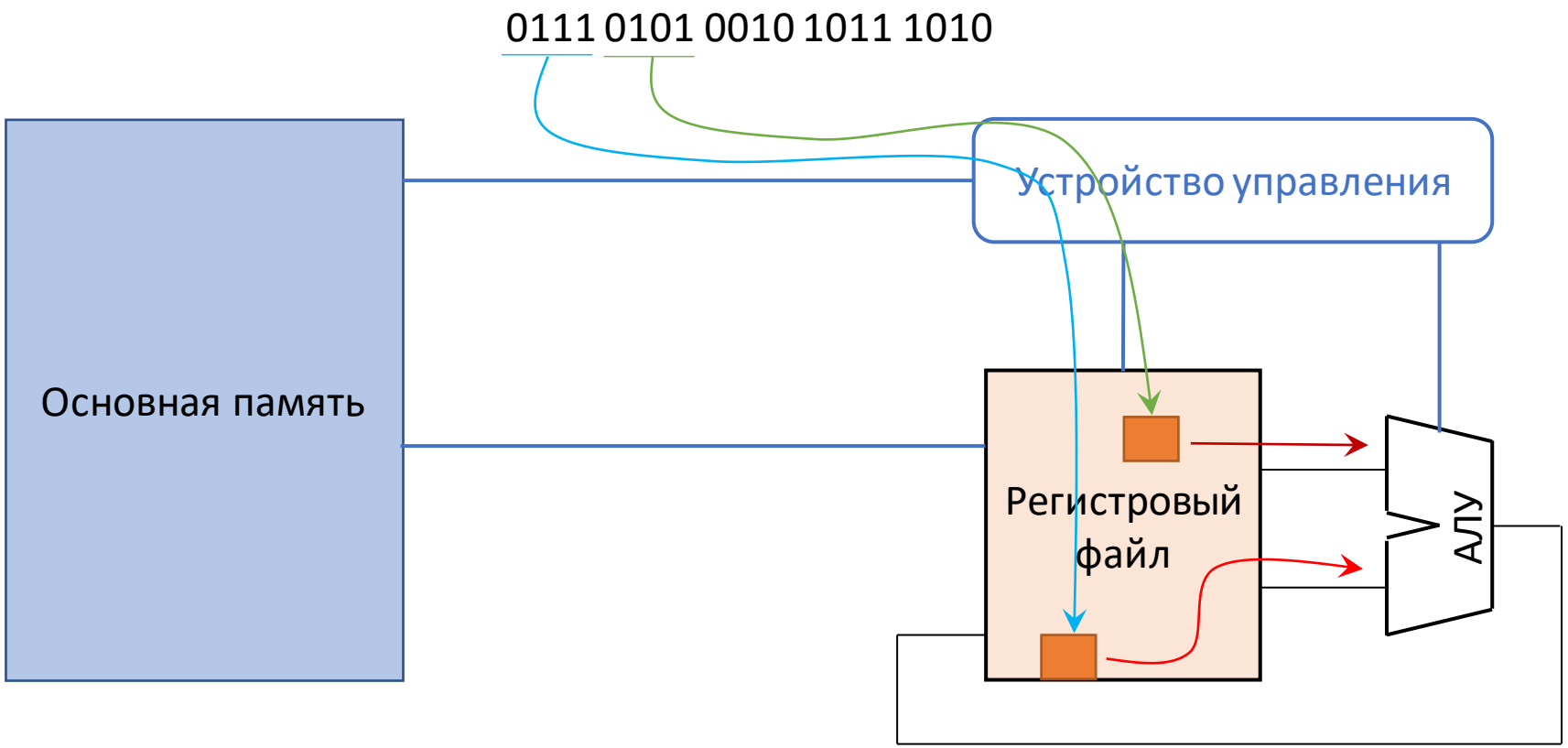


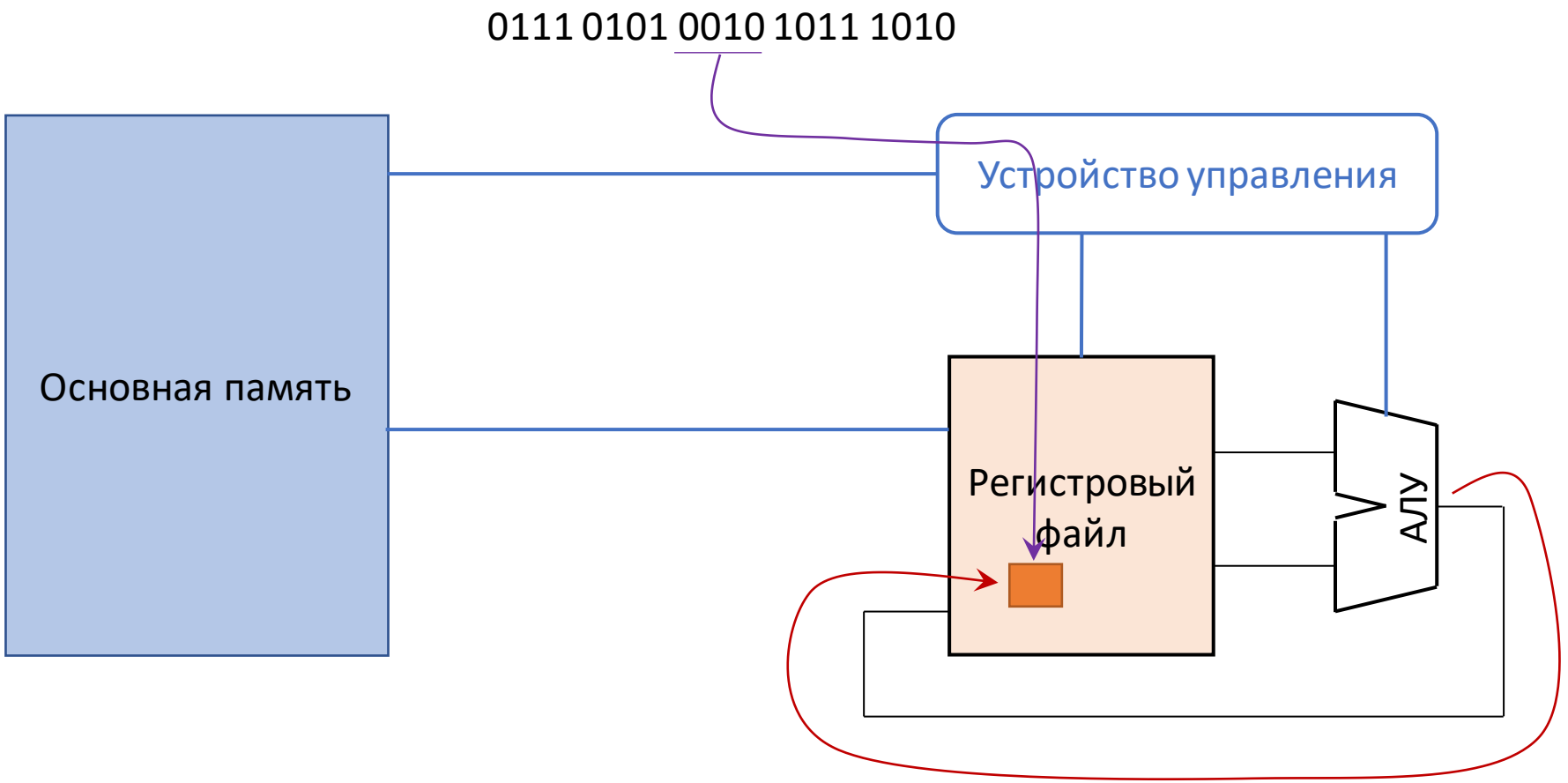

Основная память
Устройство управления
Регистровый файл
АЛУ

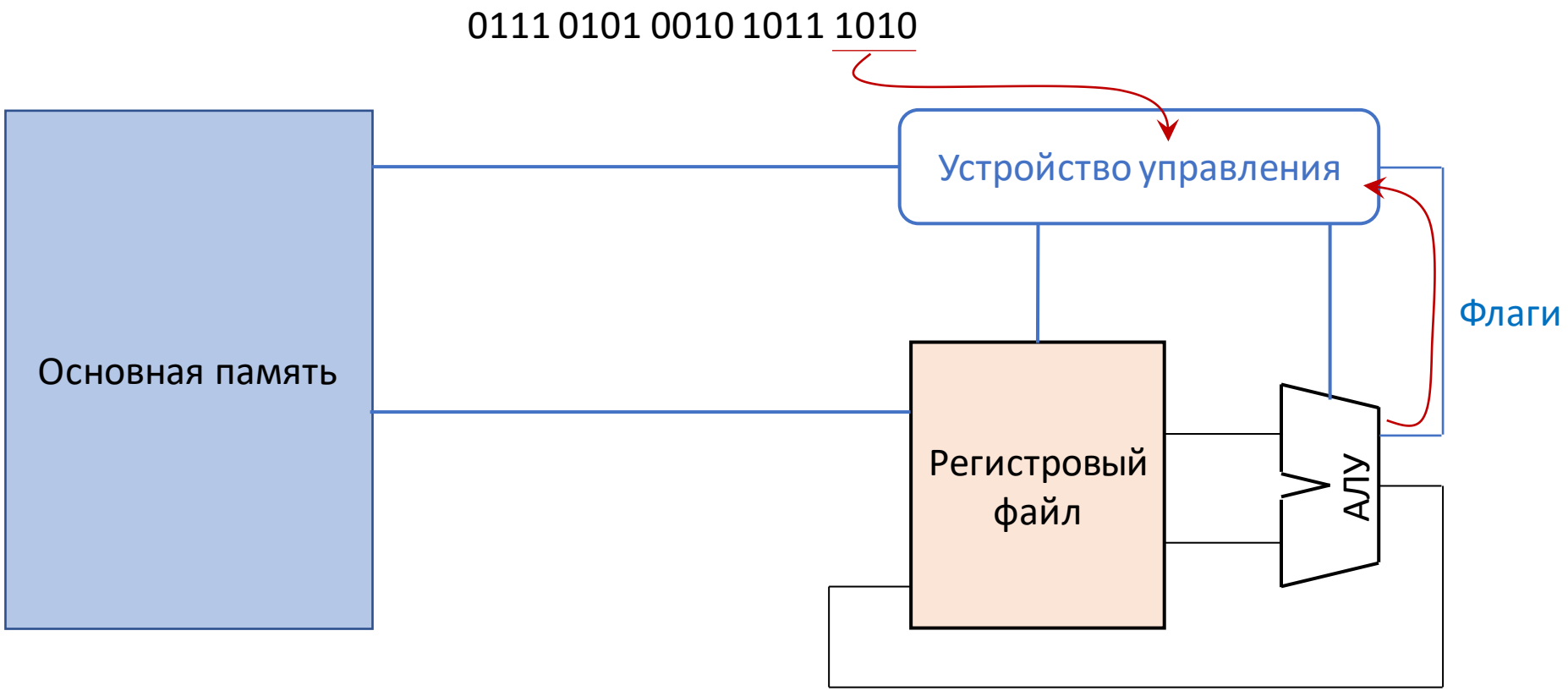

Что по адресу **PC**?
Основная память
Устройство управления
Регистровый файл
АЛУ

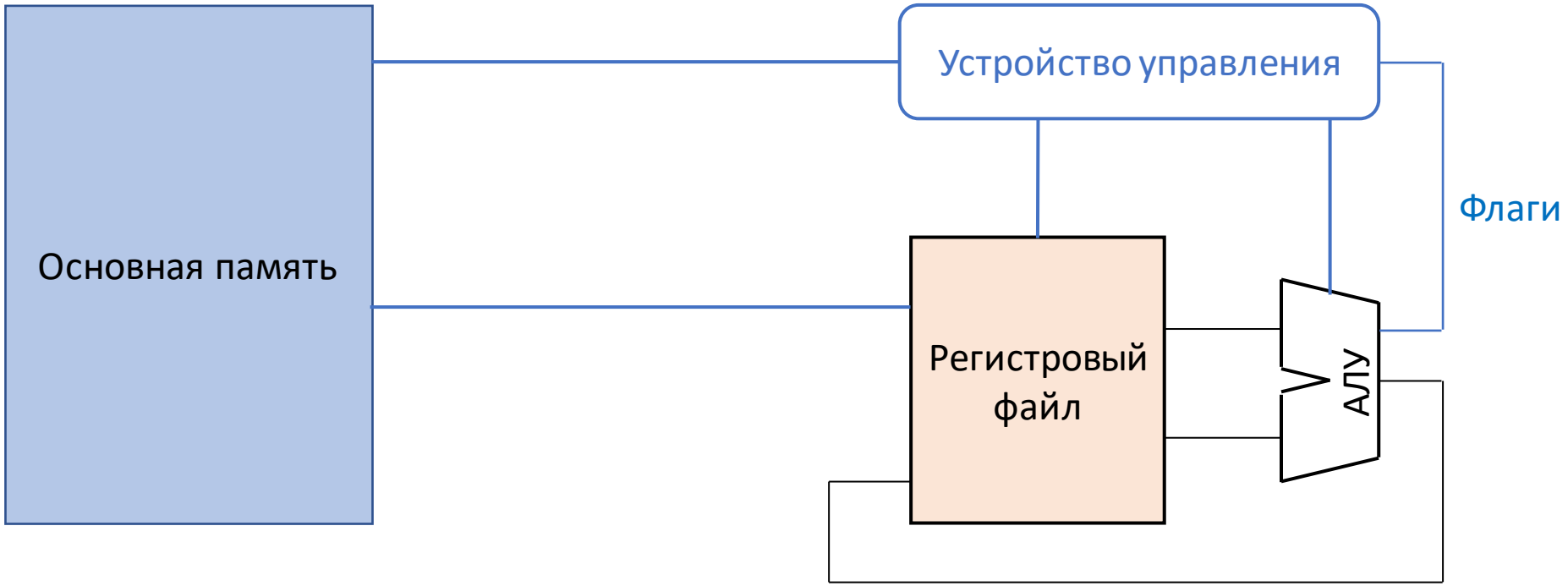

0111 0101 0010 1011 1010
Основная память
Устройство управления
Регистровый файл
АЛУ

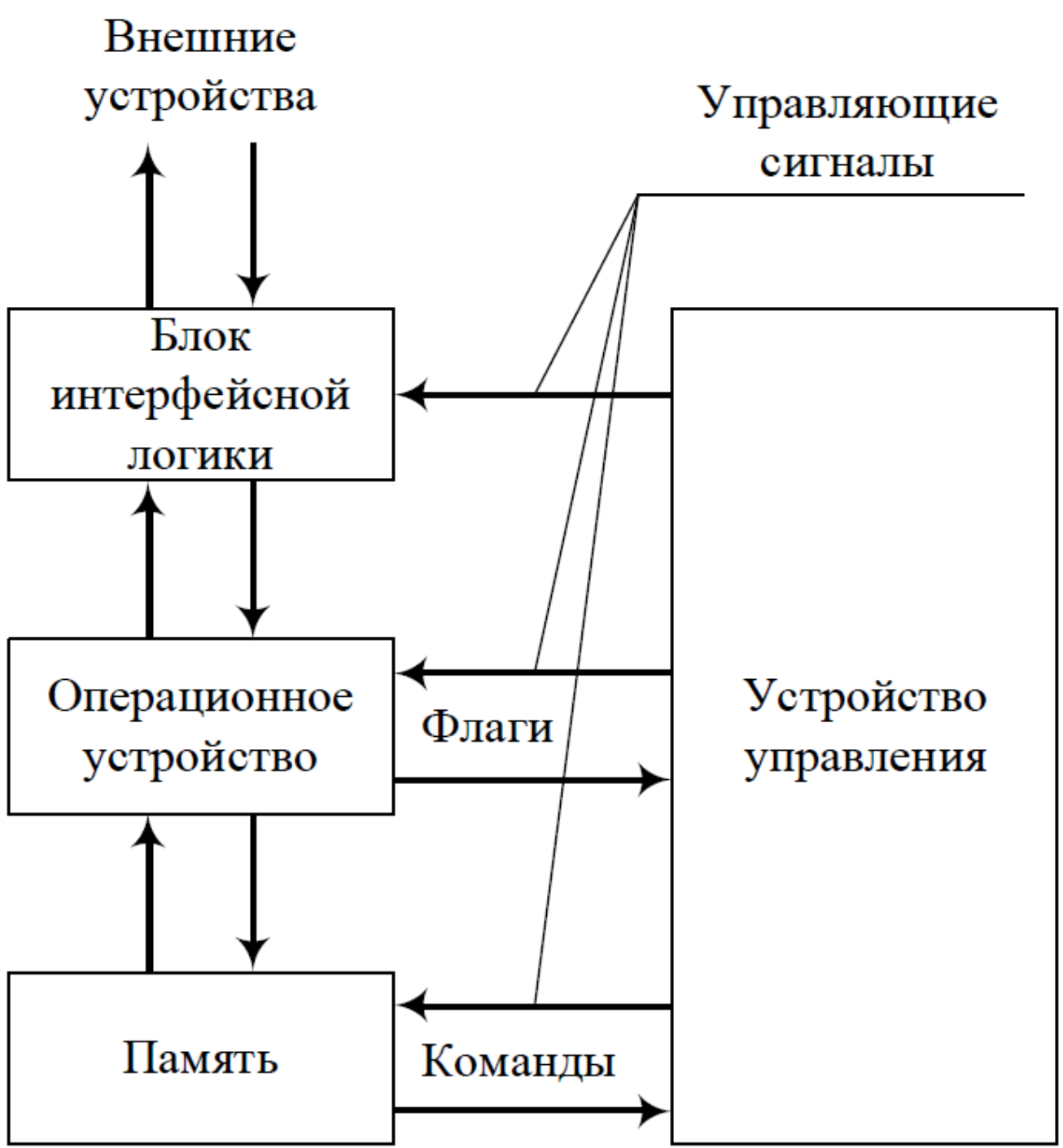

0111 0101 0010 1011 1010
Устройство управления
Основная память
Регистровый файл
АЛУ

