1

Verilog HDL. Сумматор

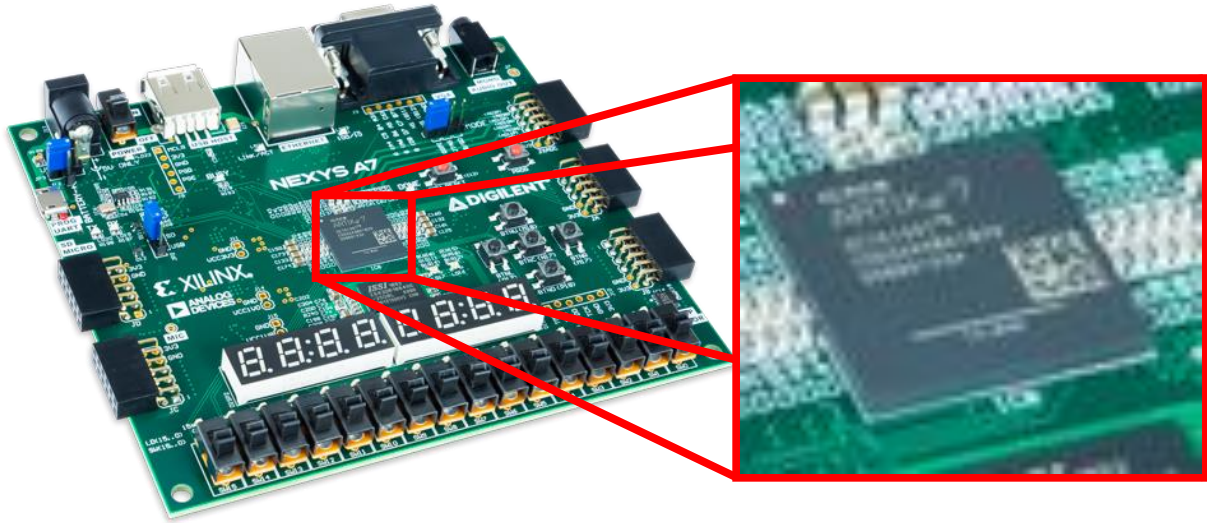
Микропроцессорные системы и средства

План лабораторной работы

- 1 пара
  - О лабораторных работах (**T**)
  - Введение в FPGA и Verilog HDL (**T**)
  - Тренинг по Vivado и Verilog HDL (**TS**)
  - Обзор отладочного стенда (**TS**)
- 2 пара
  - Теория Сумматор (**T**)
  - Описание сумматора на Verilog HDL (**S**)
  - Реализация сумматора на отладочном стенде (**S**)

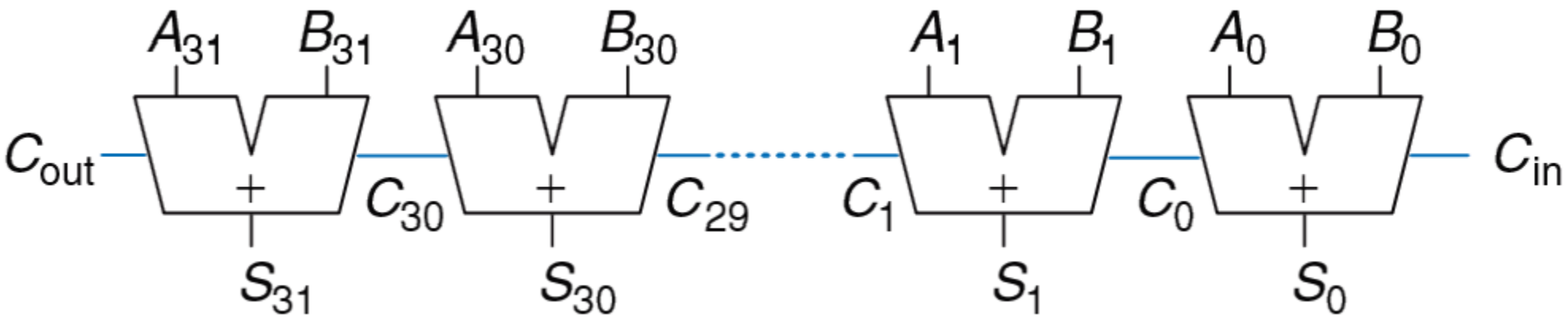
Цель лабораторных работ

- Используя Verilog HDL реализовать на базе FPGA программируемый процессор с архитектурой RISC-V

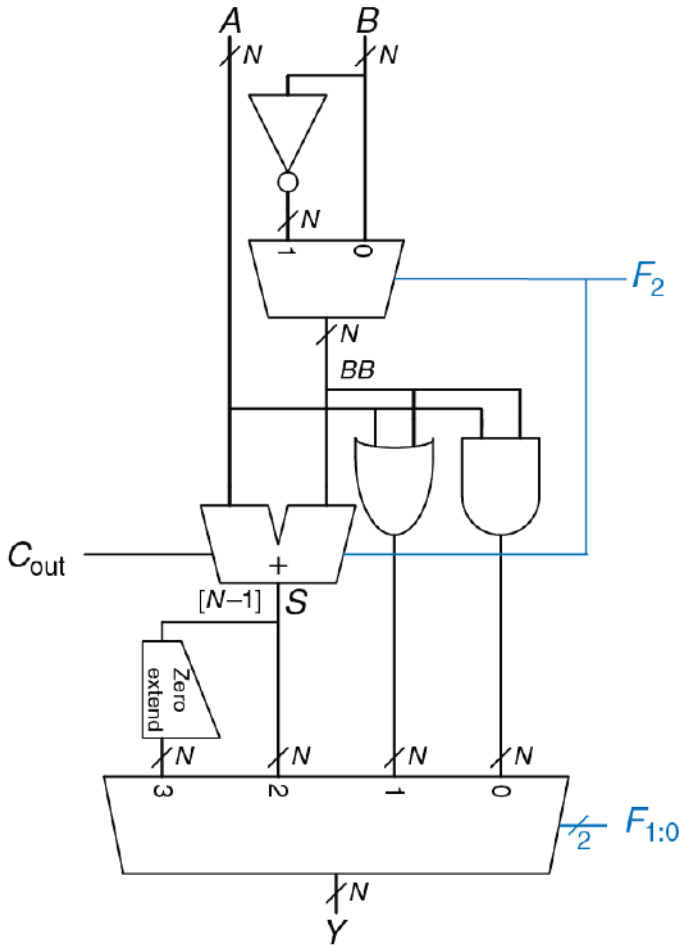


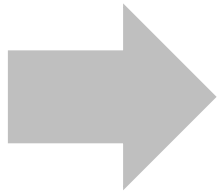

```
module fulladder (a, b, cin, s, cout);

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
```