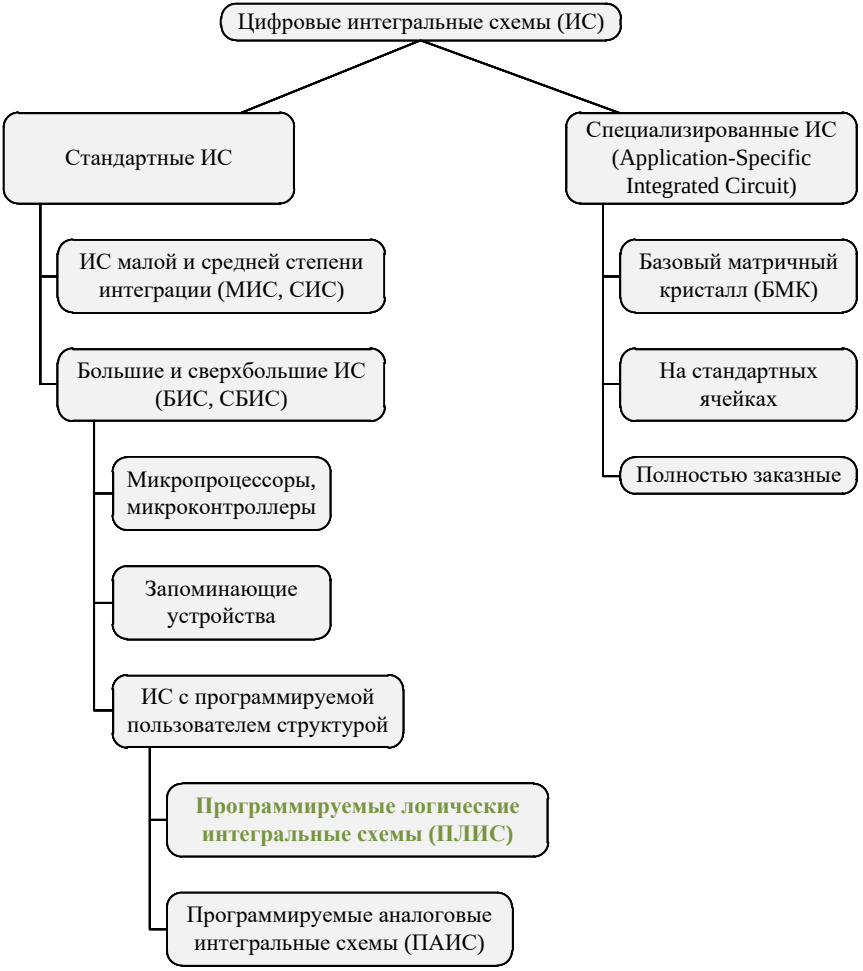

0111 0101 0010 1011 1010
Основная память
Устройство управления
Регистровый файл
АЛУ


0111 0101 0010 1011 1010
Основная память
Устройство управления
Регистровый файл
АЛУ

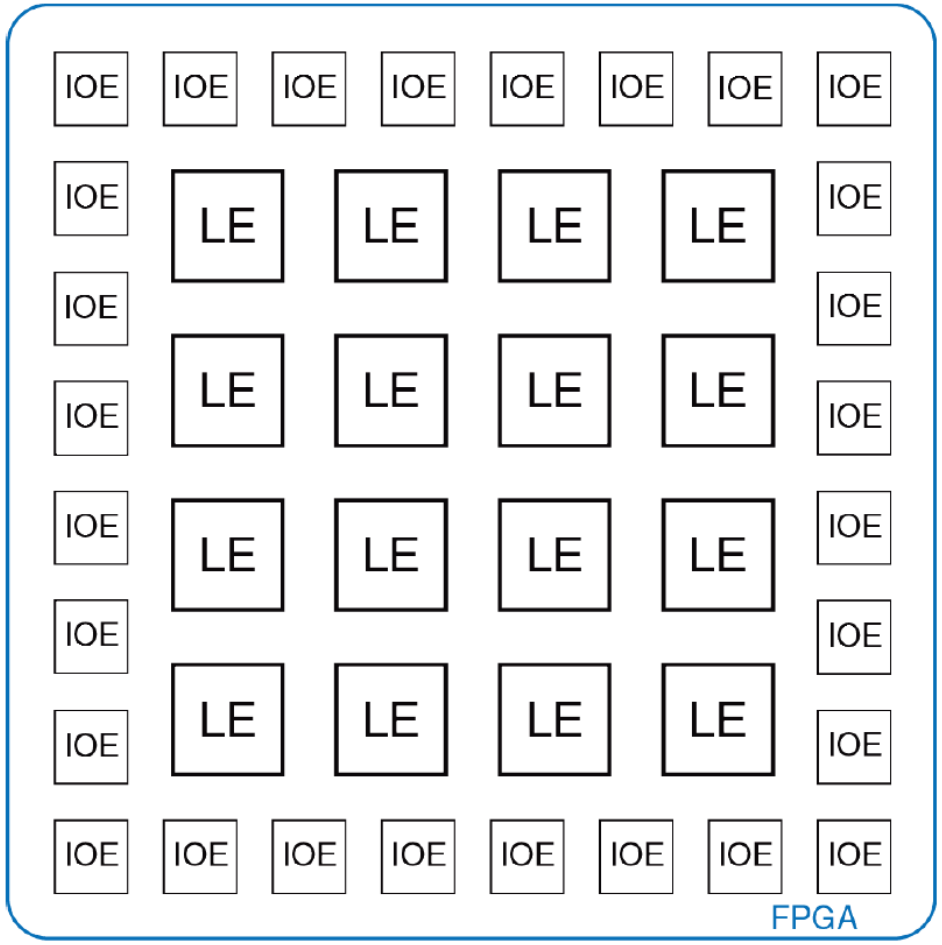

0111 0101 0010 1011 1010
Устройство управления
Основная память
Регистровый файл
АЛУ
Флаги


Основная память
Устройство управления
Флаги
Регистровый файл
АЛУ

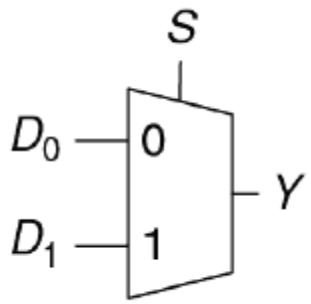

Внешние
устройства
Управляющие
сигналы
Блок
интерфейсной
логики
Операционное
устройство
Флаги
Устройство
управления
Память
Команды

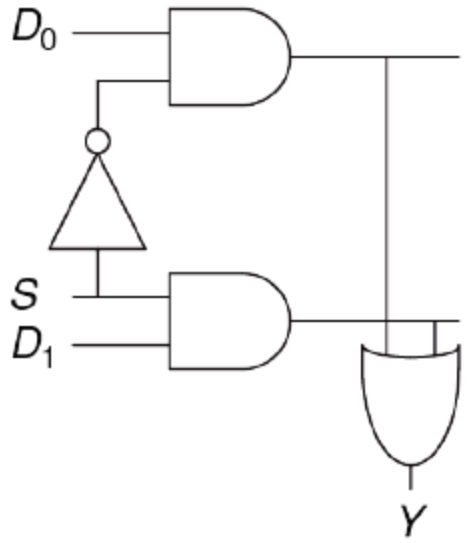

Цифровые интегральные схемы (ИС)
Стандартные ИС
Специализированные ИС (Application-Specific Integrated Circuit)
ИС малой и средней степени интеграции (МИС, СИС)
Большие и сверхбольшие ИС (БИС, СБИС)
Микропроцессоры, микроконтроллеры
Запоминающие устройства
ИС с программируемой пользователем структурой
Программируемые логические интегральные схемы (ПЛИС)
Программируемые аналоговые интегральные схемы (ПАИС)
Базовый матричный кристалл (БМК)
На стандартных ячейках
Полностью заказные