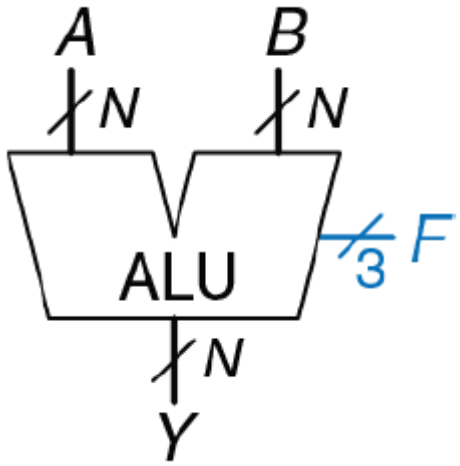
ЛР1. Verilog HDL. Сумматор



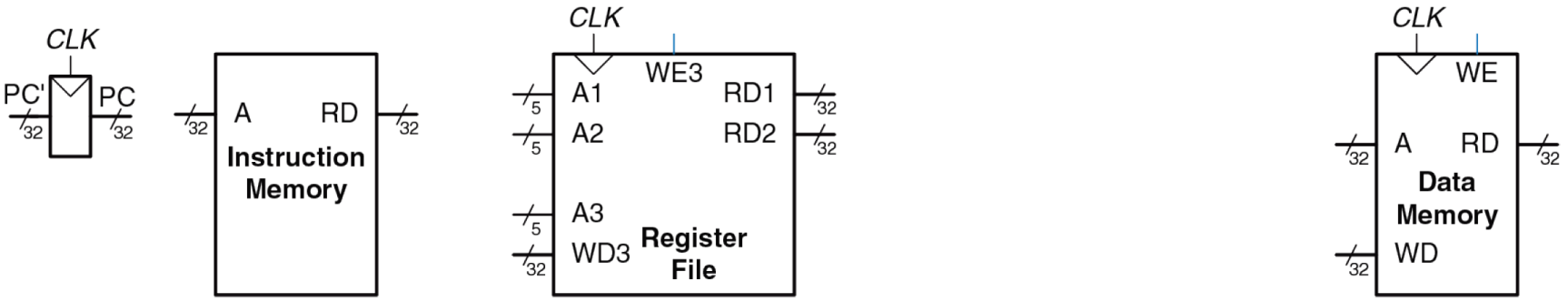
ЛР2. Арифметико-логическое устройство



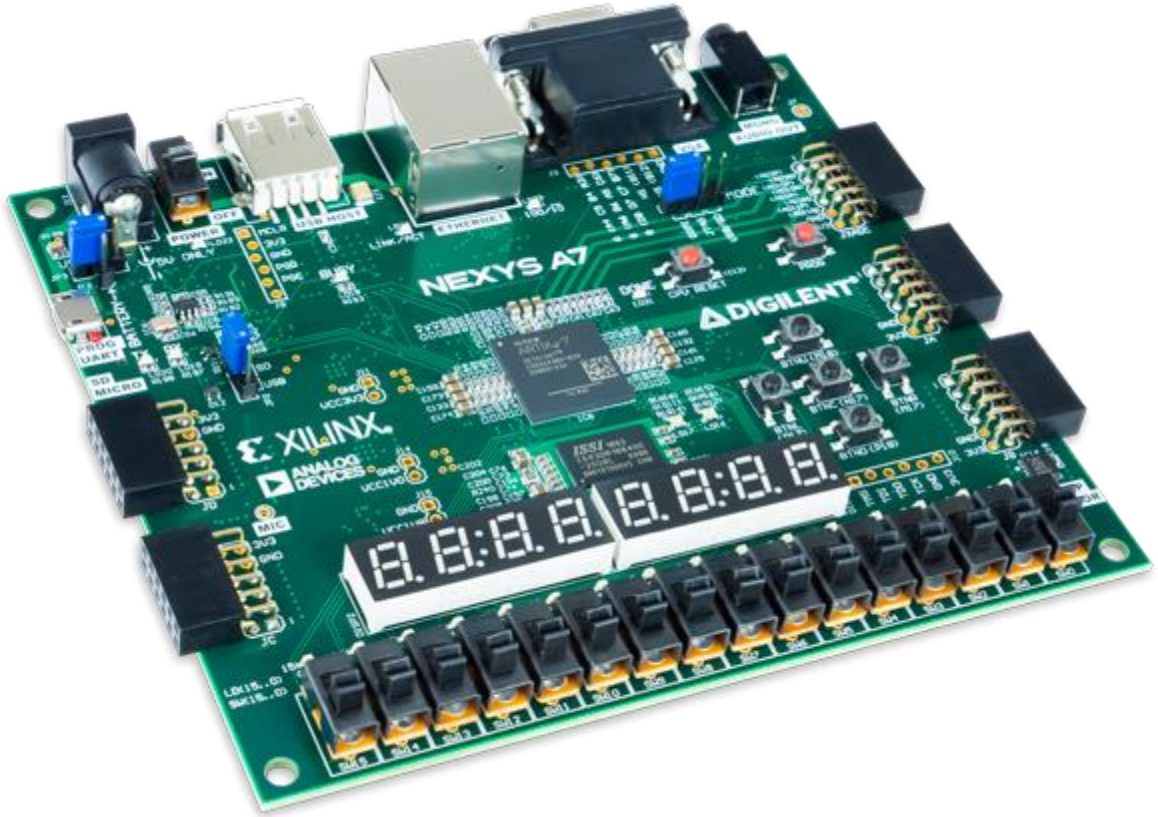




ЛР3. Регистровый файл. Память.

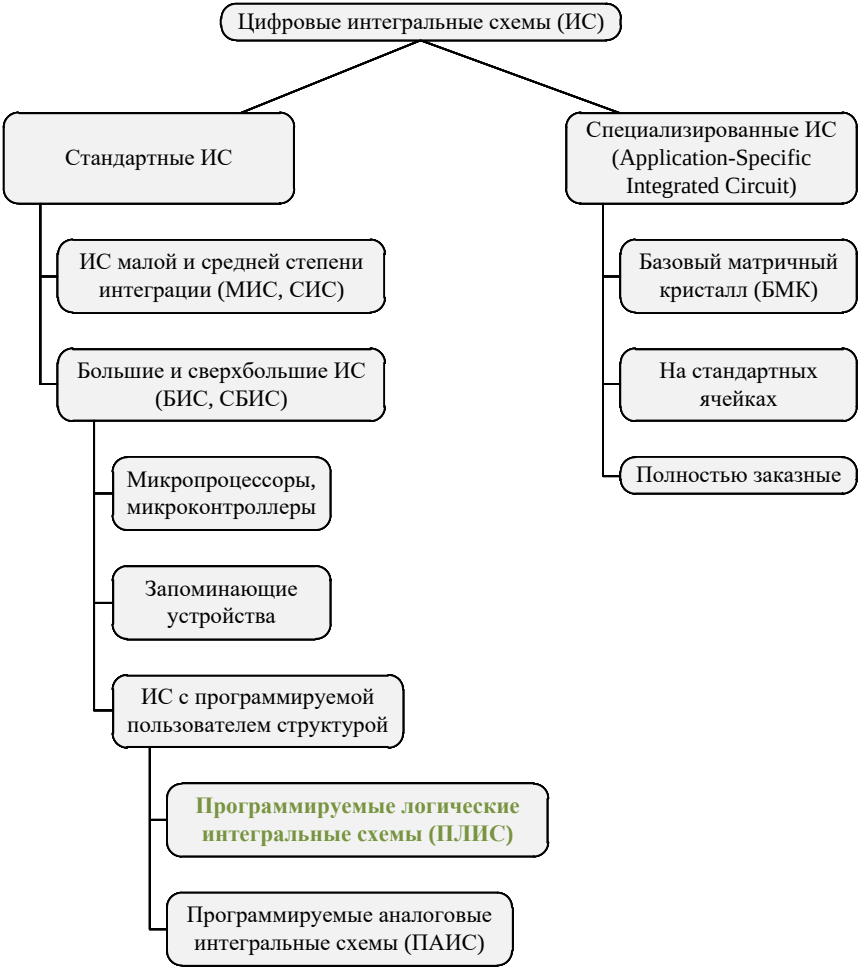


ЛР4. Процессор. Программирование

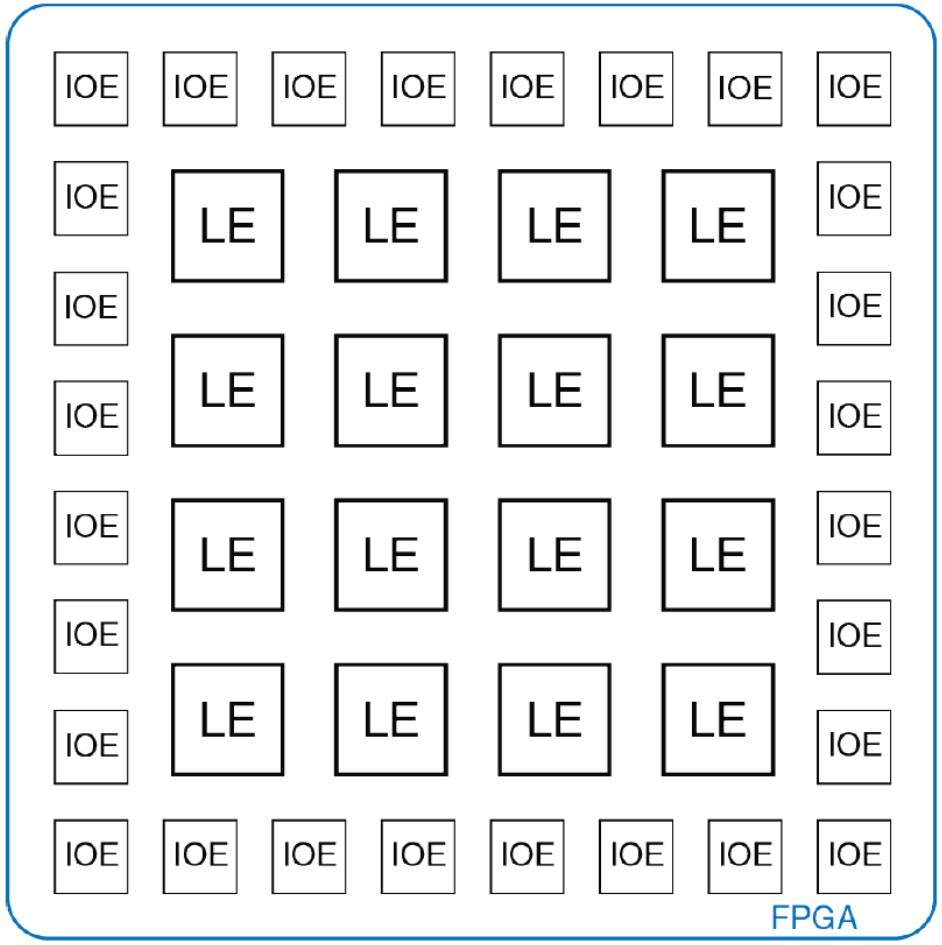


План лабораторных работ

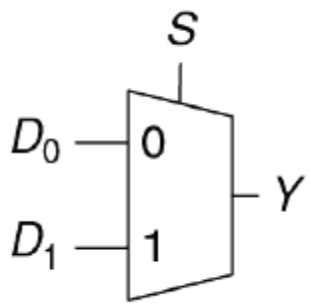
1. Verilog HDL. Сумматор → **3**
2. Арифметико-логическое устройство → **7**
3. Регистровый файл. Память. → **11**
4. Процессор. Программирование → **15**

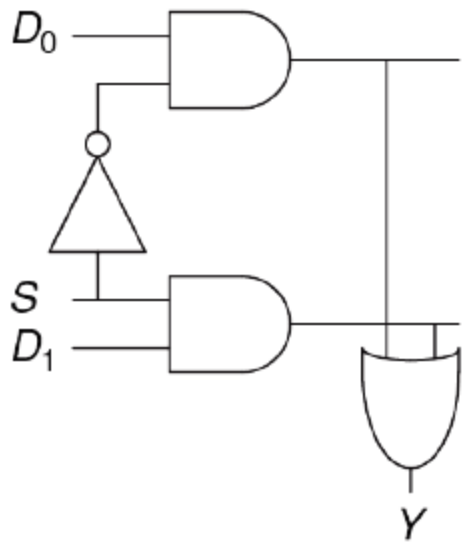

Цифровые интегральные схемы (ИС)
Стандартные ИС
Специализированные ИС (Application-Specific Integrated Circuit)
ИС малой и средней степени интеграции (МИС, СИС)
Большие и сверхбольшие ИС (БИС, СБИС)
Микропроцессоры, микроконтроллеры
Запоминающие устройства
ИС с программируемой пользователем структурой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