FPGA

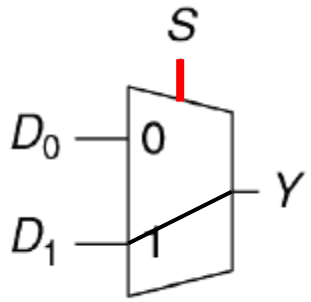


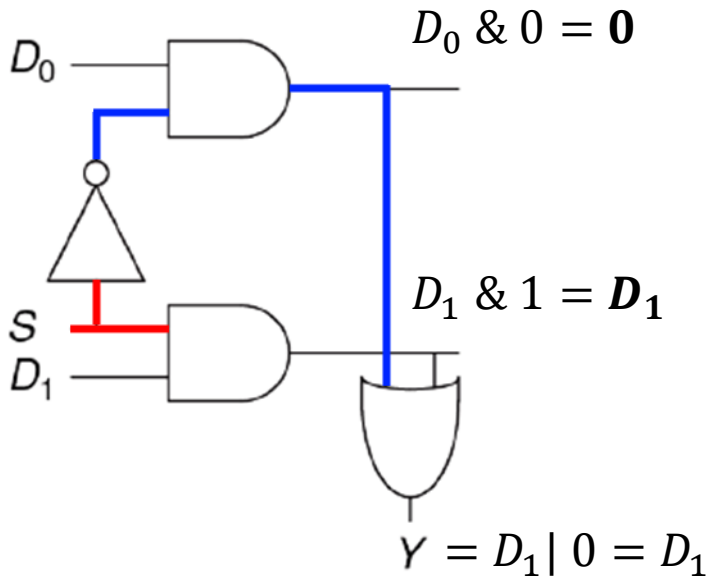
Мультиплексор



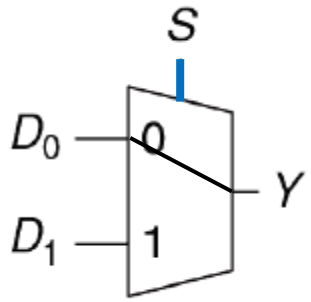


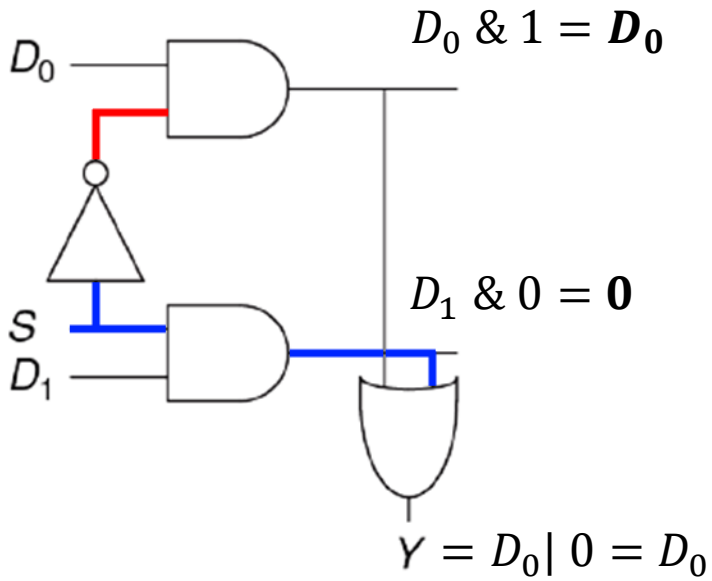
Мультиплексор



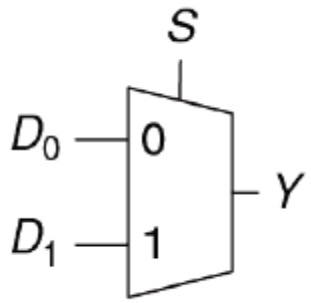


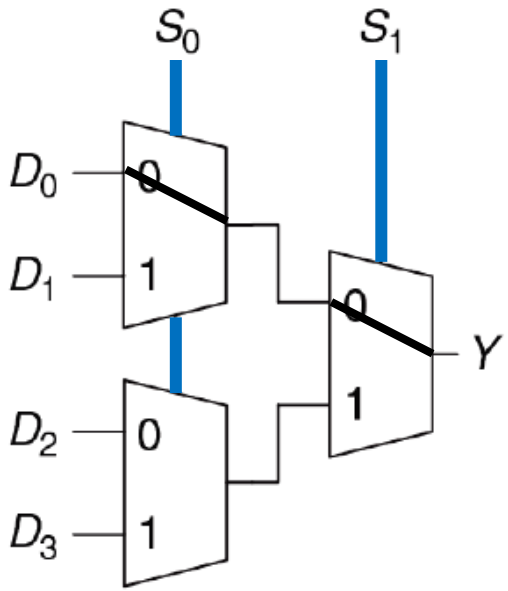
Мультиплексор



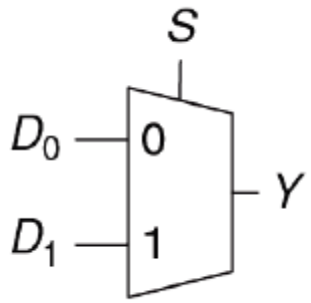


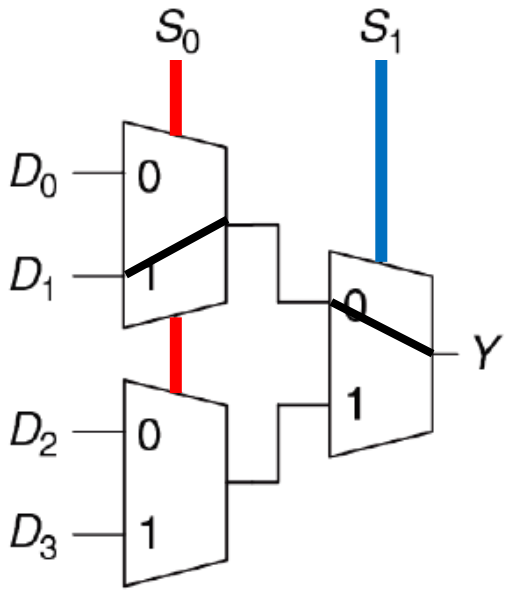
Мультиплексор



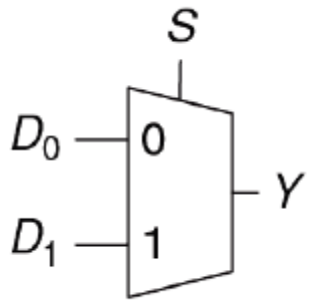


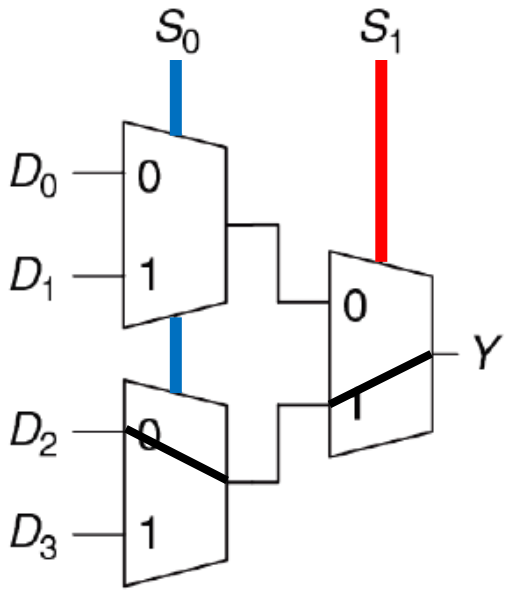
Мультиплексор



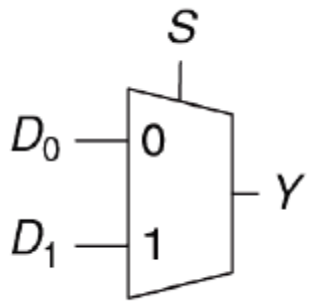


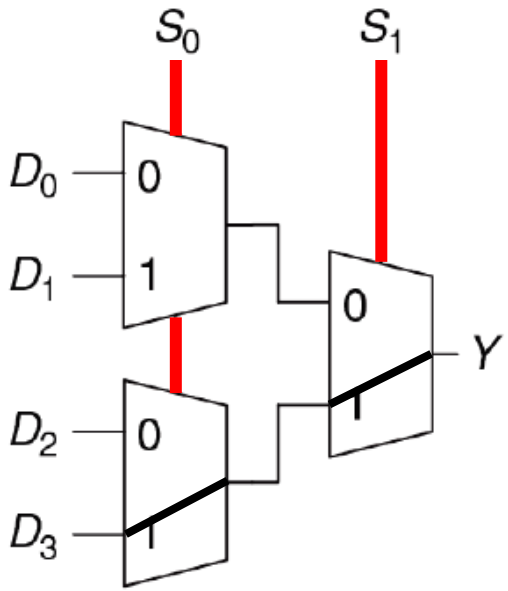
Мультиплексор



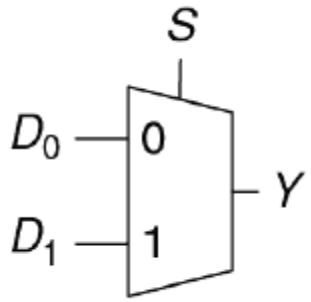


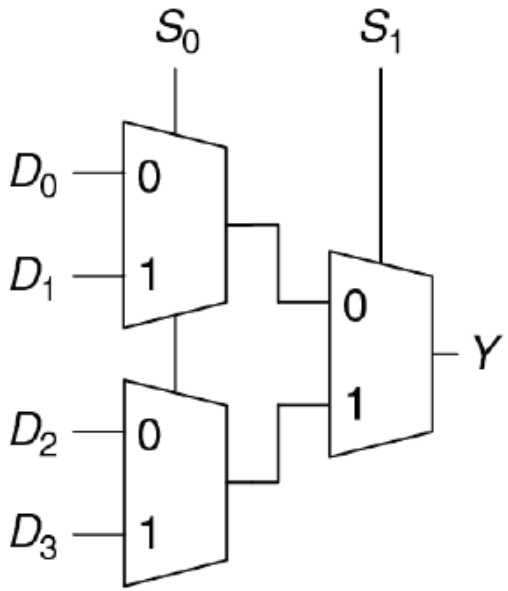
Мультиплексор

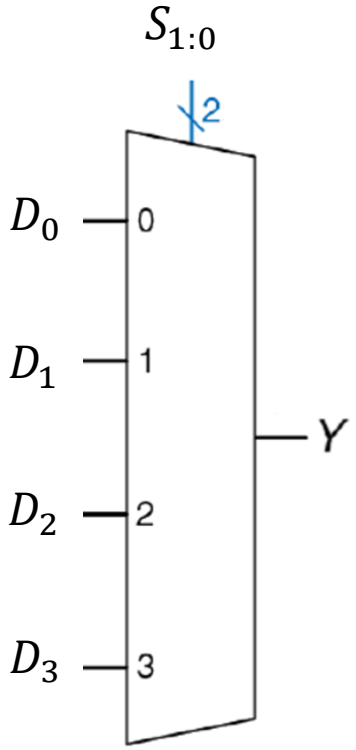




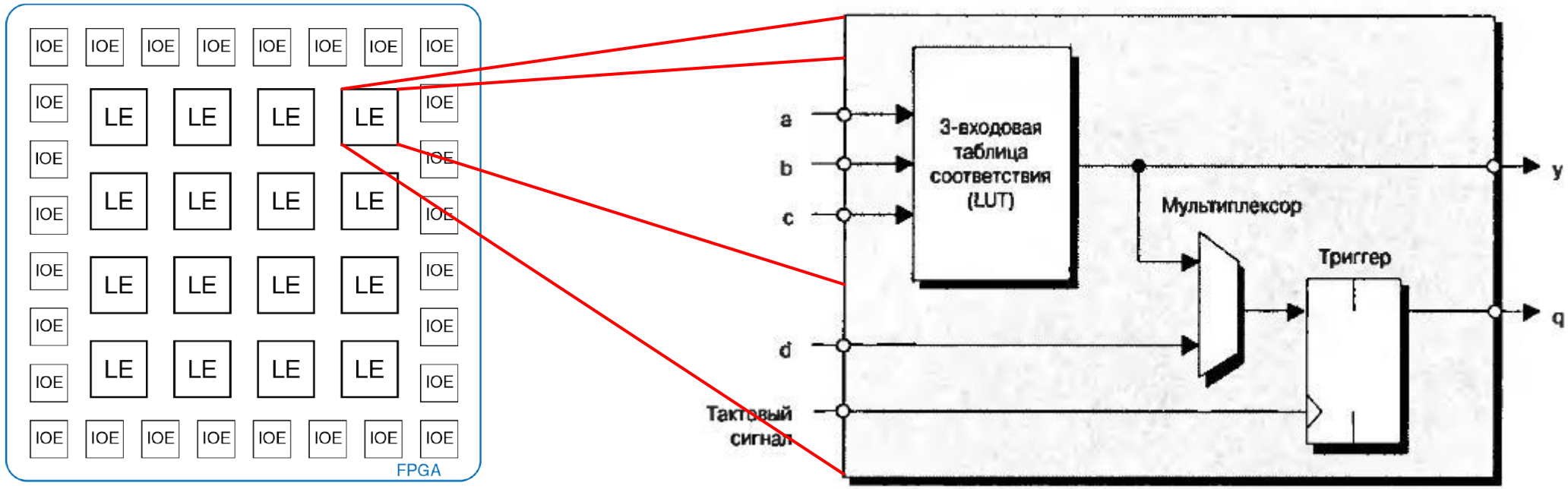
Мультиплексор



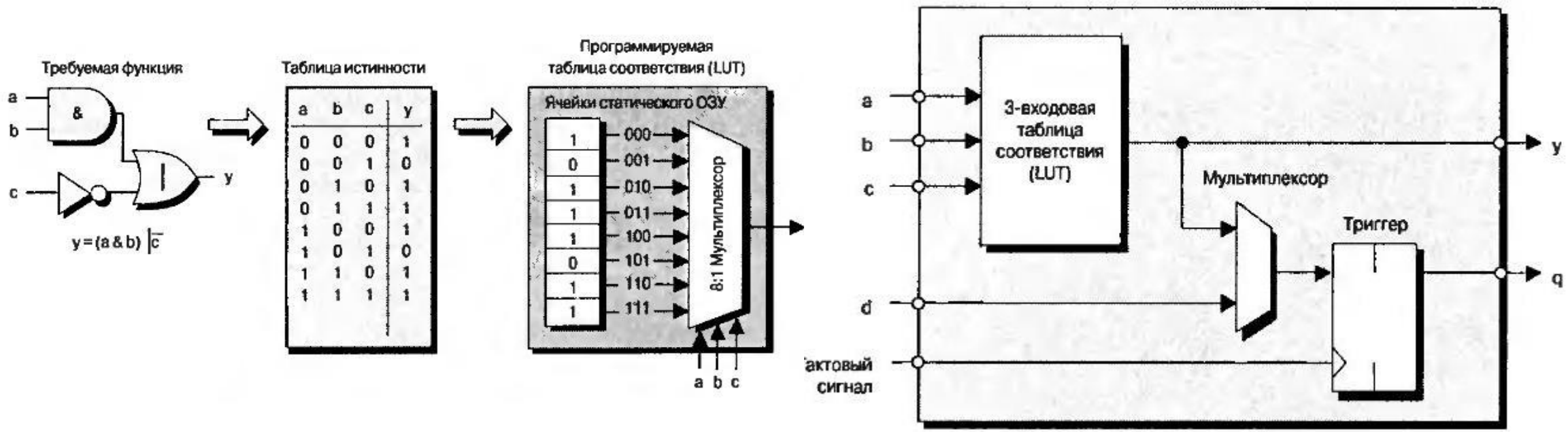


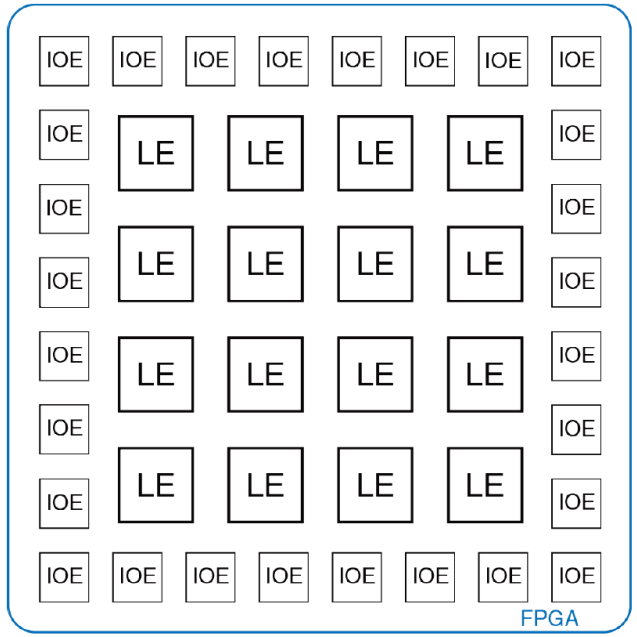


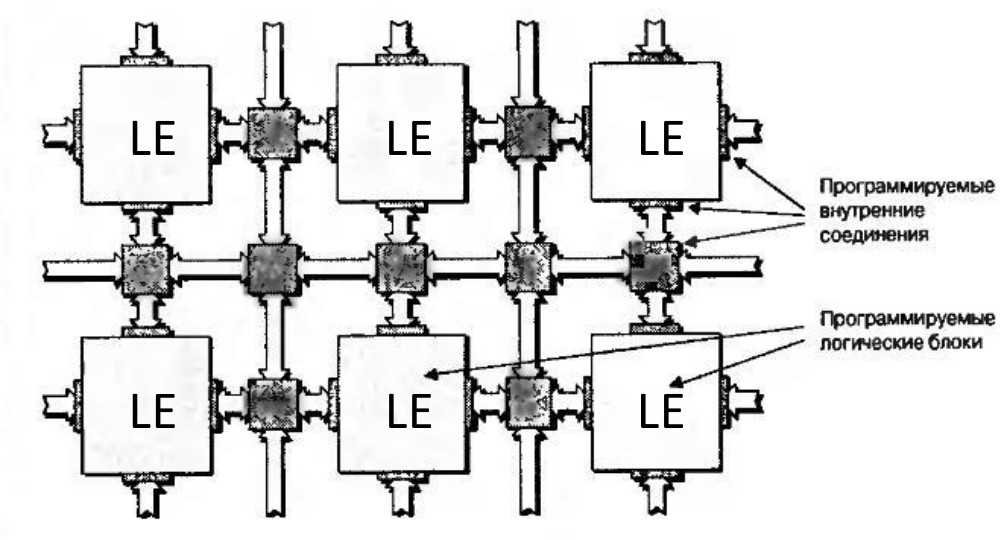
LE

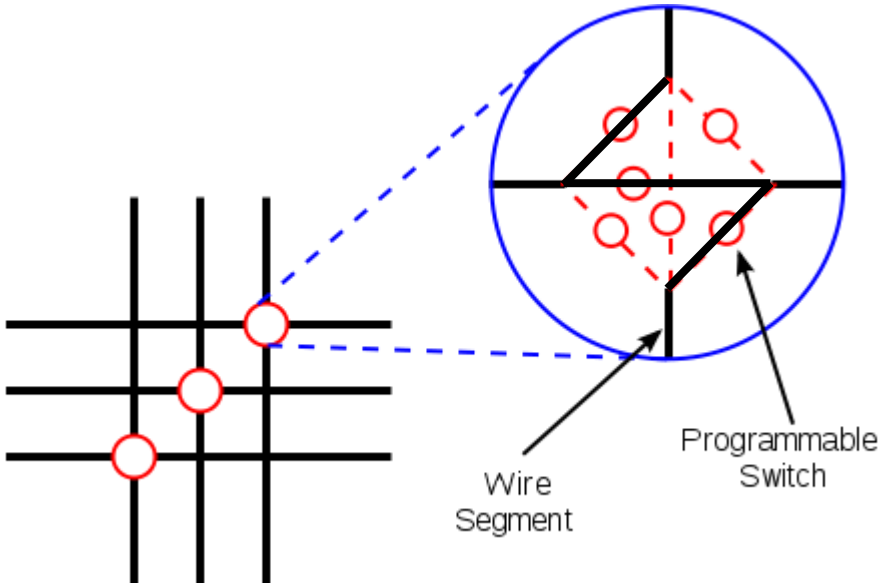

IOE
LE
FPGA
a
b
c
d
Тактовый сигнал
3-входовая таблица соответствия (LUT)
Мультиплексор
Триггер
y
q

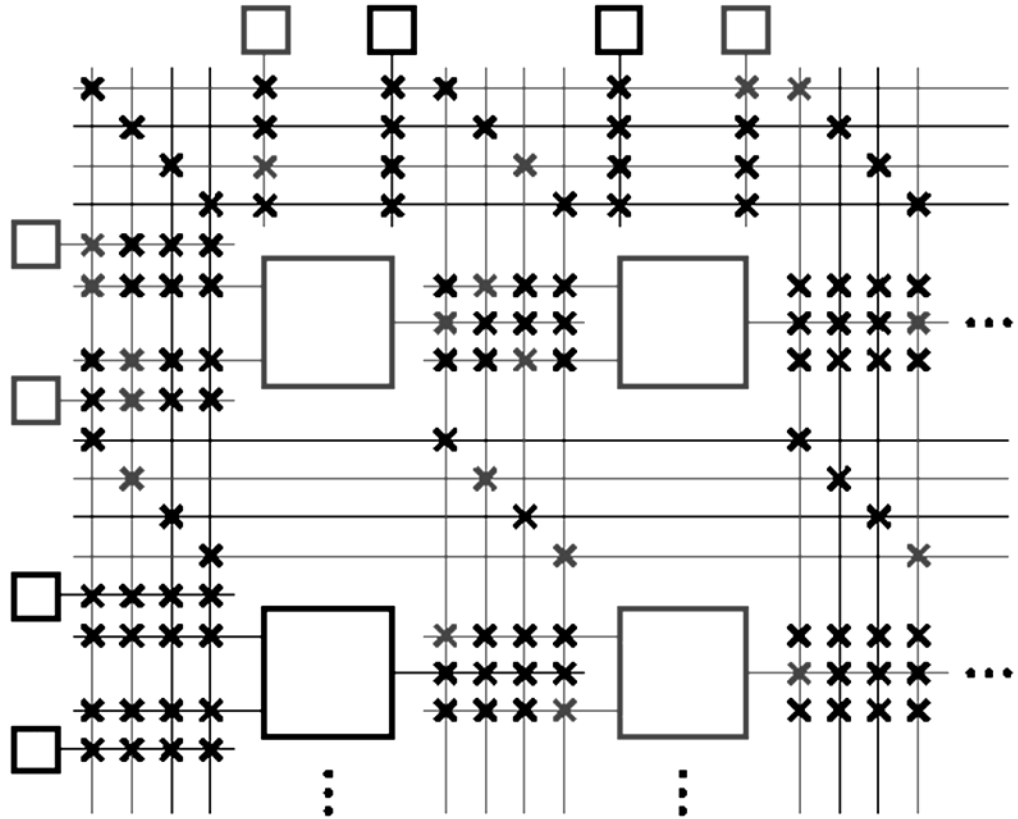
LE

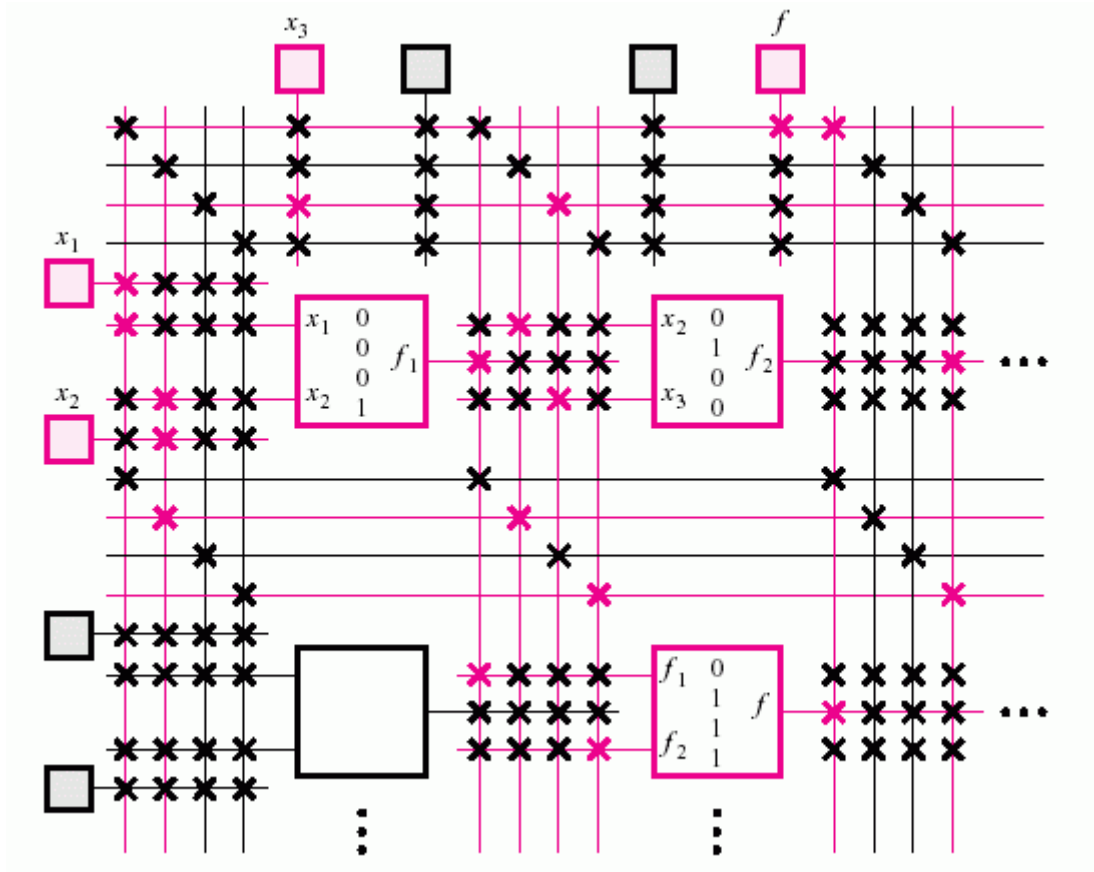

Требуемая функция
a
b
c
&
y
y = (a & b) |c
Таблица истинности
a b c y
0 0 0 1
0 0 1 0
0 1 0 1
0 1 1 1
1 0 0 1
1 0 1 0
1 1 0 1
1 1 1 1
Программируемая таблица соответствия (LUT)
Ячейки статического ОЗУ
1
0
1
1
1
0
1
1
000
001
010
011
100
101
110
111
8:1 Мультиплексор
a b c
a
b
c
d
Тактовый сигнал
3-входовая таблица соответствия (LUT)
Мультиплексор
Триггер
y
q