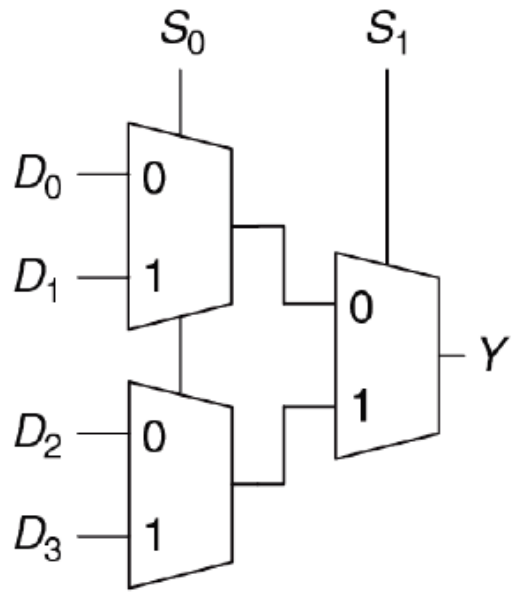
FPGA



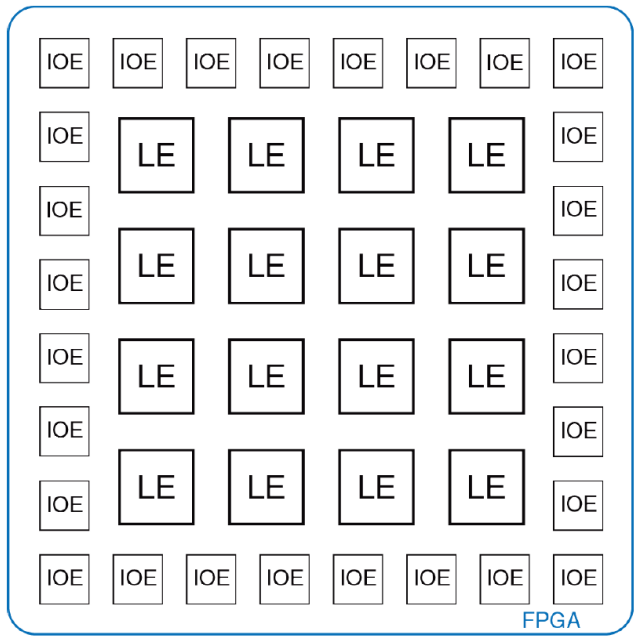
Мультиплексор

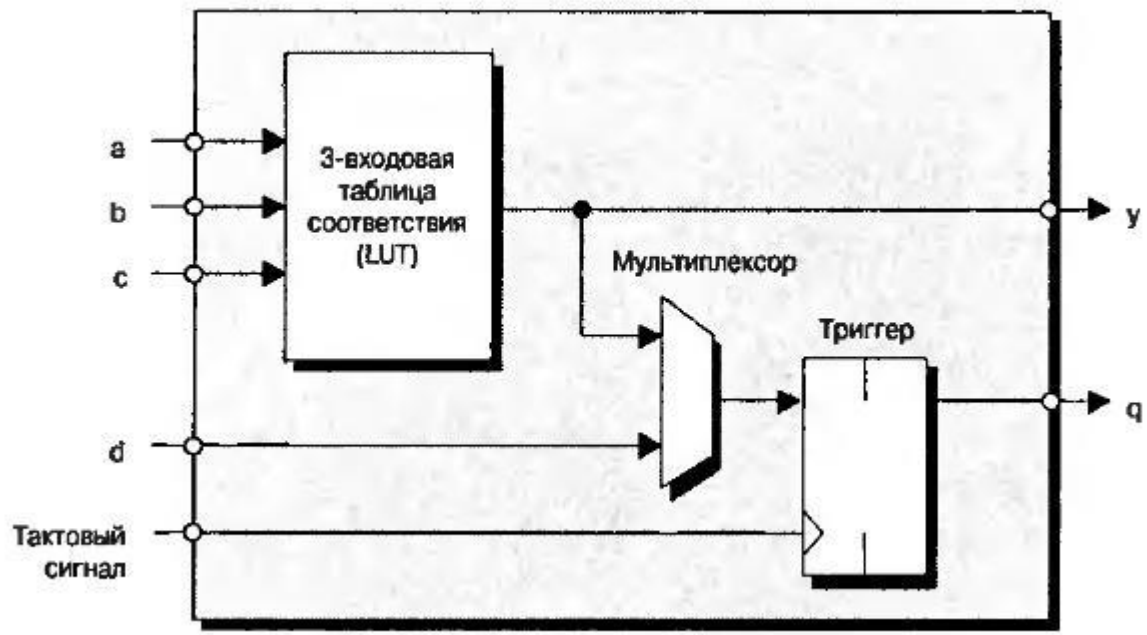




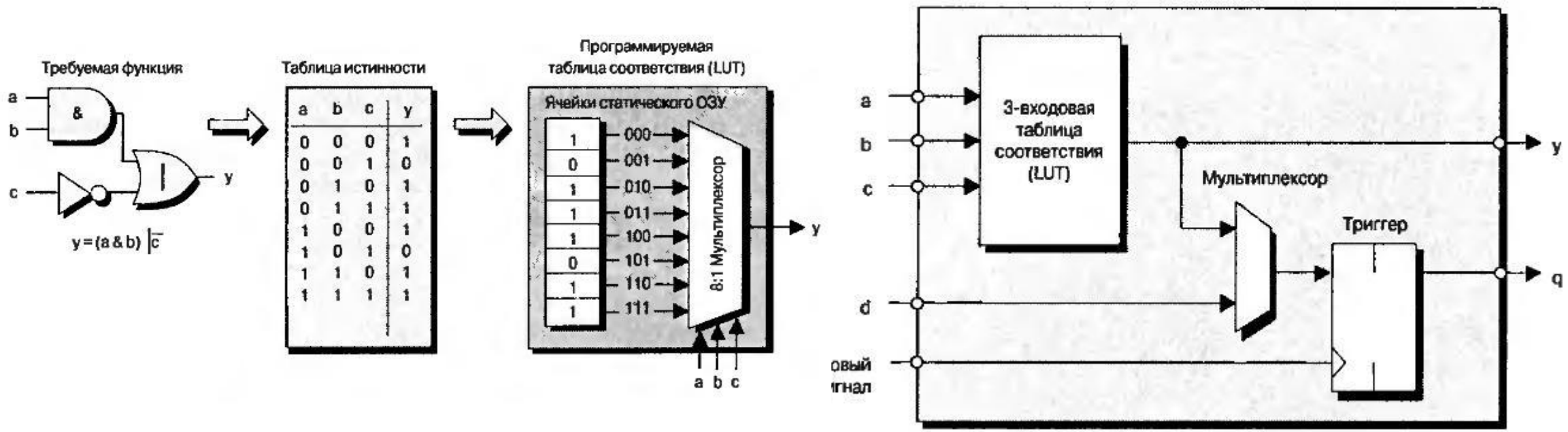


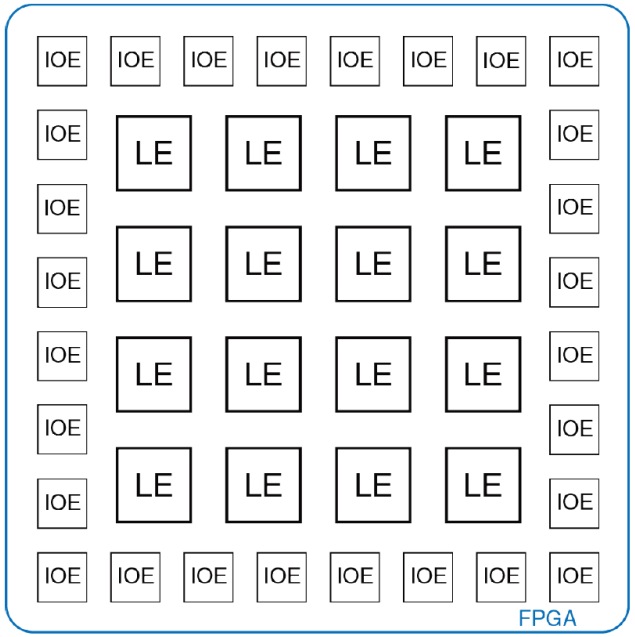
LE

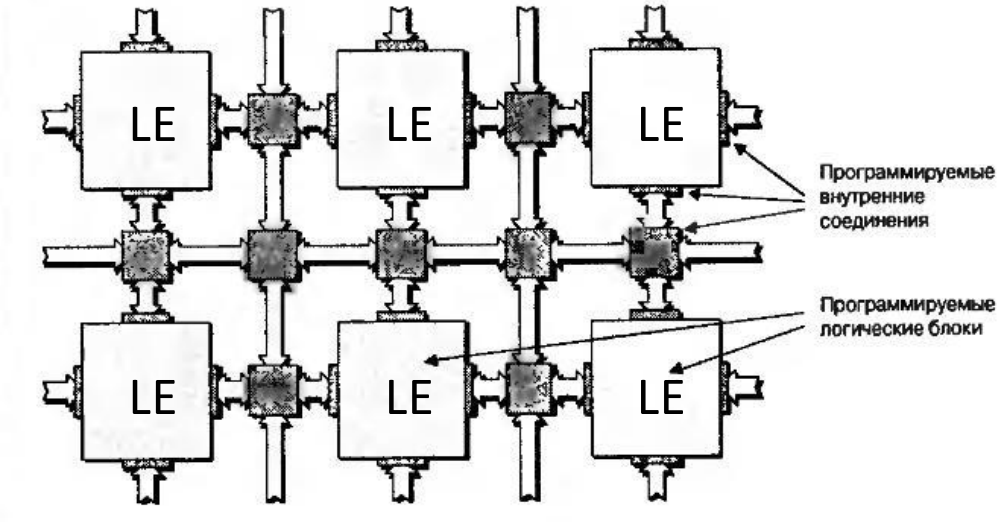

IOE
LE
FPGA

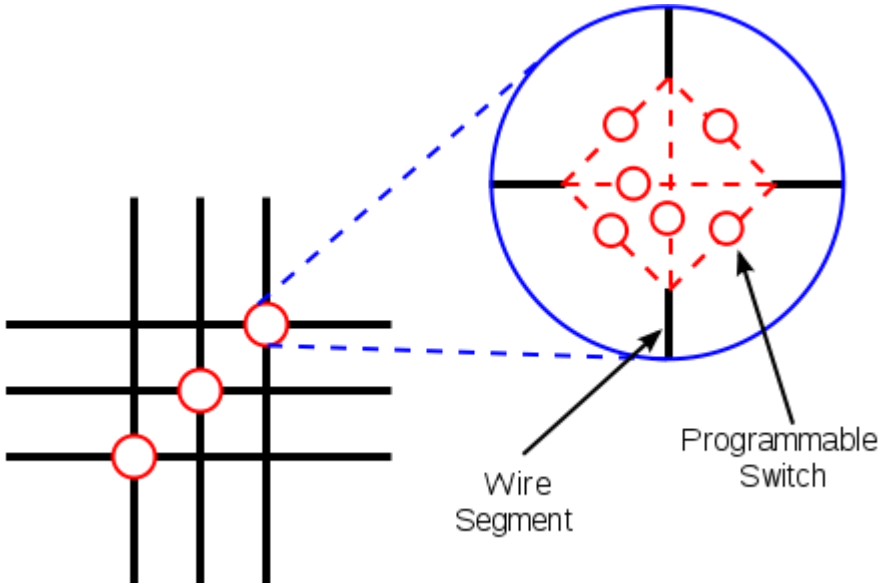

a
b
c
d
Тактовый сигнал
3-входовая таблица соответствия (LUT)
Мультиплексор
Триггер
y
q