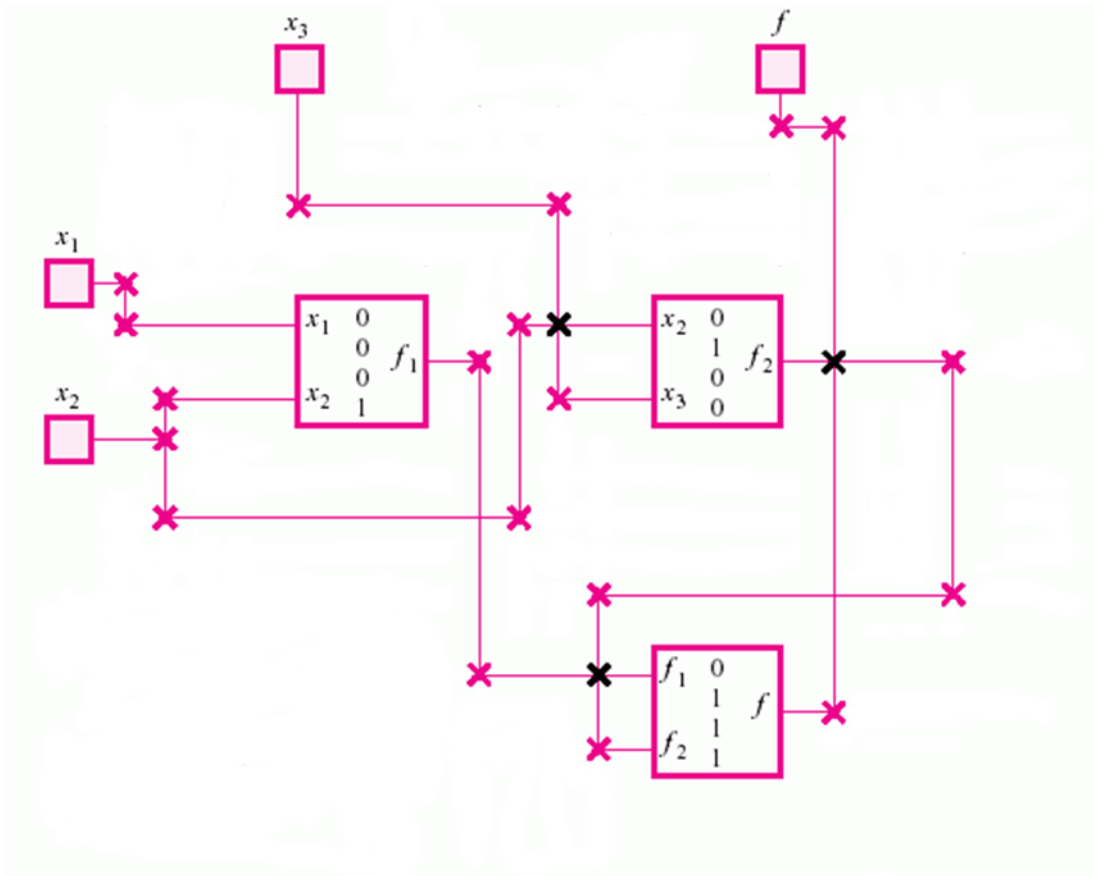

IOE
LE
FPGA

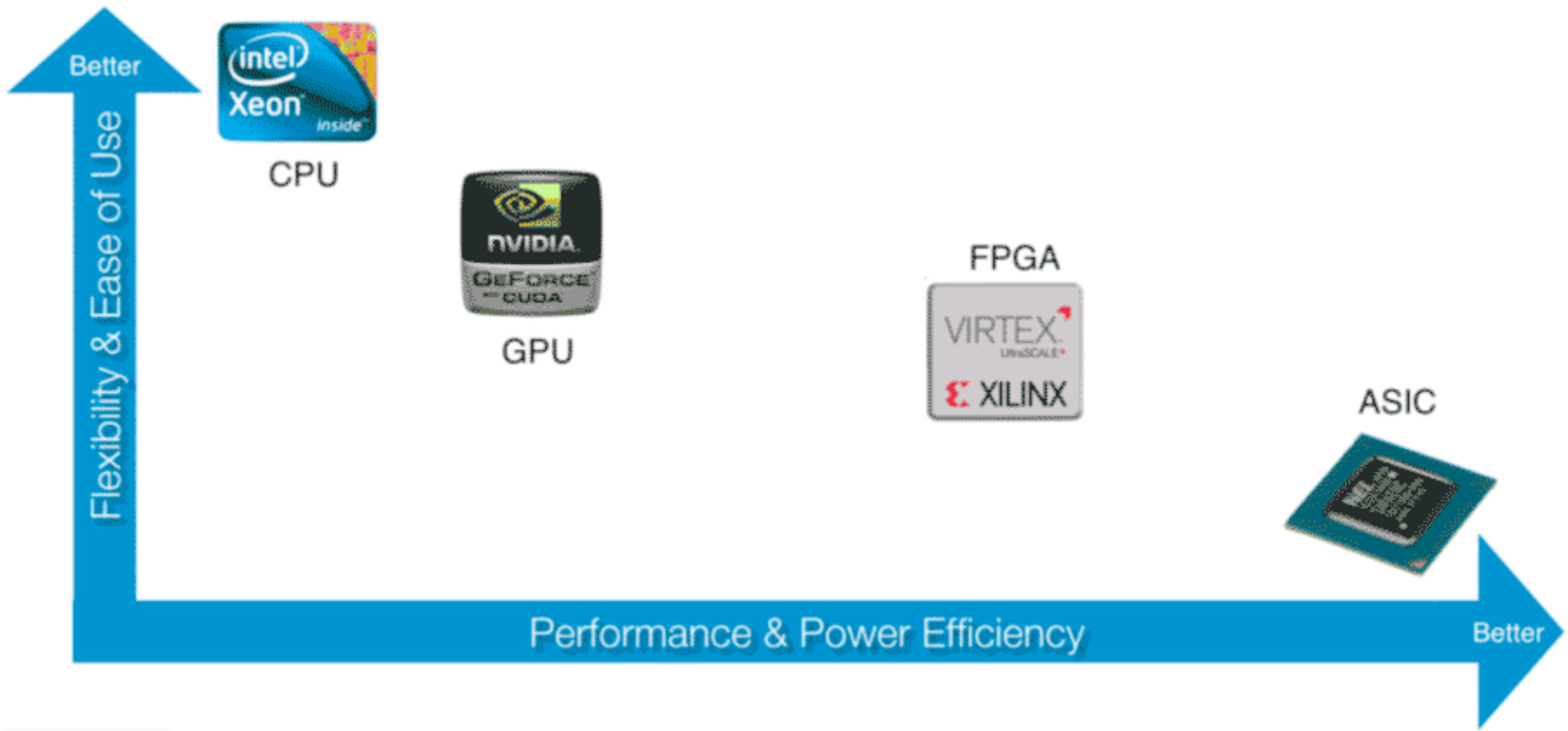

LE
Программируемые внутренние соединения
Программируемые логические блоки


Wire Segment
Programmable Switch



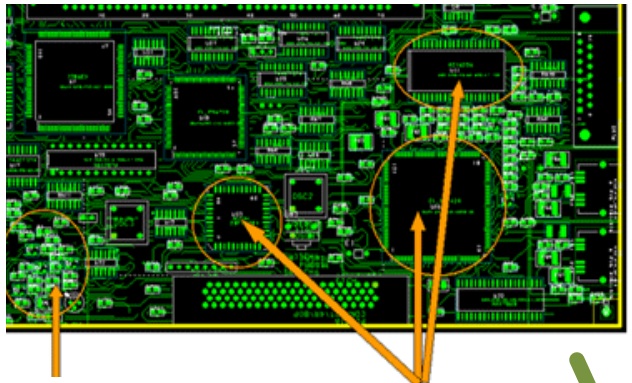

x_3
f
x_1
x_2
x_1 0 0 0 x_2 1 f_1
x_2 0 1 0 x_3 0 f_2
f_1 0 1 1 f_2 1 f


x_3
f
x_1
x_2
x_1 0 0 0 f_1 x_2 1
x_2 0 1 0 f_2 x_3 0
f_1 0 1 1 f f_2 1


Better
Flexibility & Ease of Use
intel
Xeon
inside
CPU
NVIDIA
GeForce
CUDA
GPU
FPGA
VIRTEX
UltraSCALE+
XILINX
ASIC
Performance & Power Efficiency
Better

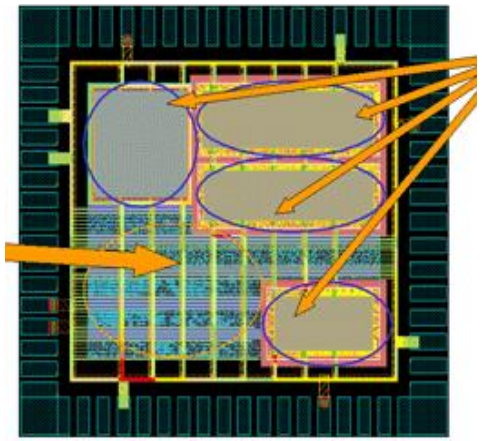
Преимущества системы на кристалле:

- повышение степени интеграции приводит к миниатюризации устройств, как правило, система состоит из одной СБИС и ограниченного набора дискретных компонент, которые по технологическим причинам не могут быть интегрированы внутрь ИС.
- снижение потребляемой мощности.
- повышение надежности. Объединение нескольких компонент в одной ИС позволяет существенно уменьшить число паяных соединений.



Дискретные элементы

Электронные компоненты

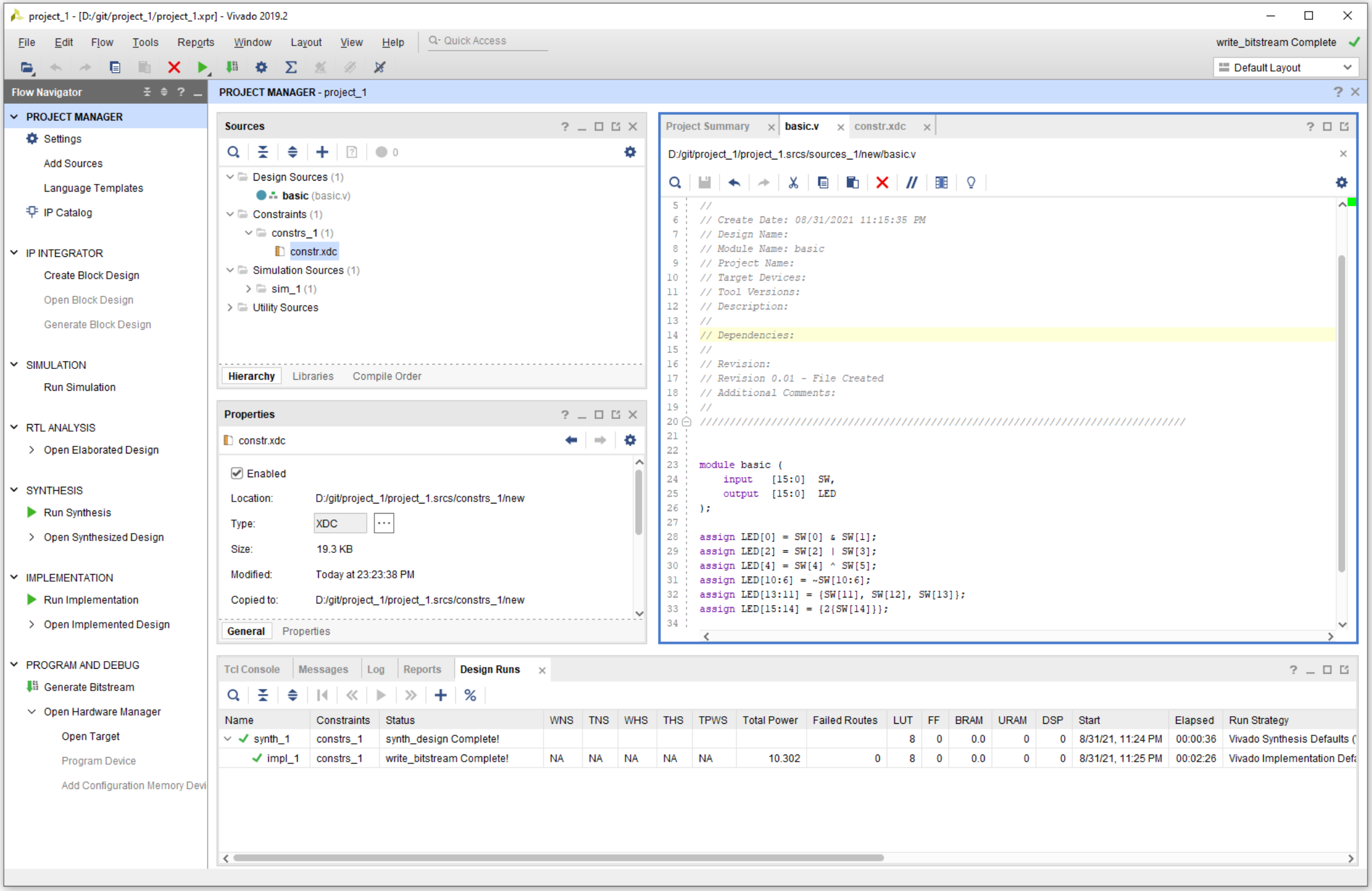


Жесткая логика

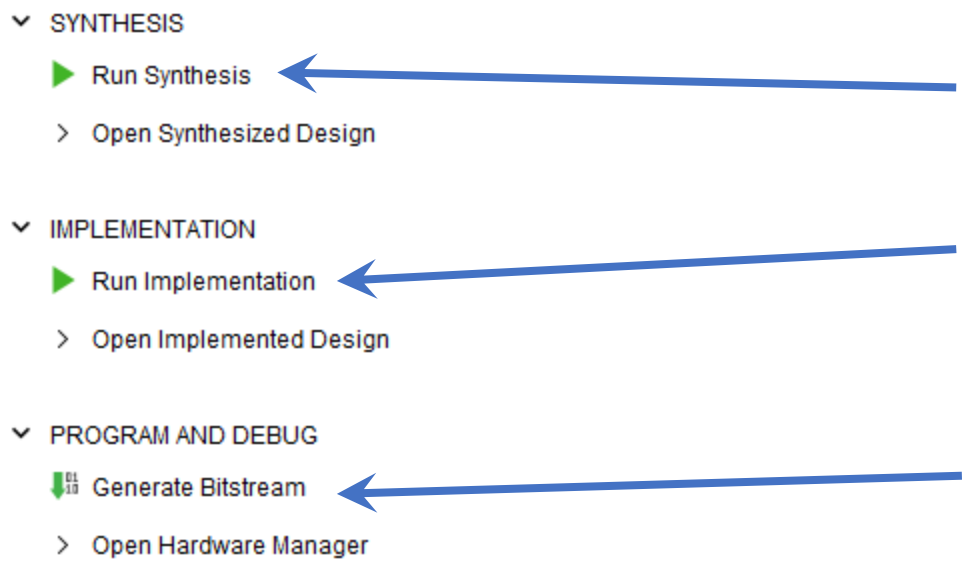
IP-Блоки (Intellectual Property):

- микропроцессор,
- модули ЦОС,
- ОЗУ,
- ПЗУ,
- интерфейсные модули.


VIVADO
HLx Editions
2019.2
XILINX
Copyright 1986-2019 Xilinx, Inc.
All Rights Reserved.


project_1 - [D:/git/project_1/project_1.xpr] - Vivado 2019.2
File Edit Flow Tools Reports Window Layout View Help
Quick Access
write_bitstream Complete
Default Layout
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager
Open Target
Program Device
Add Configuration Memory Devi
PROJECT MANAGER - project_1
Sources
Design Sources (1)
basic (basic.v)
Constraints (1)
constrs_1 (1)
constr.xdc
Simulation Sources (1)
sim_1 (1)
Utility Sources
Hierarchy Libraries Compile Order
Properties
constr.xdc
Enabled
Location: D:/git/project_1/project_1.srcs/constrs_1/new
Type: XDC
Size: 19.3 KB
Modified: Today at 23:23:38 PM
Copied to: D:/git/project_1/project_1.srcs/constrs_1/new
General Properties
Project Summary
basic.v
constr.xdc
D:/git/project_1/project_1.srcs/sources_1/new/basic.v
5 //
6 // Create Date: 08/31/2021 11:15:35 PM
7 // Design Name:
8 // Module Name: basic
9 // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //
21
22
23 module basic (
24 input [15:0] SW,
25 output [15:0] LED
26);
27
28 assign LED[0] = SW[0] & SW[1];
29 assign LED[2] = SW[2] | SW[3];
30 assign LED[4] = SW[4] ^ SW[5];
31 assign LED[10:6] = ~SW[10:6];
32 assign LED[13:11] = {SW[11], SW[12], SW[13]};
33 assign LED[15:14] = {2{SW[14]}};
34
Tcl Console Messages Log Reports Design Runs
Name Constraints Status WNS TNS WHS THS TPWS Total Power Failed Routes LUT FF BRAM URAM DSP Start Elapsed Run Strategy
synth_1 constrs_1 synth_design Complete! 8 0 0.0 0 0 8/31/21, 11:24 PM 00:00:36 Vivado Synthesis Defaults (
impl_1 constrs_1 write_bitstream Complete! NA NA NA NA NA 10.302 0 8 0 0.0 0 0 8/31/21, 11:25 PM 00:02:26 Vivado Implementation Defa

Процесс компиляции



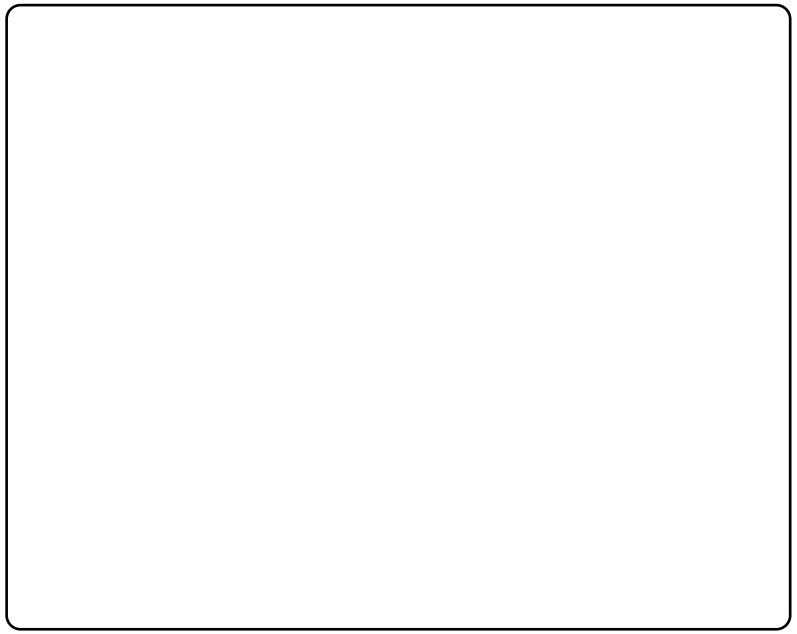
Синтез и верификация Verilog кода

Имплементация (размещение на кристалле)

Bitstream (генерация прошивки)

Verilog HDL

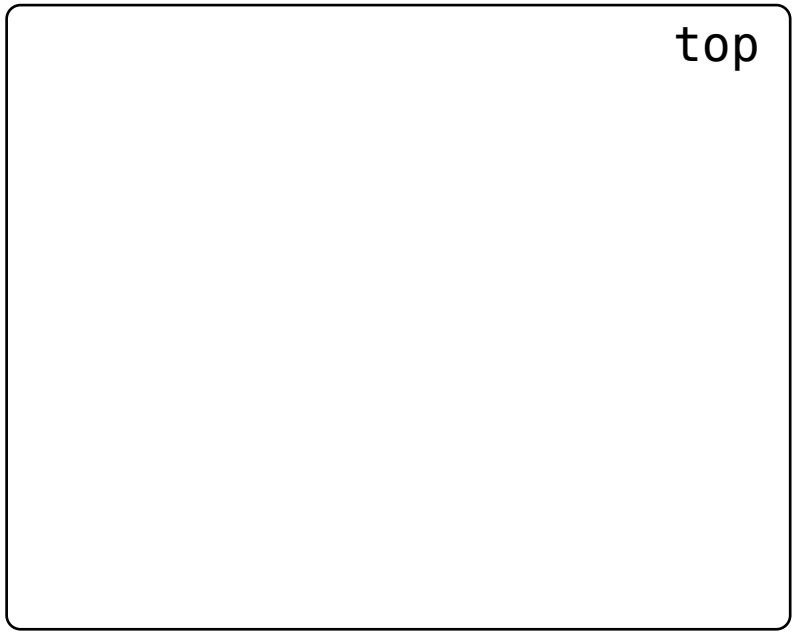
```
module
```



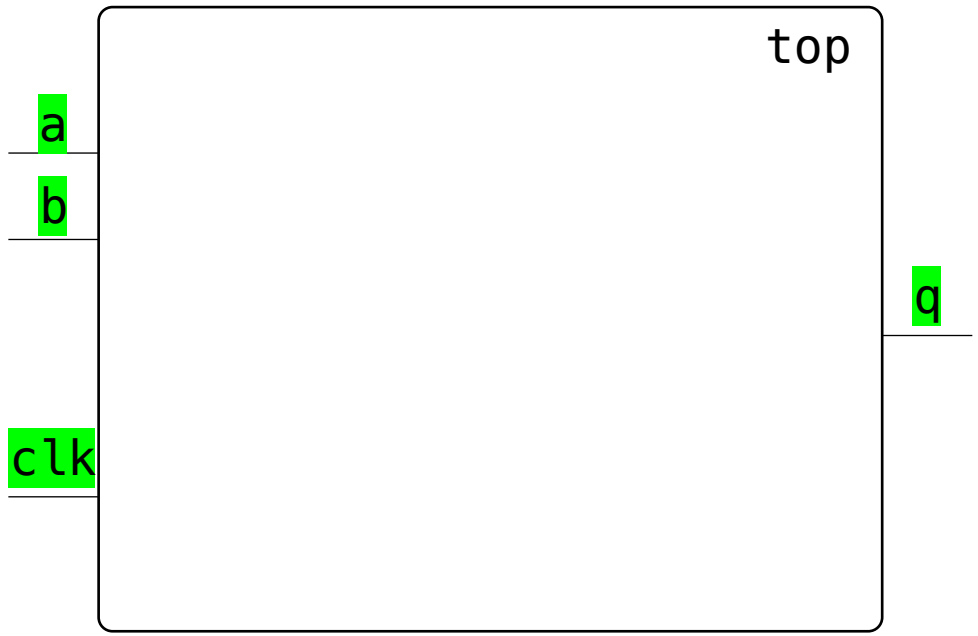
```
endmodule
```

```
module top

endmodule
```

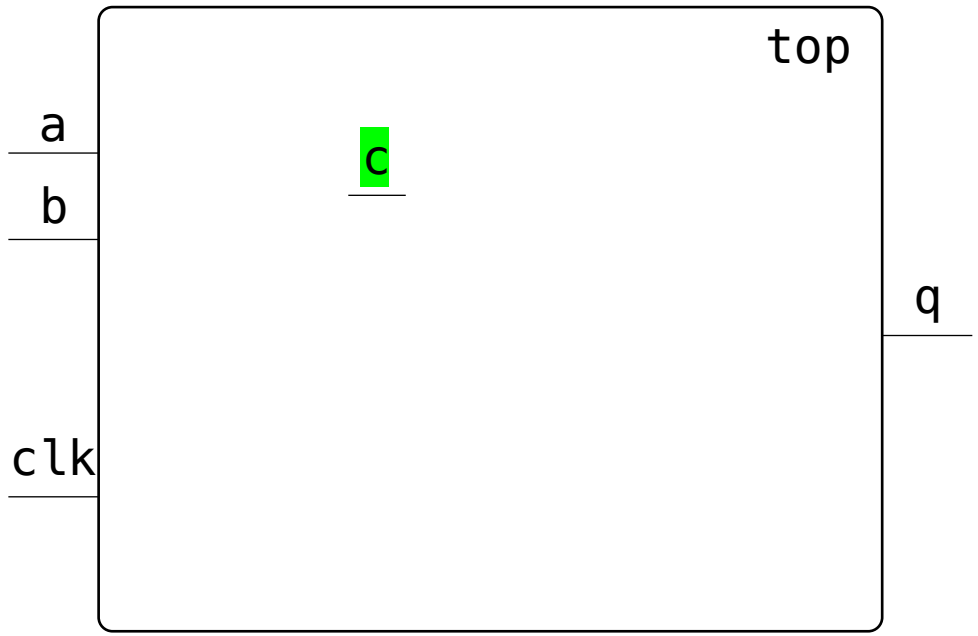
```
module top ();

endmodule
```



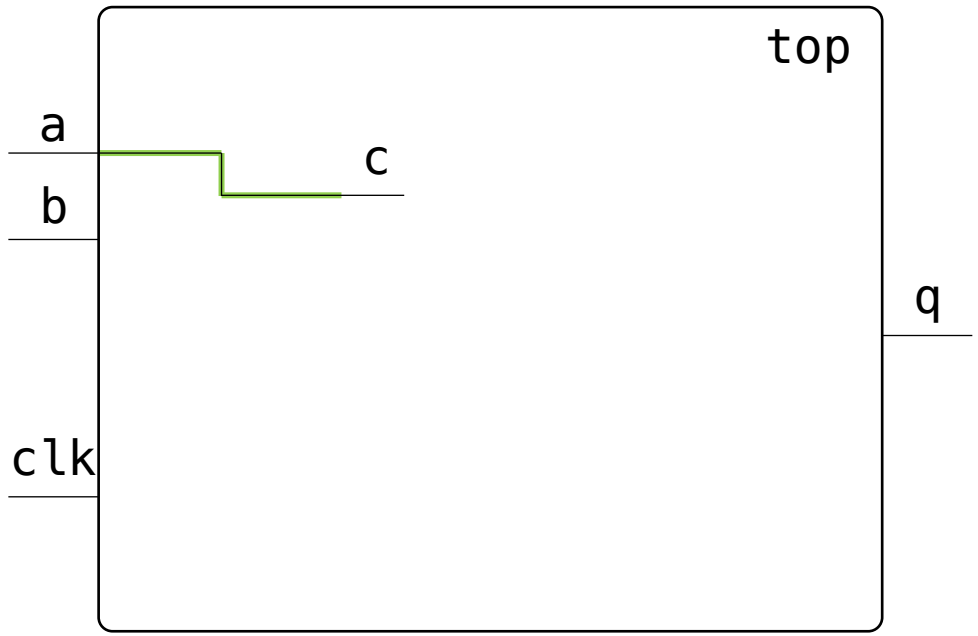
```
module top (
        input           a,
        input           b,
        input           clk,
        output          q
);




endmodule
```



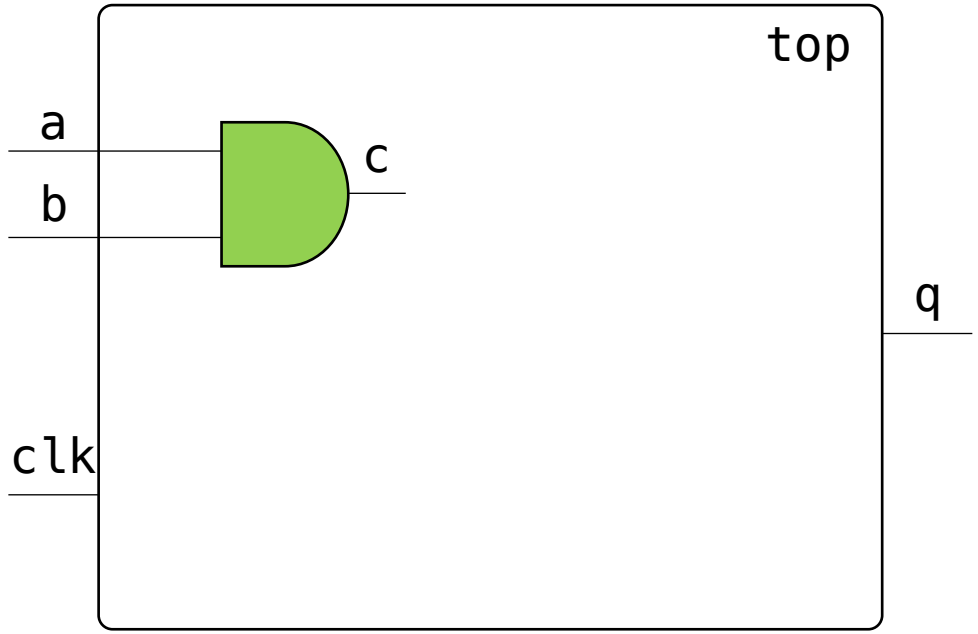
```
module top (
        input           a,
        input           b,
        input           clk,
        output          q
);

wire c;



endmodule
```



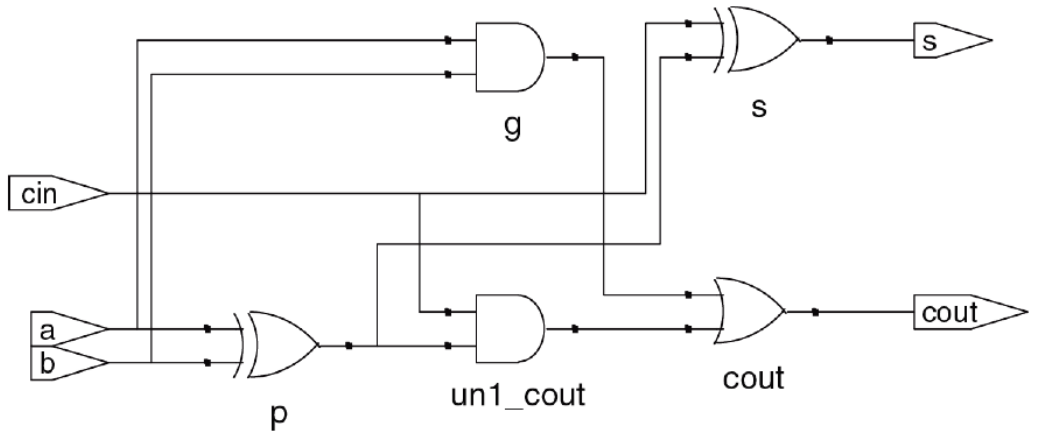
```
module top (
        input           a,
        input           b,
        input           clk,
        output          q
);

wire c;

assign c = a;



endmodule
```



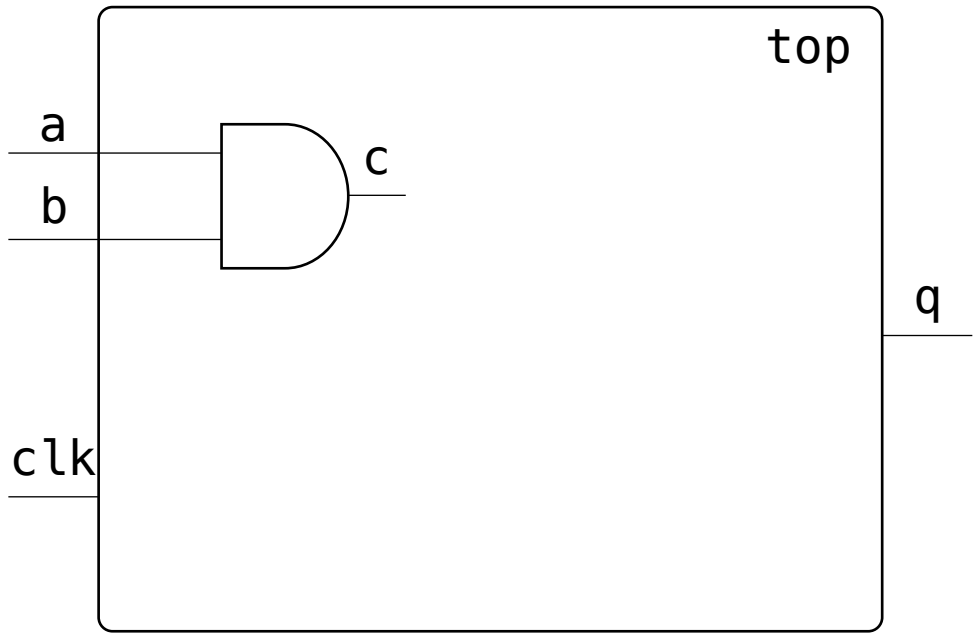
```
module top (
        input           a,
        input           b,
        input           clk,
        output          q
);

wire c;

assign c = a & b;



endmodule
```



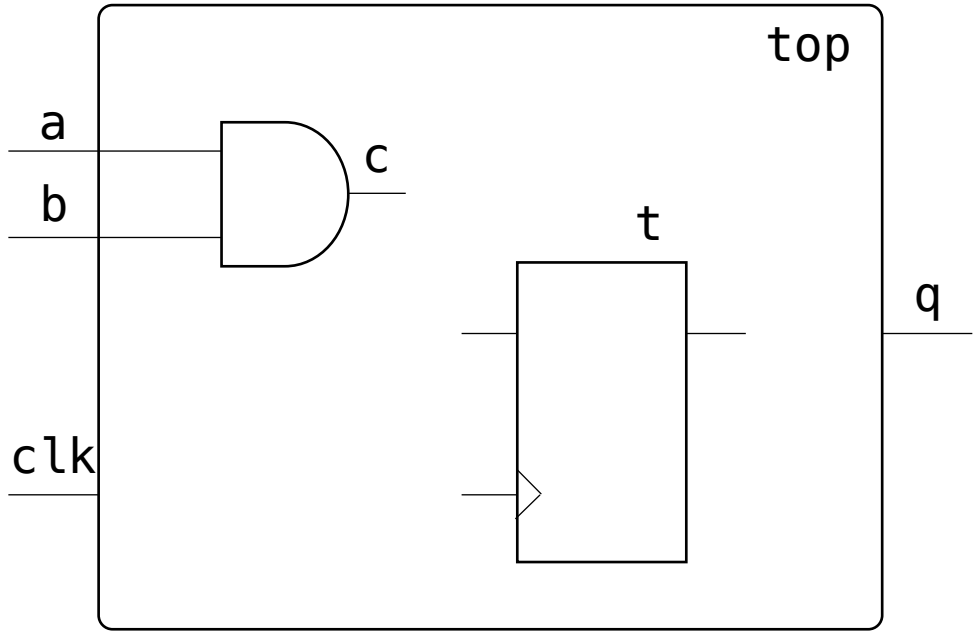
```
module fulladder (a, b, cin, s, cout);

       input  a, b, cin;
       output s, cout;

wire p, g;

assign p = a ^ b;
assign g = a & b;
assign s = p ^ cin;
assign cout = g |(p & cin);

endmodule
```



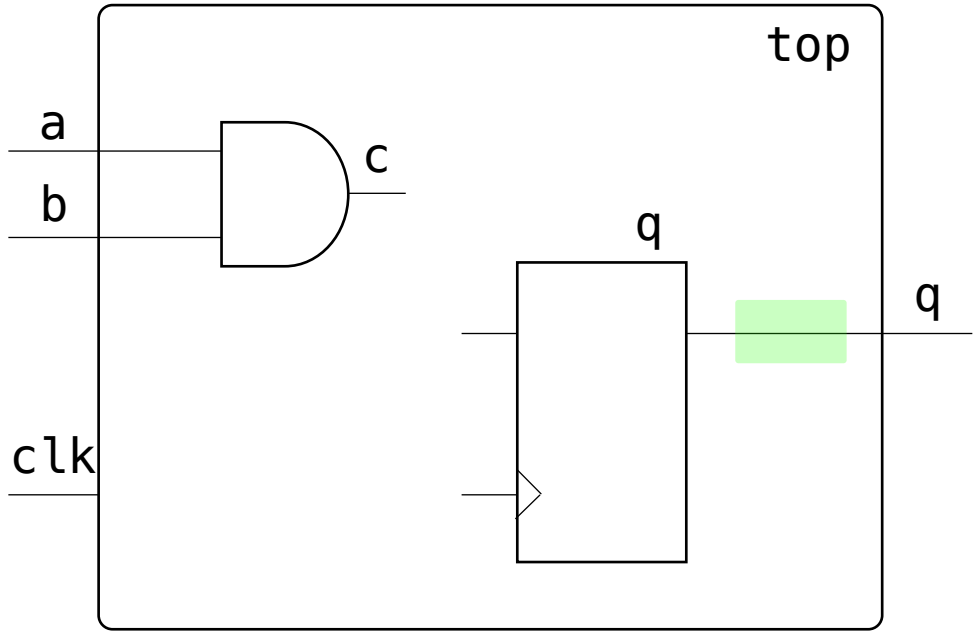
```
module top (
        input           a,
        input           b,
        input           clk,
        output          q
);

wire c;

assign c = a & b;



endmodule
```