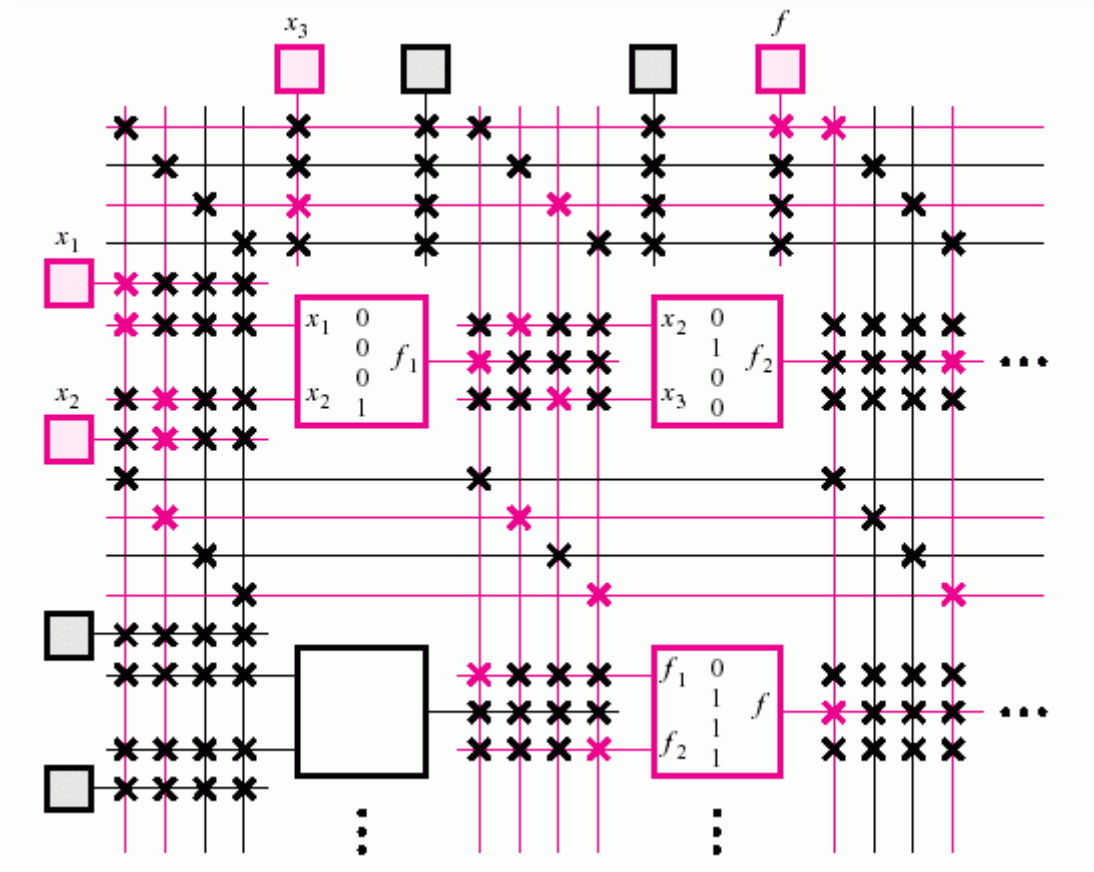
LE

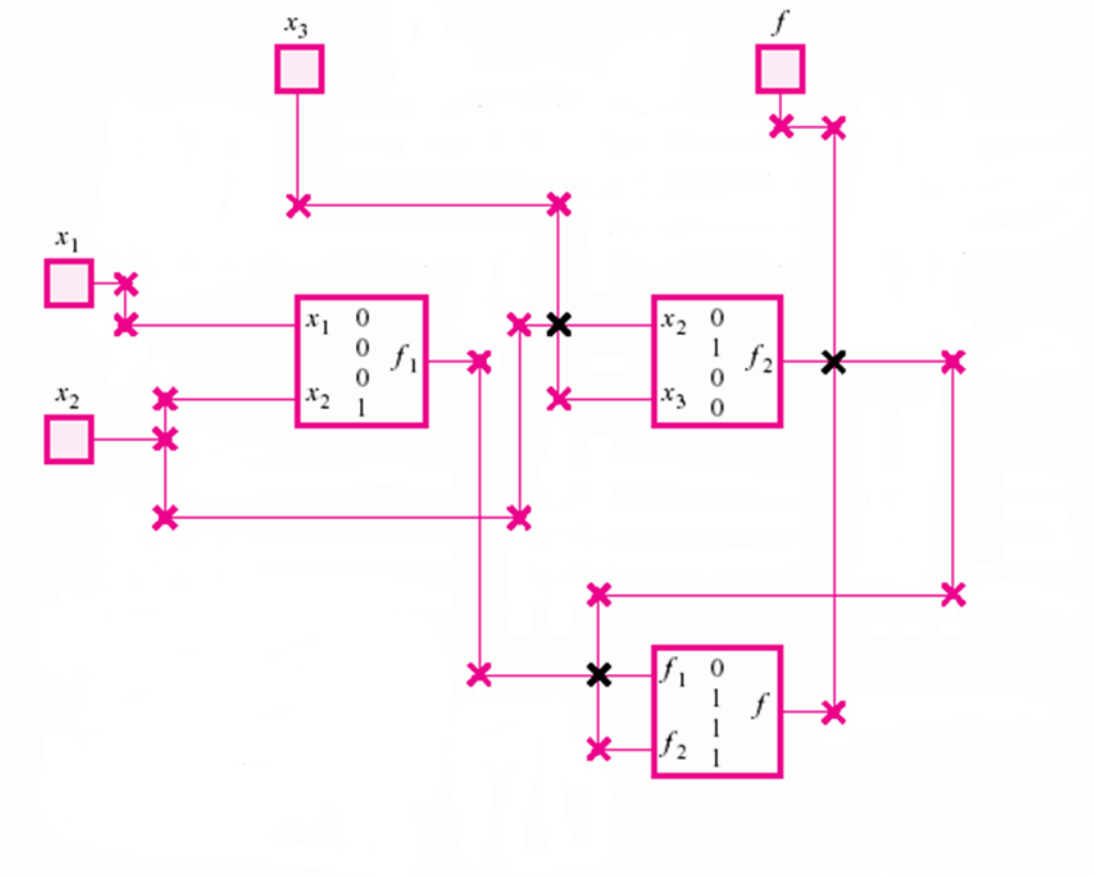

Требуемая функция
a
b
c
&
y
y = (a & b) |c|
Таблица истинности
a b c y
0 0 0 1
0 0 1 0
0 1 0 1
0 1 1 1
1 0 0 1
1 0 1 0
1 1 0 1
1 1 1 1
Программируемая таблица соответствия (LUT)
Ячейки статического ОЗУ
1
0
1
1
1
0
1
1
000
001
010
011
100
101
110
111
8:1 Мультиплексор
y
a b c
a
b
c
d
овый
гнал
3-входовая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