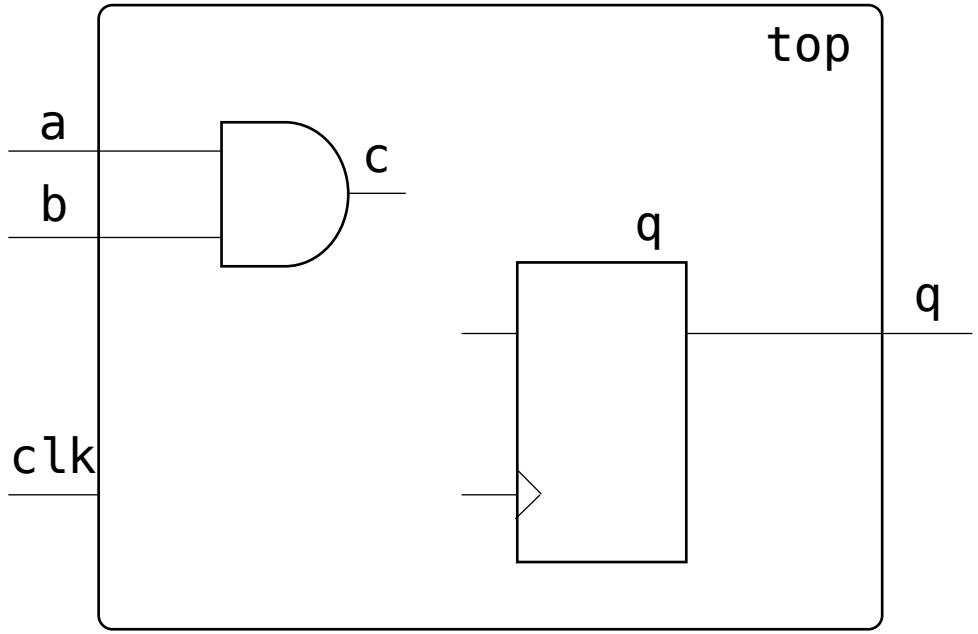
```
module top (
        input           a,
        input           b,
        input           clk,
        output          q
);

wire c;
reg  t;
assign c = a & b;



endmodule
```



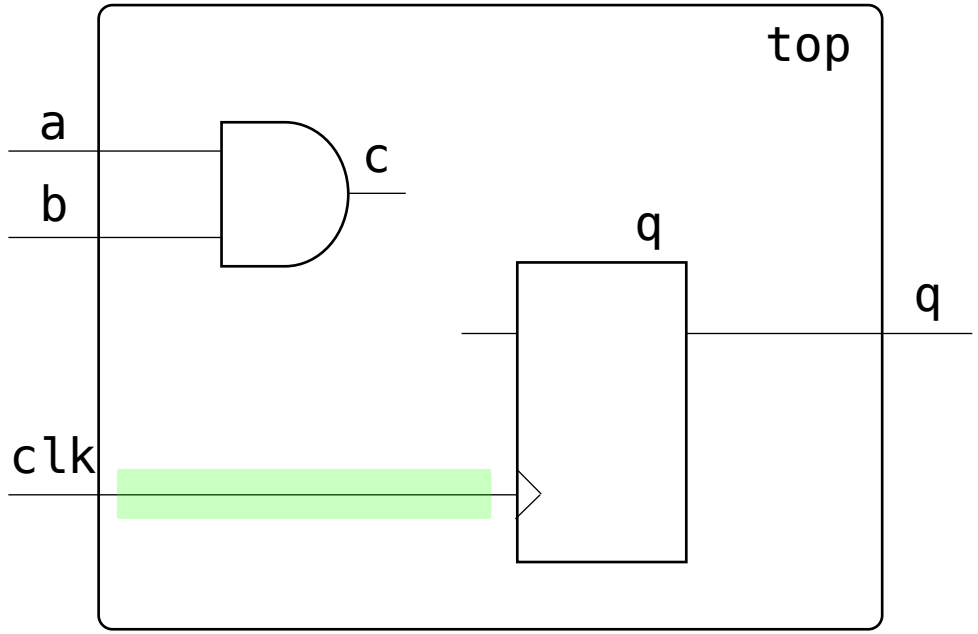
```
module top (
        input           a,
        input           b,
        input           clk,
        output  reg     q
);

wire c;

assign c = a & b;



endmodule
```



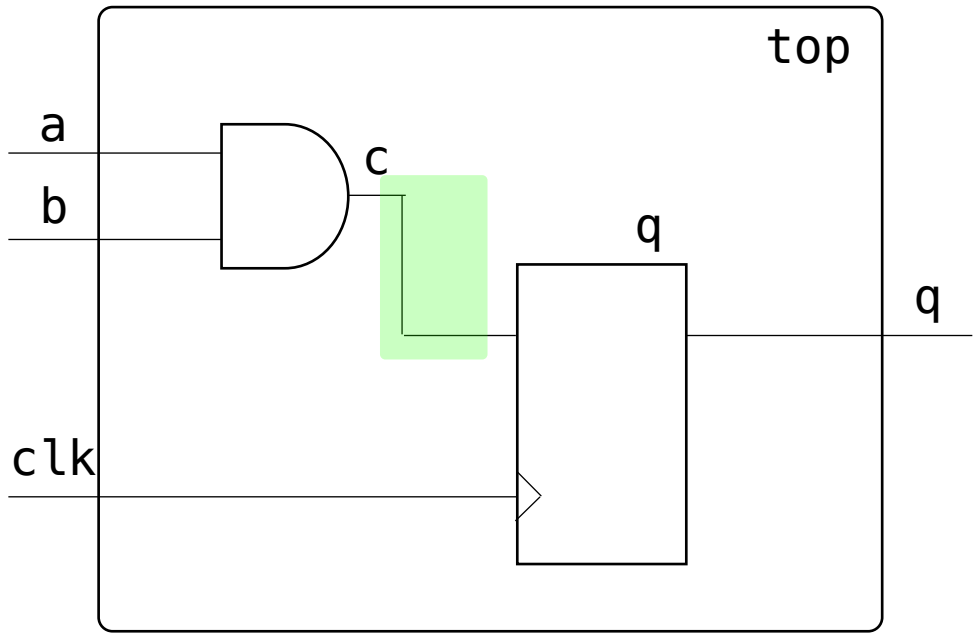
```
module top (
       input         a,
       input         b,
       input         clk,
       output  reg   q
);

wire c;

assign c = a & b;

always @ ()


endmodule
```



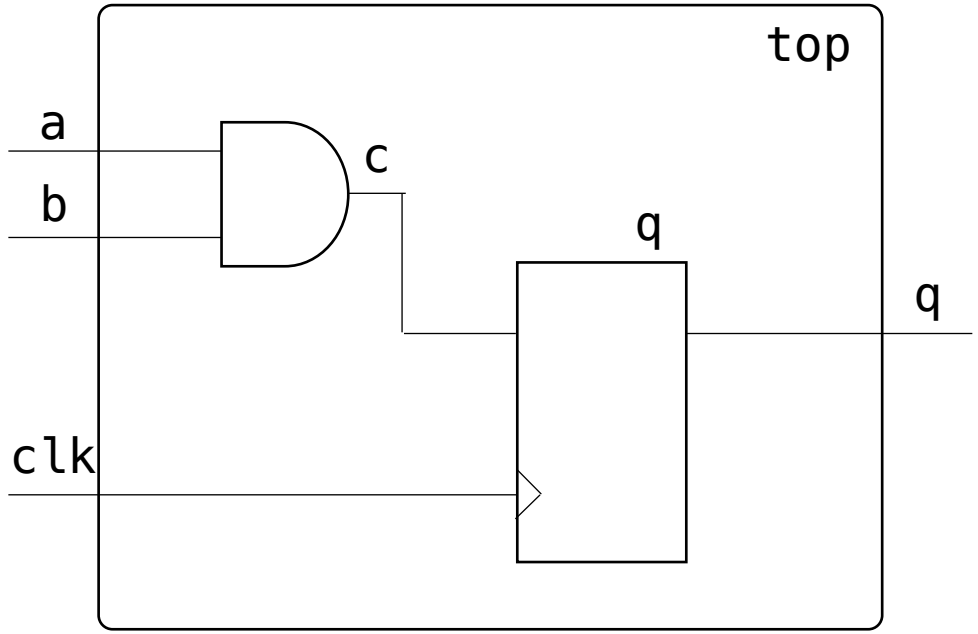
```
module top (
       input         a,
       input         b,
       input         clk,
       output  reg   q
);

wire c;

assign c = a & b;

always @ (posedge clk)


endmodule
```



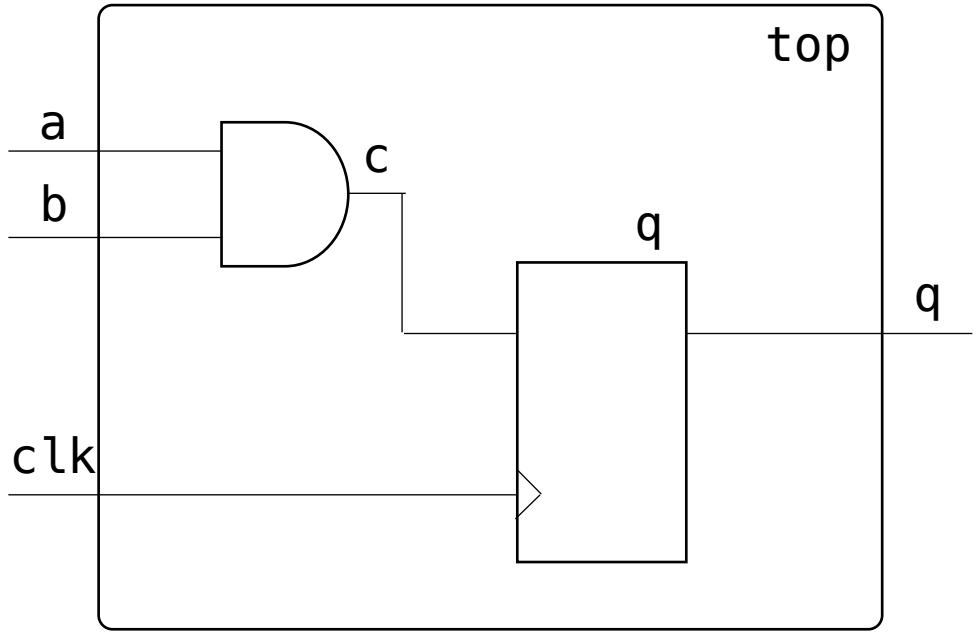
```
module top (
        input           a,
        input           b,
        input           clk,
        output  reg     q
);

wire c;

assign c = a & b;

always @ (posedge clk)
        q <= c;

endmodule
```



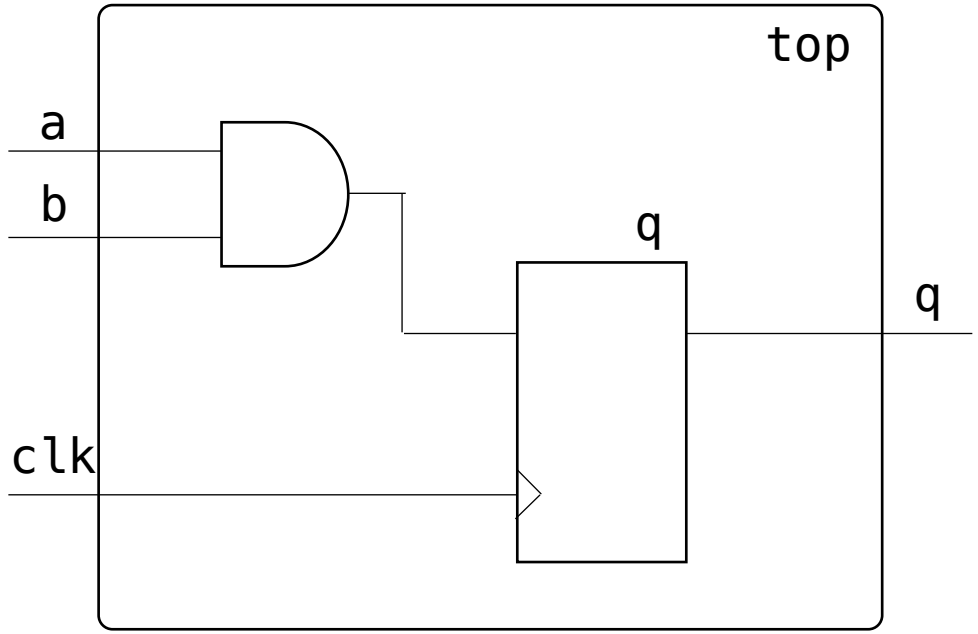
```
module top (
        input           a,
        input           b,
        input           clk,
        output  reg     q
);

wire c;

assign c = a & b;

always @ (posedge clk)
        q <= c;

endmodule
```



```
module top (
        input           a,
        input           b,
        input           clk,
        output  reg     q
);

wire c;

always @ (posedge clk)
        q <= c;

assign c = a & b;

endmodule
```

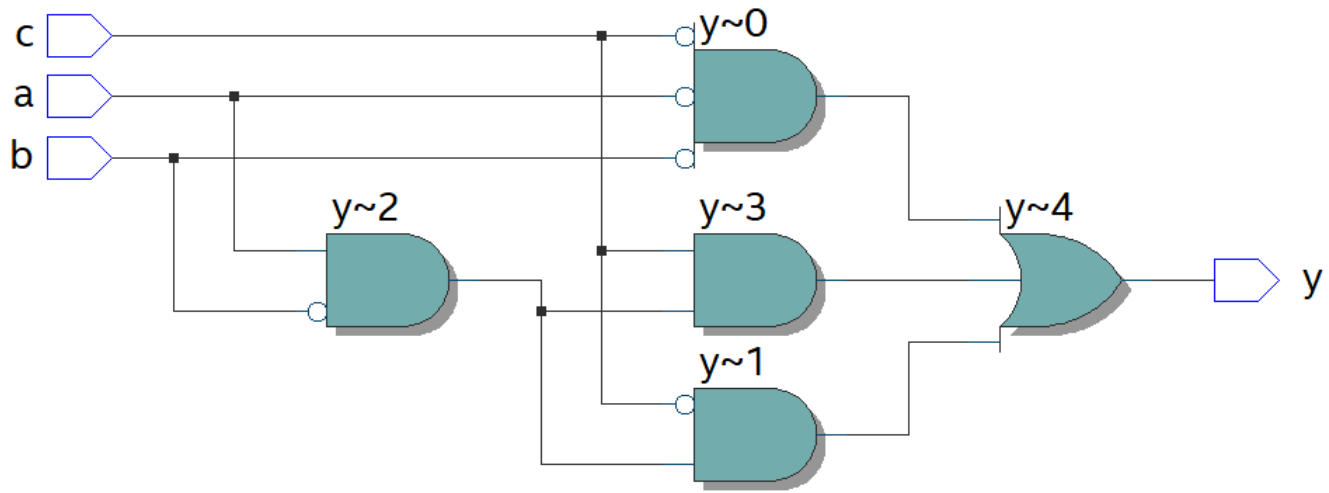


```
module top (
        input           a,
        input           b,
        input           clk,
        output  reg     q
);


always @ (posedge clk)
        q <= a & b;


endmodule
```



Комбинационная логика

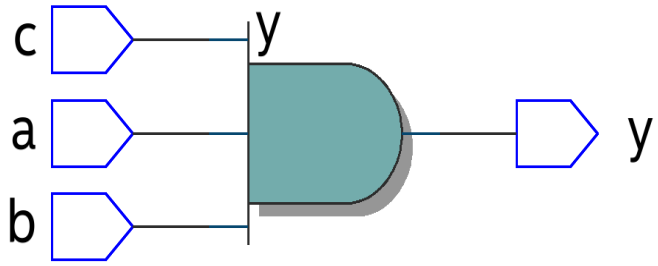
```
module dut (
        input  a, b, c,
        output y
);

assign y = a & ~b & ~c | a & ~b & c | a & ~b & c;

endmodule
```

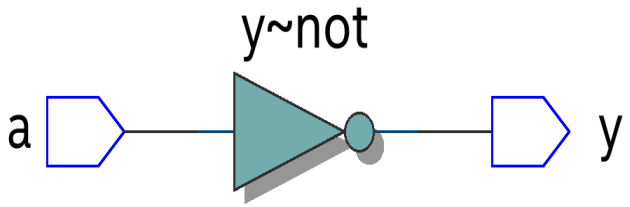


Иерархия модулей в Verilog

```
module and_3 (
       input  a, b, c,
       output y
);

assign y = a & b & c;

endmodule
```

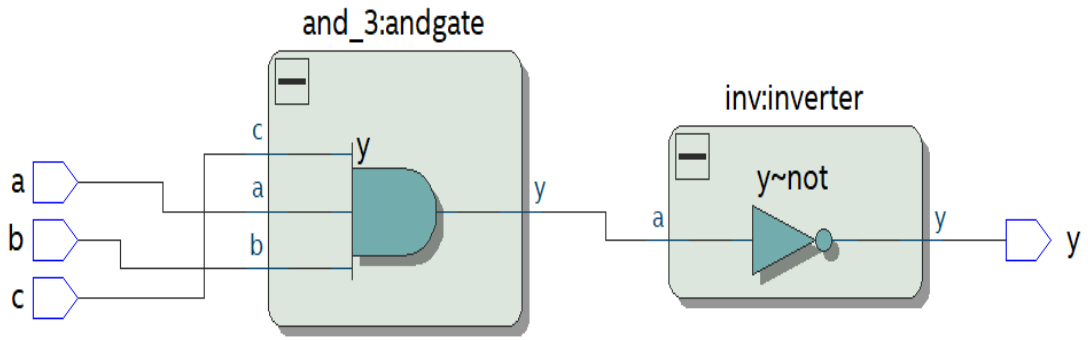
```
module inv (
       input  a
       output y
);

assign y = ~a;

endmodule
```