IOE
LE
FPGA

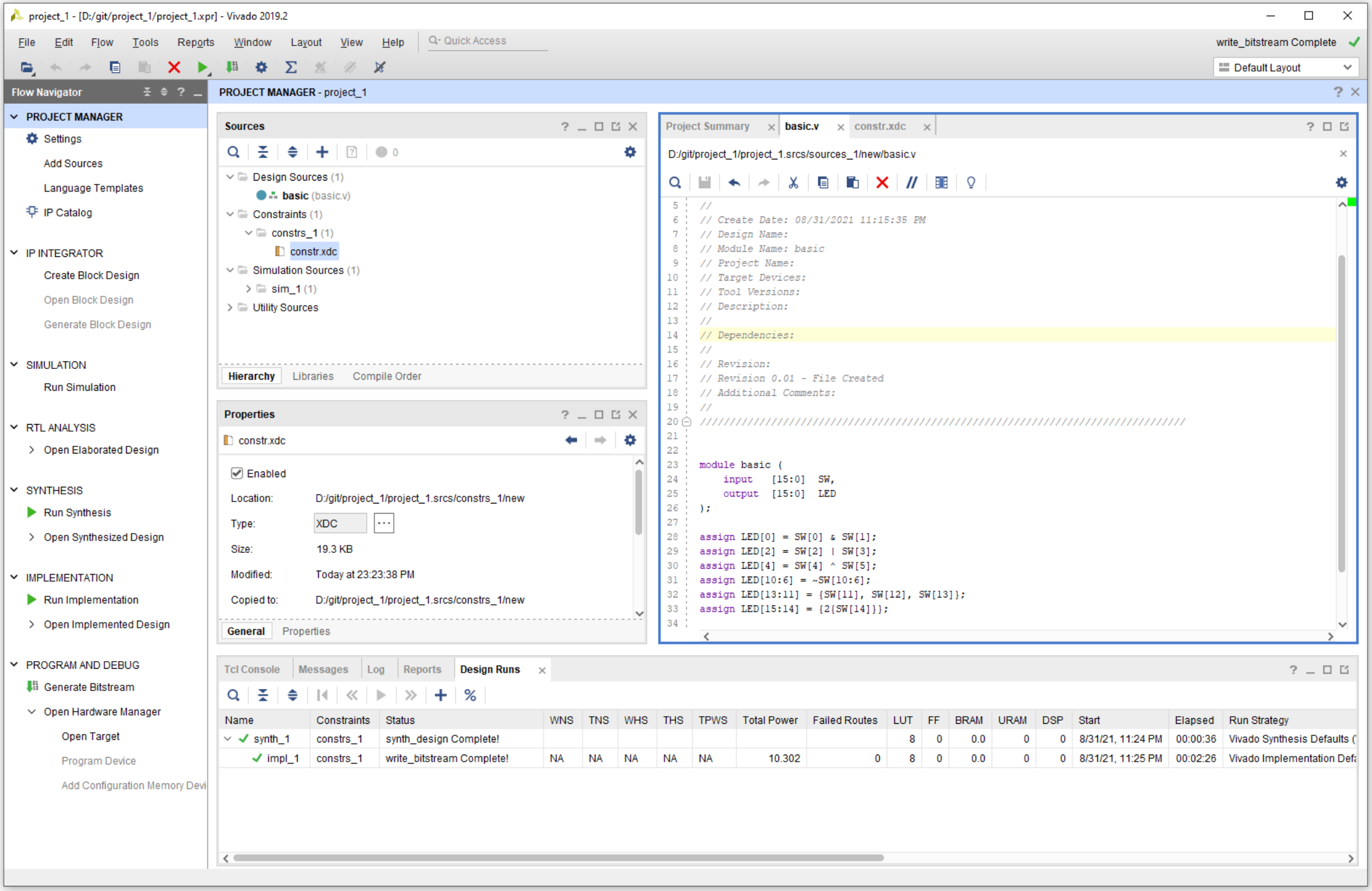

LE
Программируемые внутренние соединения
Программируемые логические блоки


Wire Segment
Programmable Switch

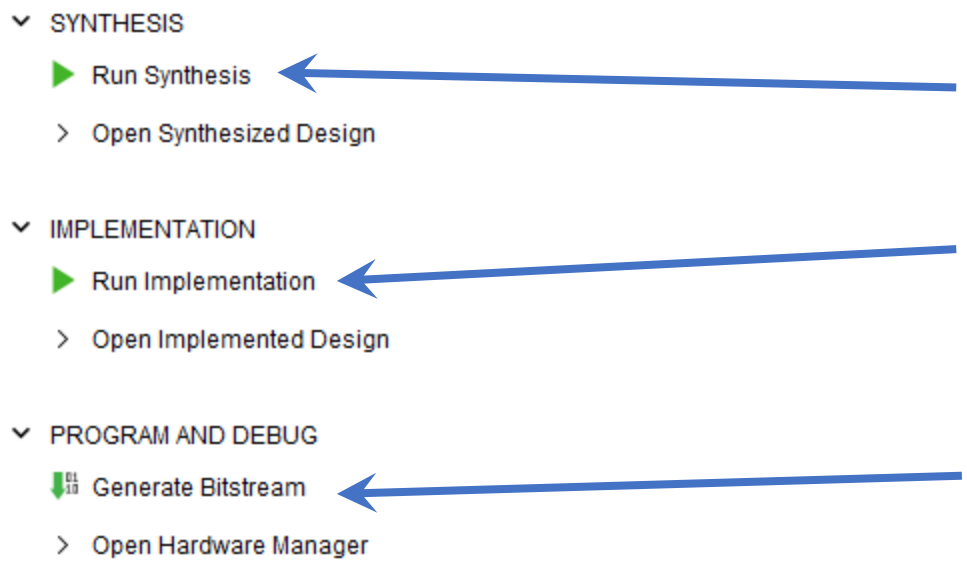

x_3
f
x_1
x_2
x_1 0 0 0 x_2 1 f_1
x_2 0 1 0 x_3 0 f_2
f_1 0 1 1 f_2 1 f


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


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



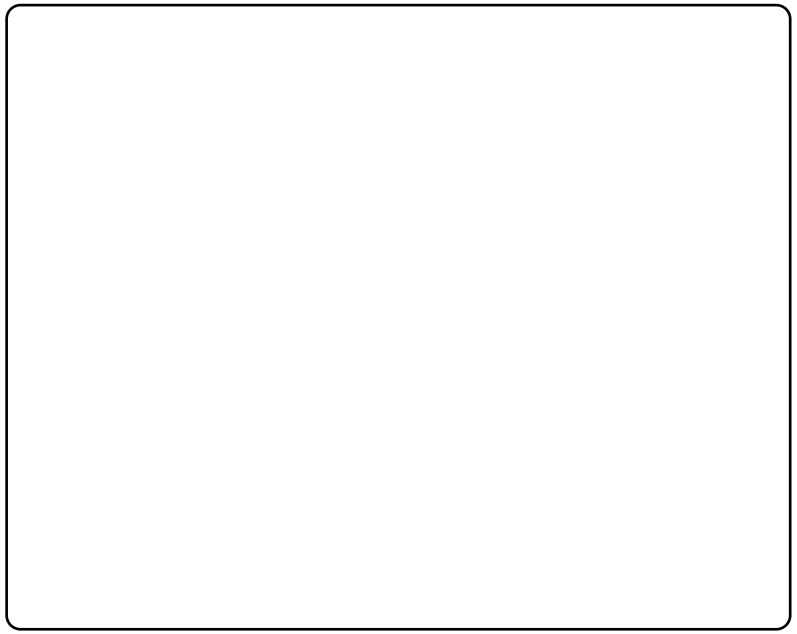
Синтез и верификация Verilog кода

Имплементация (размещение на кристалле)

Bitstream (генерация прошивки)