Иерархия модулей в Verilog

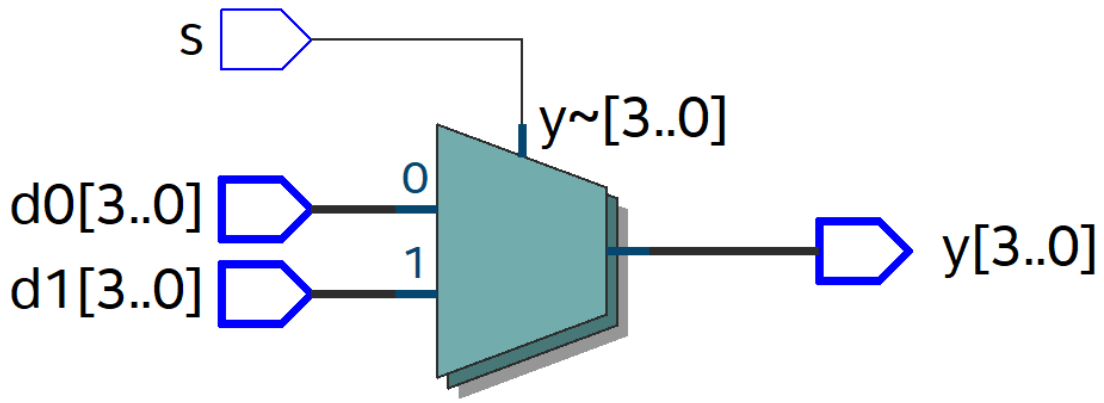
```
module dut (
        input  a, b, c,
        output y
);

wire n1;

and_3 andgate (
        .a(a),
        .b(b),
        .c(c),
        .y(n1)
);

inv inverter (
        .a(n1),
        .y(y)
);

endmodule
```



- Имя подключаемого модуля (**and_3**, **inv**)
- Название примитива. Например, нам может понадобиться **3** копии модуля **and_3**. Тогда мы сможем подключить 3 экземпляра модуля **and_3**, используя различные наименования для прототипов (andgate_1, andgate_2 …)
- Символ точка, перед наименованием порта отсылает к реальному порту подключаемого модуля (у модуля **inverter**, порты именуются **a, y**). В скобках обозначается куда будут подключаться сигналы в *top*-модуле

Тернарный оператор

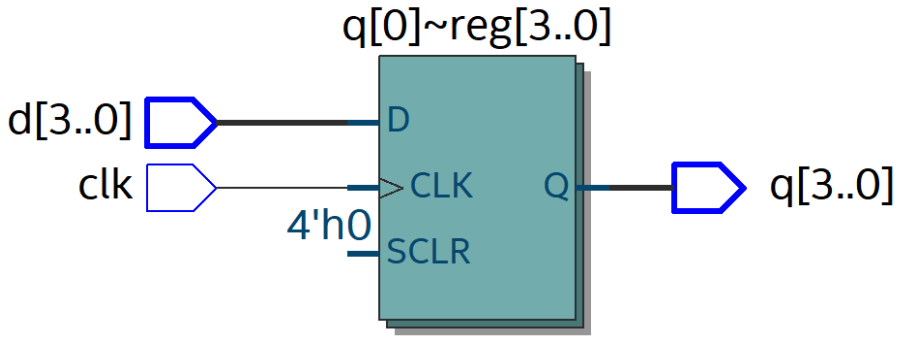
```
module ternarny (
        input   [3:0]   d0, d1,
        input           s,
        output  [3:0]   y
);

assign y = s ? d1 : d0;

endmodule
```



`? :` данный оператор называется тернарным, т.к. в нем используется **3** входных значения: `s`, `d1`, `d0`.

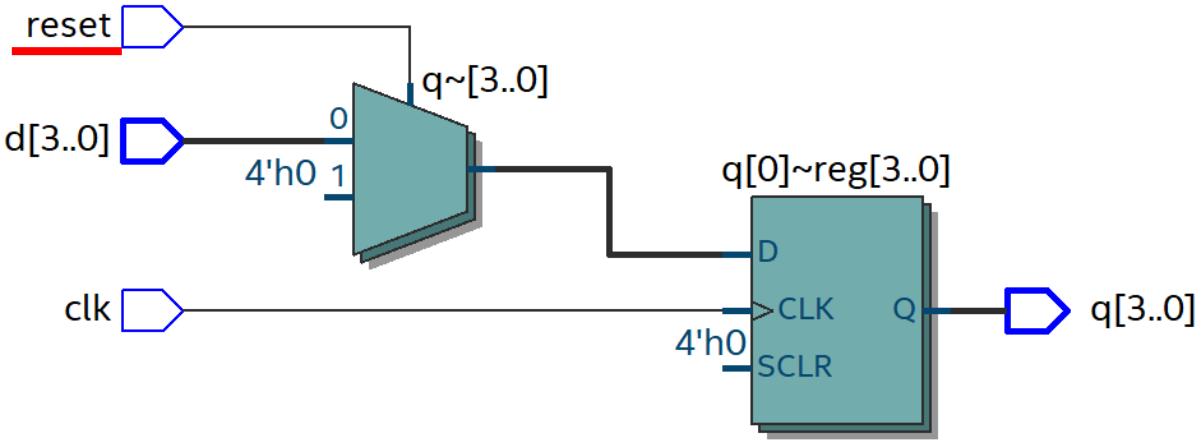
D-триггер

```
module flop (
        input               clk,
        input        [3:0]  d,
        output reg   [3:0]  q
);

always @ (posedge clk)
        q <= d;

endmodule
```



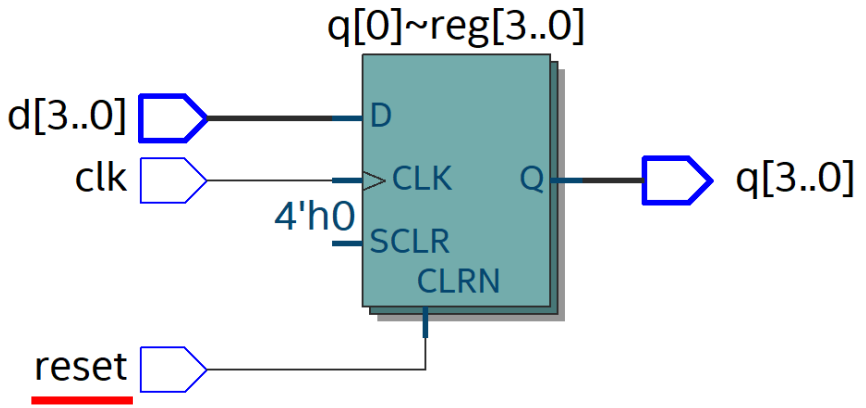
D-триггер, с синхронным сбросом

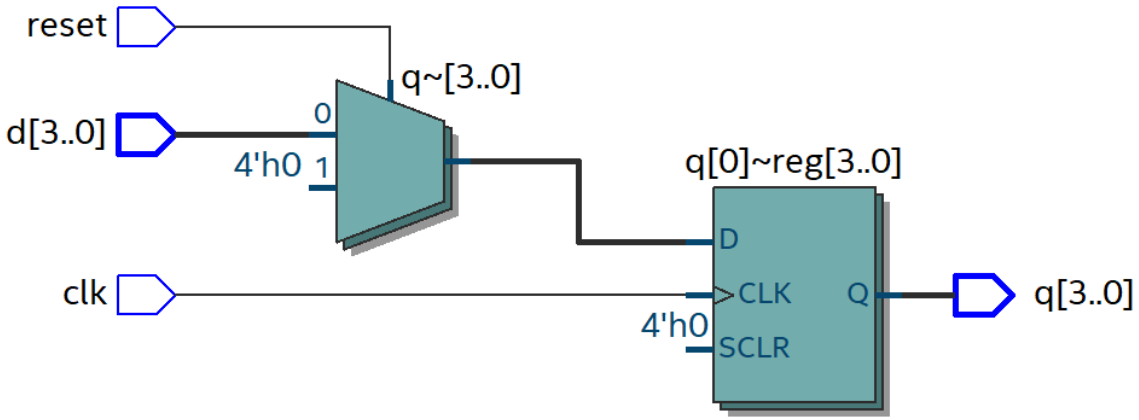
```
module flopr (
        input               clk,
        input               reset,
        input        [3:0]  d,
        output reg   [3:0]  q
);

always @ (posedge clk)
        if (reset)      q <= 4'b0;
        else            q <= d;

endmodule
```

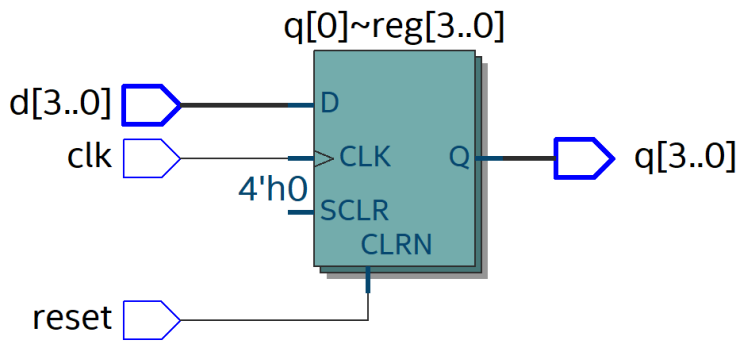


D-триггер, с асинхронным сбросом

```
module flopr (
        input               clk,
        input               reset,
        input        [3:0]  d,
        output reg   [3:0]  q
);

always @ (posedge clk or posedge reset)
        if (reset)      q <= 4'b0;
        else            q <= d;

endmodule
```

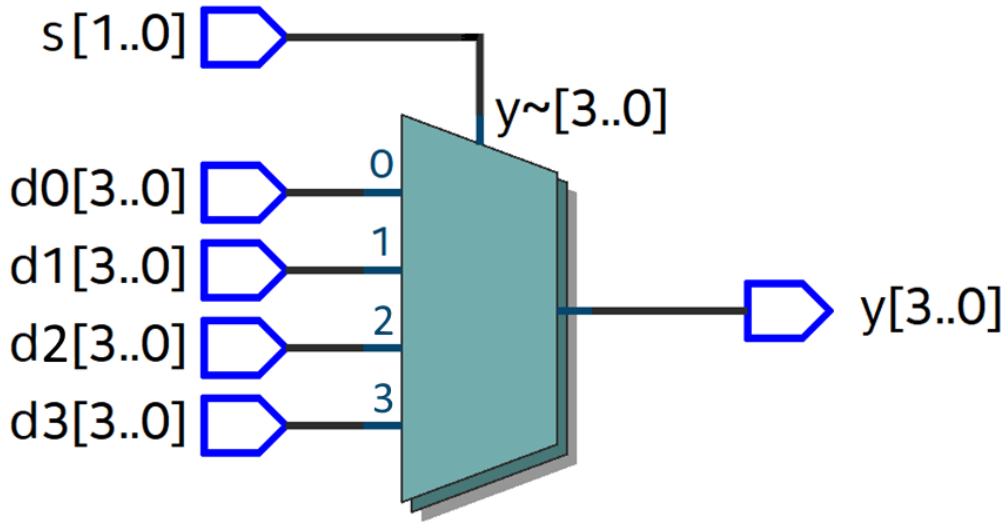



reset
d[3..0]
clk
q~[3..0]
0
4'h0
1
q[0]~reg[3..0]
D
CLK
Q
4'h0
SCLR
q[3..0]


q[0]~reg[3..0]
d[3..0]
clk
D
CLK
Q
4'h0
SCLR
CLRN
reset
q[3..0]

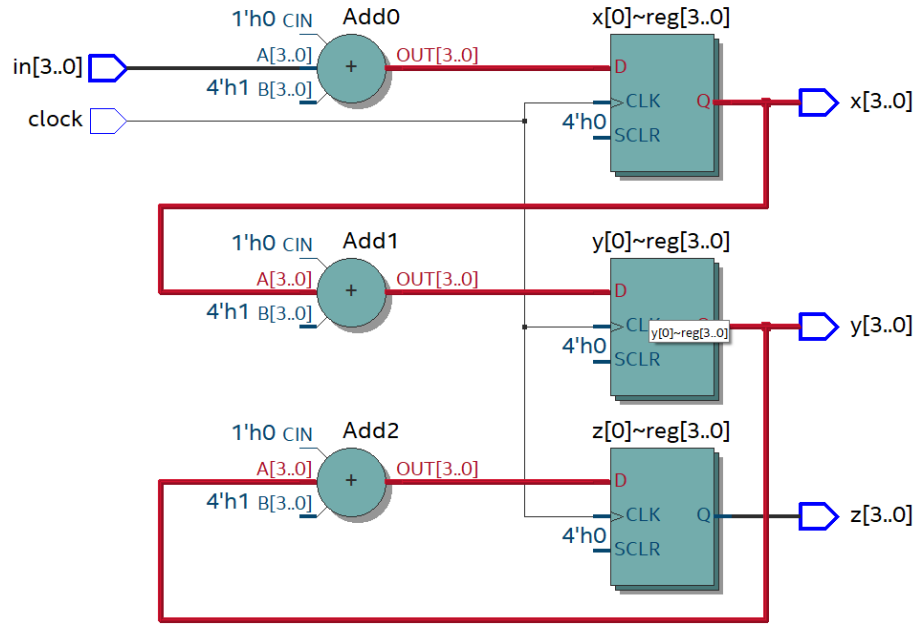
CASE

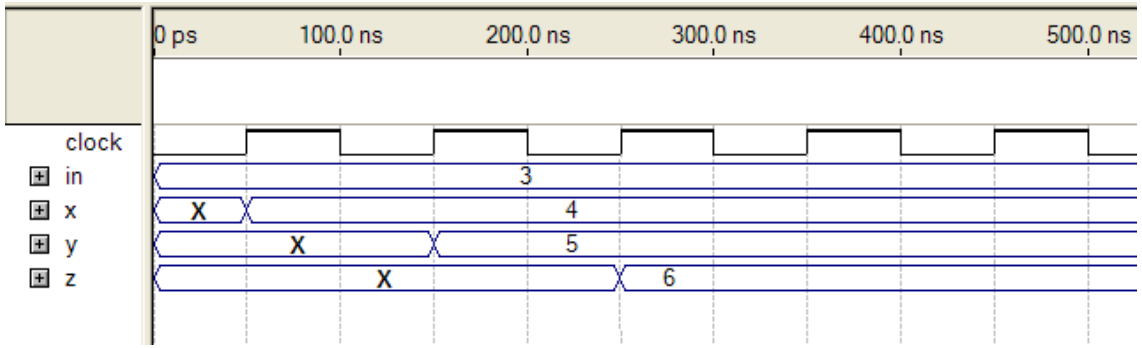
```
module mux (
        input       [3:0]   d0, d1, d2, d3,
        input       [1:0]   s,
        output  reg [3:0]   y
);

always @ (*) begin
        case (s)
                2'b00:  y = d0;
                2'b01:  y = d1;
                2'b10:  y = d2;
                2'b11:  y = d3;
        endcase
end

endmodule
```



Неблокирующее присваивание

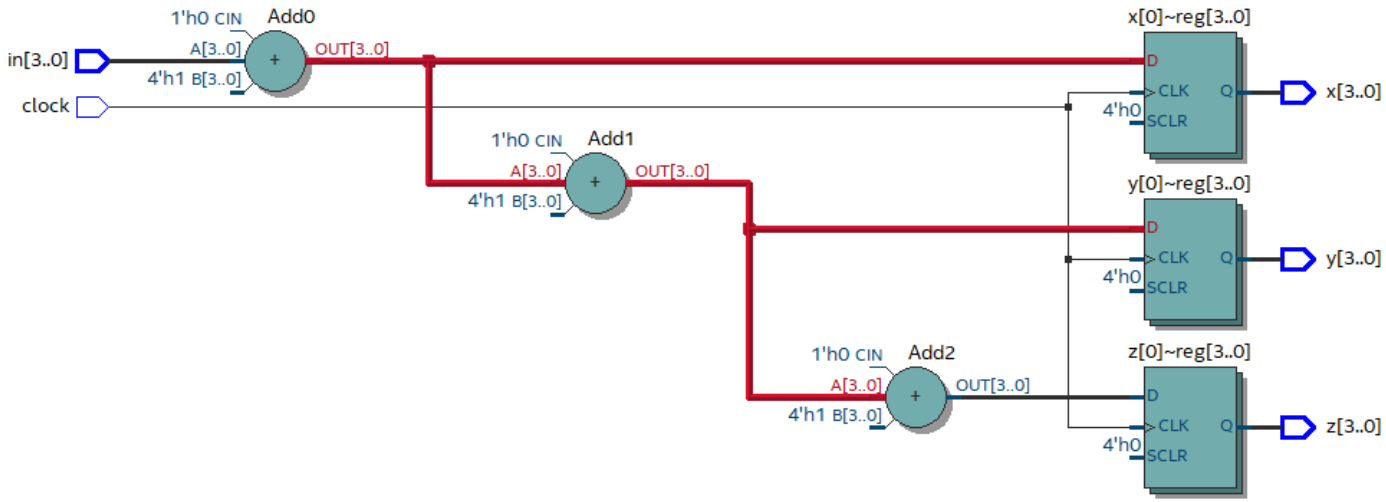
```
module test (
        input               clock,
        input        [3:0]  in,
        output reg   [3:0]  x,
        output reg   [3:0]  y,
        output reg   [3:0]  z
);

always @ (posedge clock) begin
        x <= in + 1;
        y <= x + 1;
        z <= y + 1;
end

endmodule
```

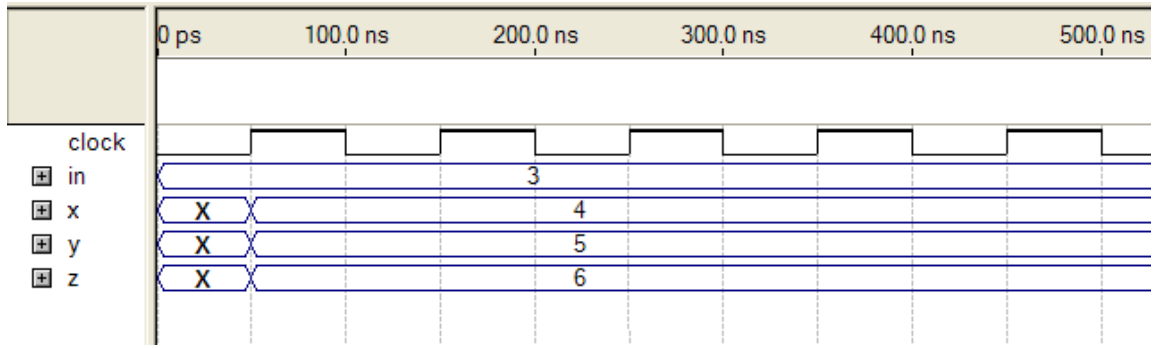




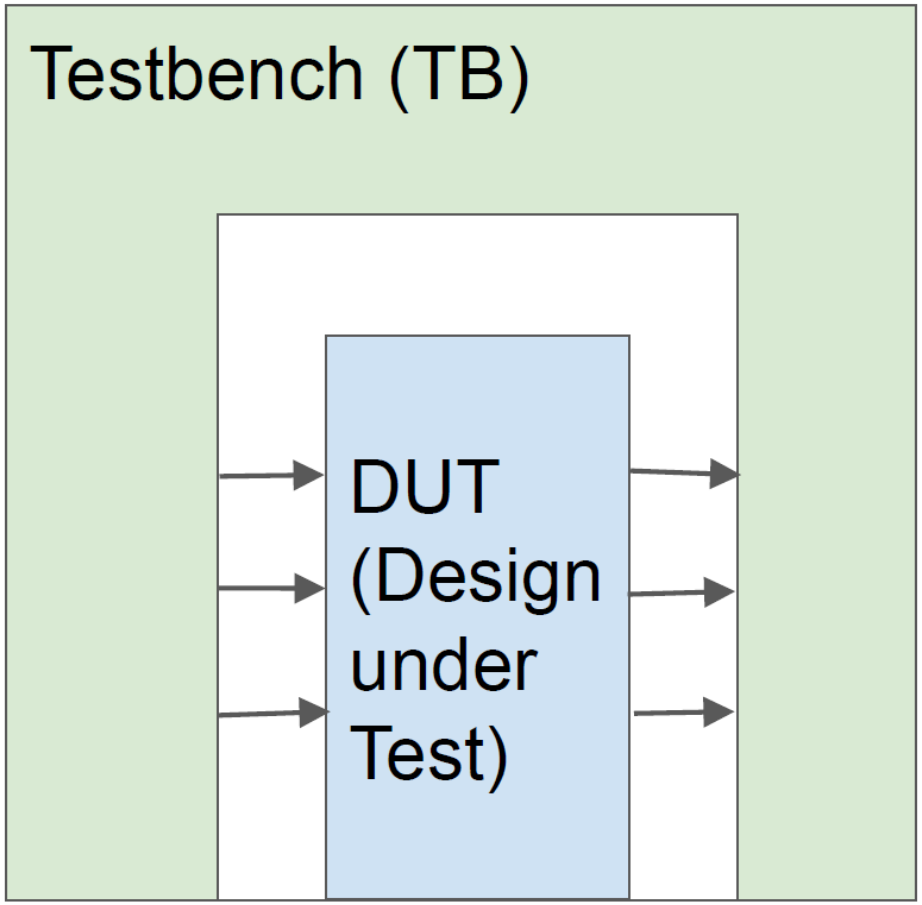
Блокирующее присваивание

```
module test (
        input               clock,
        input       [3:0]   in,
        output reg  [3:0]   x,
        output reg  [3:0]   y,
        output reg  [3:0]   z
);

always @ (posedge clock) begin
        x = in + 1;
        y = x + 1;
        z = y + 1;
end

endmodule
```

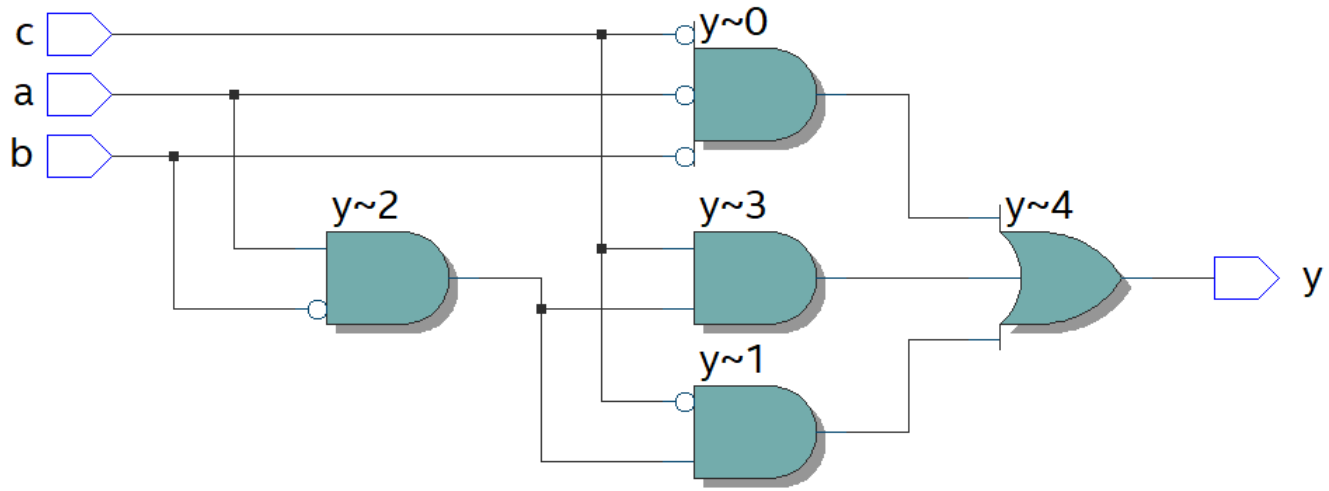




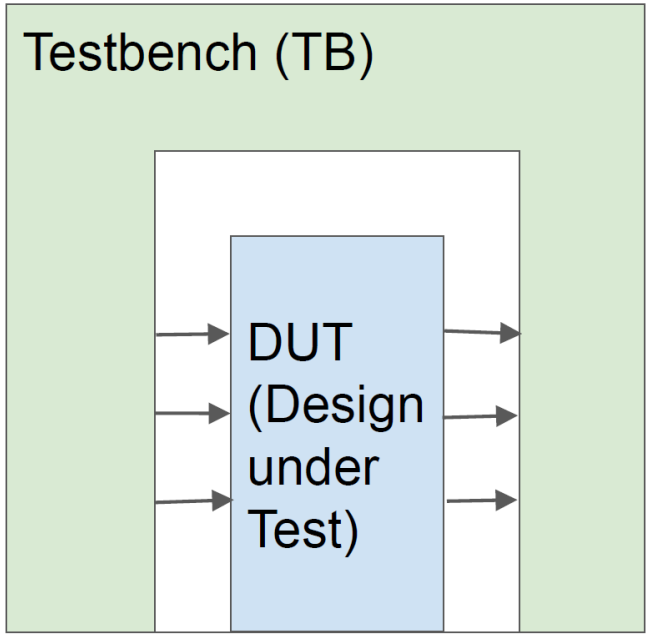
Тестовое окружение

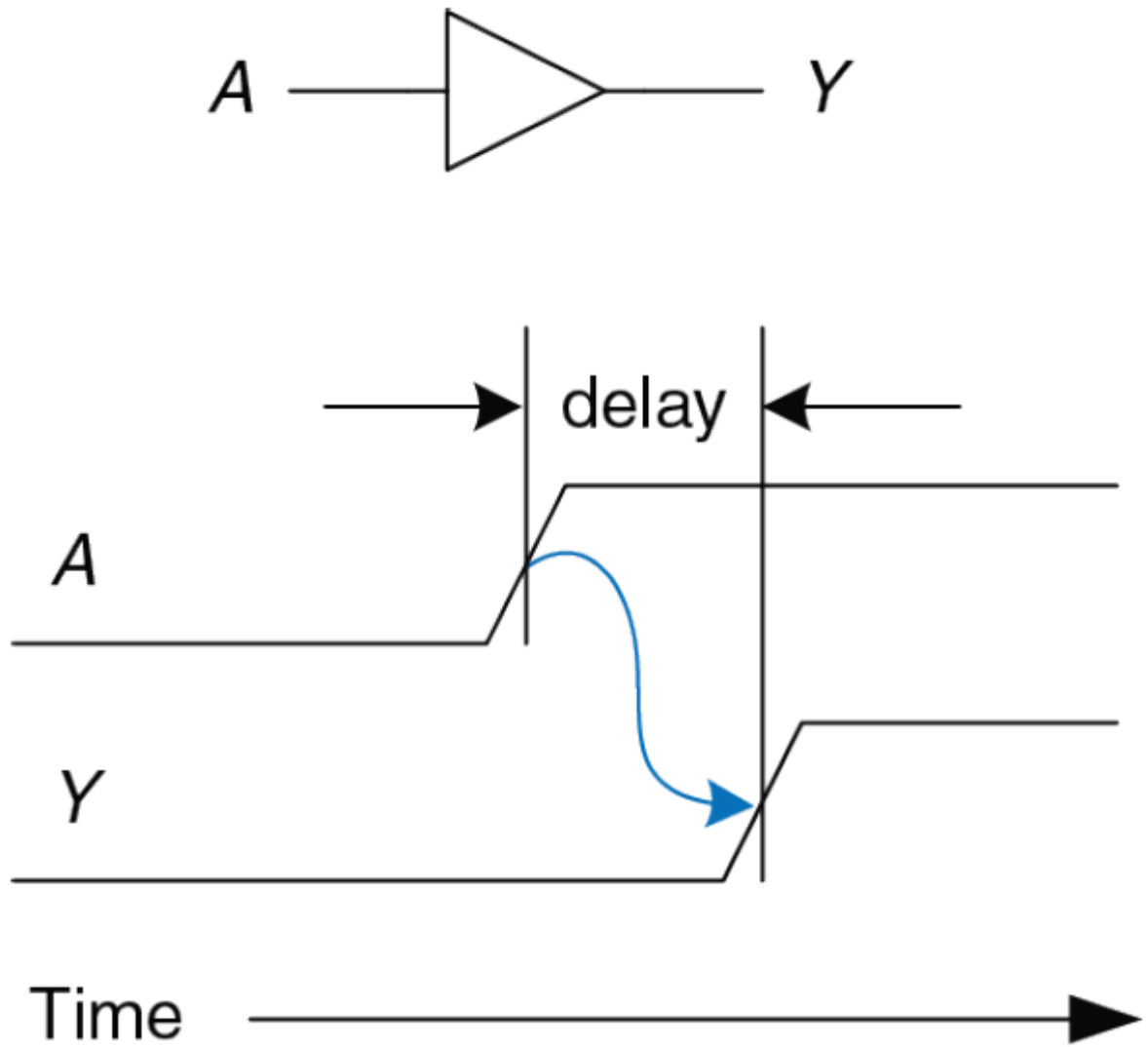


```
module my_module (
        input  a, b, c,
        output y
);

assign y = a & ~b & ~c | a & ~b & c | a & ~b & c;

endmodule
```

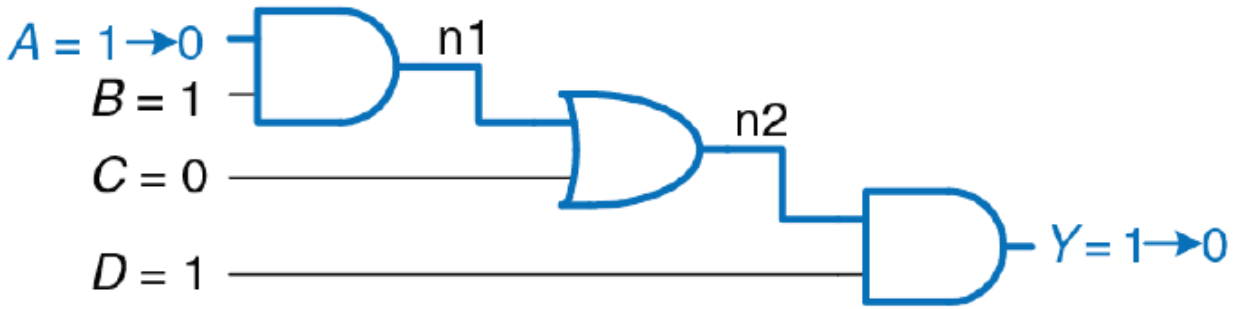


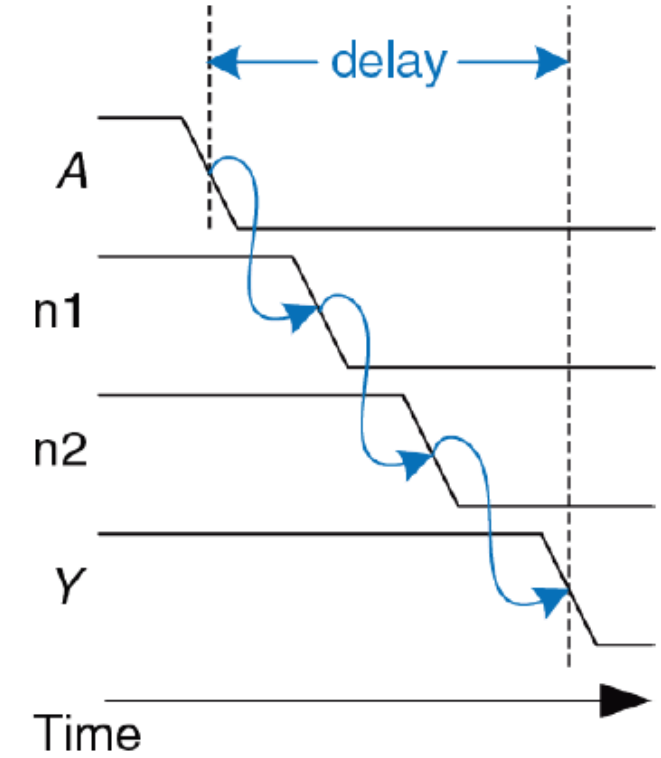
```
`timescale 1ns / 1ps

module testbench ();

reg    a, b, c;
wire   y;

my_module dut (a, b, c, y);

initial begin
       a = 0; b = 0; c = 0; #10;
       if (y === 1)
              $display("Good");
       else
              $display("Bad");
       c = 1; #10;
end

endmodule
```



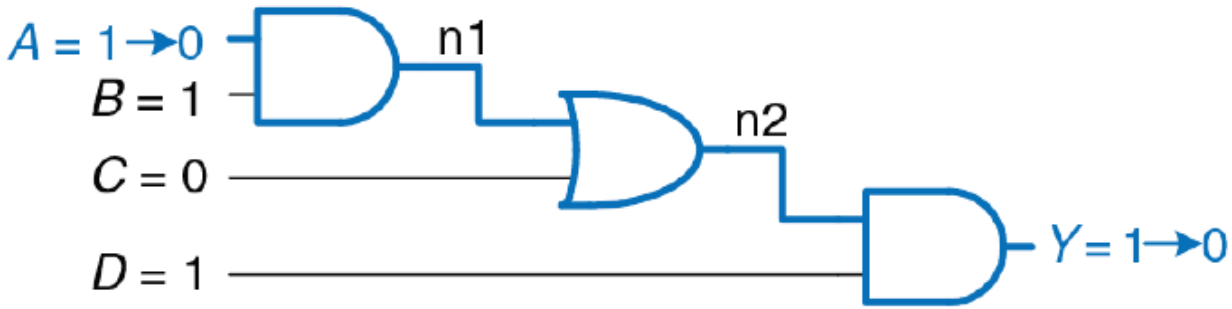

A
Y
delay
A
Y
Time

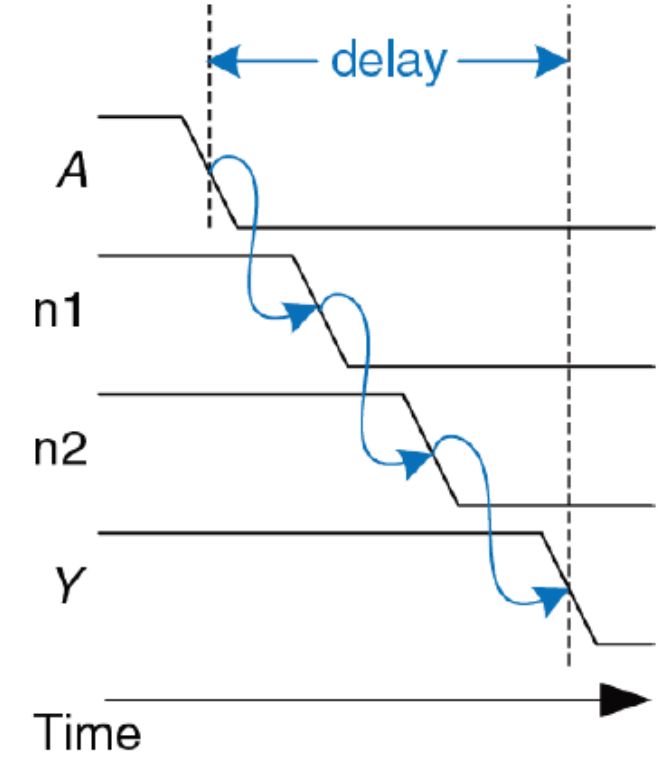
Critical Path





Critical Path





Short Path