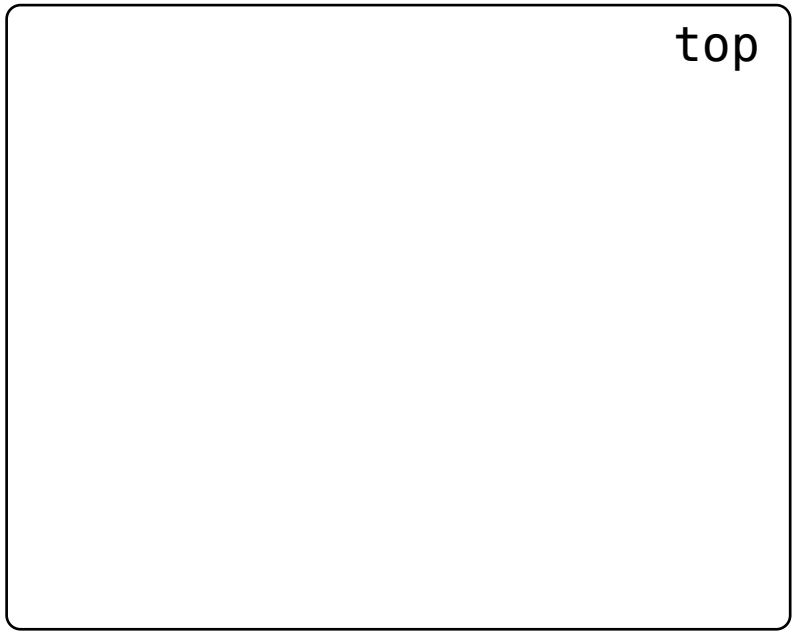
Verilog HDL

```
module
```

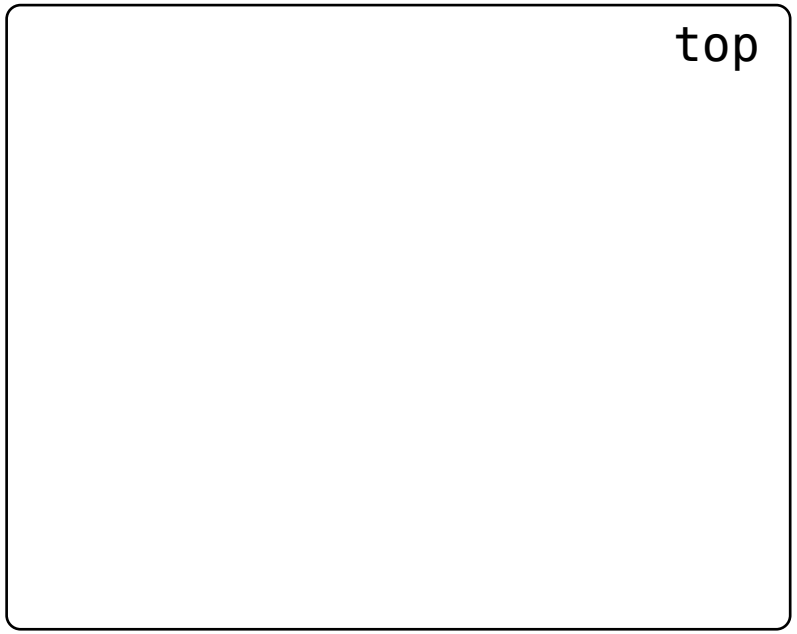


```
endmodule
```

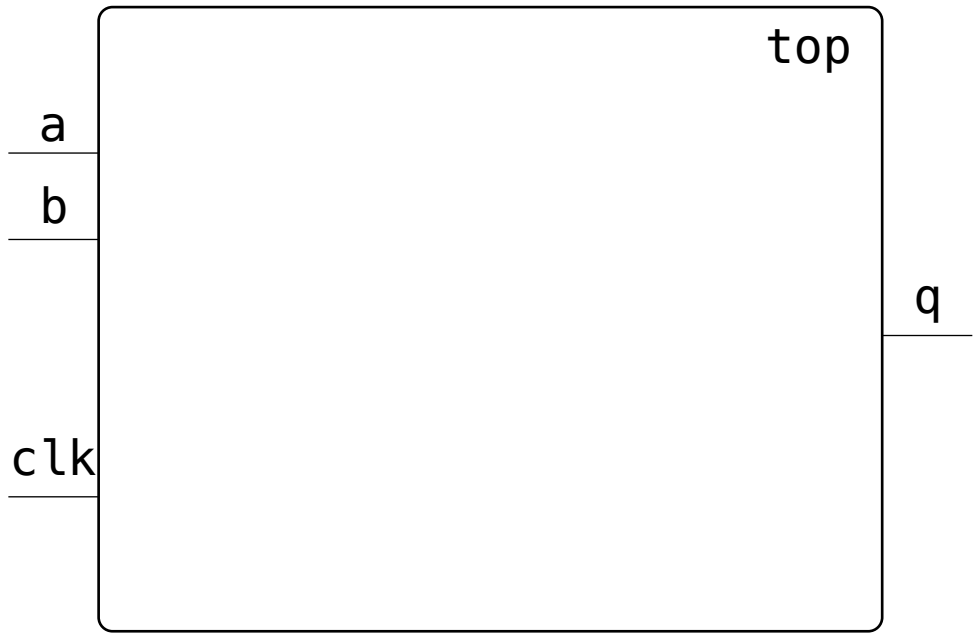
```
module top

endmodule
```



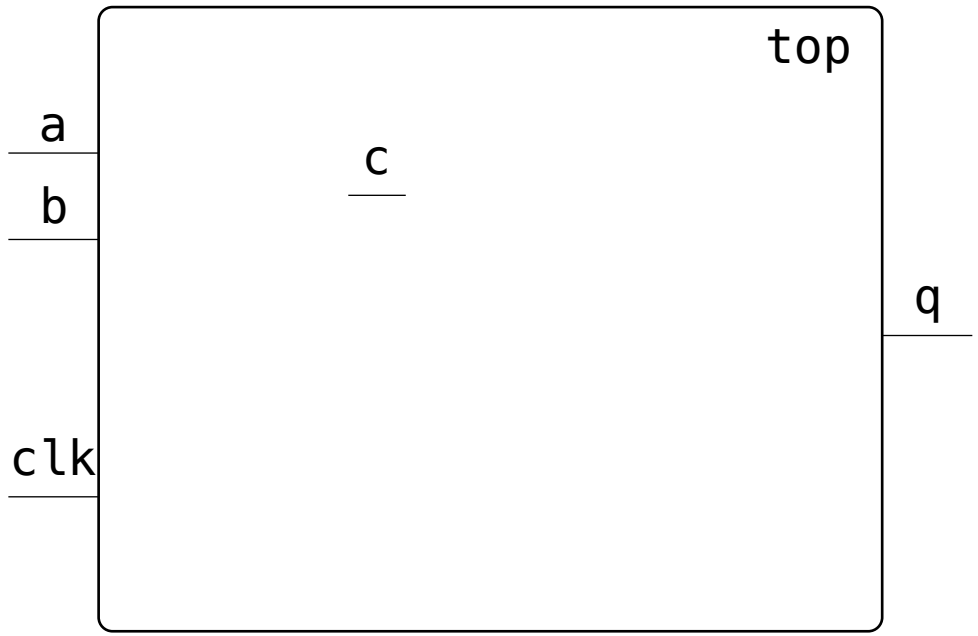
```
module top ();

endmodule
```



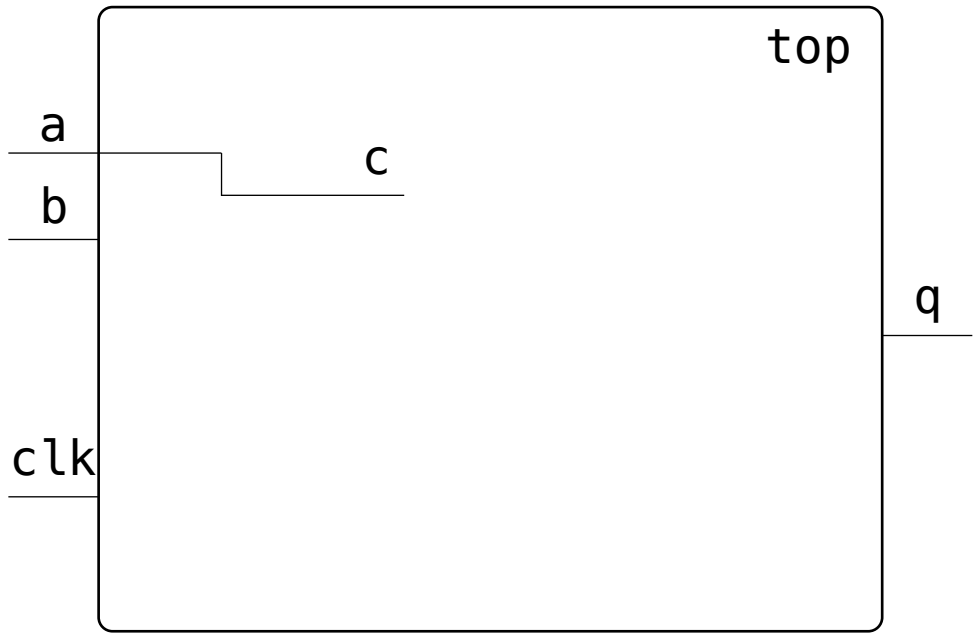
```
module top (
        input           a,
        input           b,
        input           clk,
        output          q
);


endmodule
```



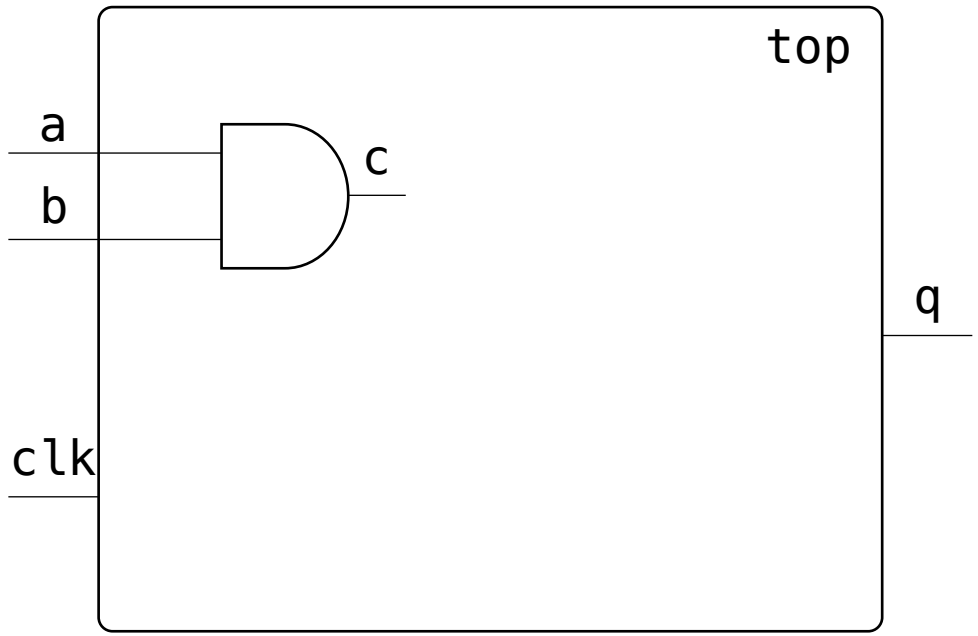
```
module top (
        input           a,
        input           b,
        input           clk,
        output          q
);

wire c;



endmodule
```



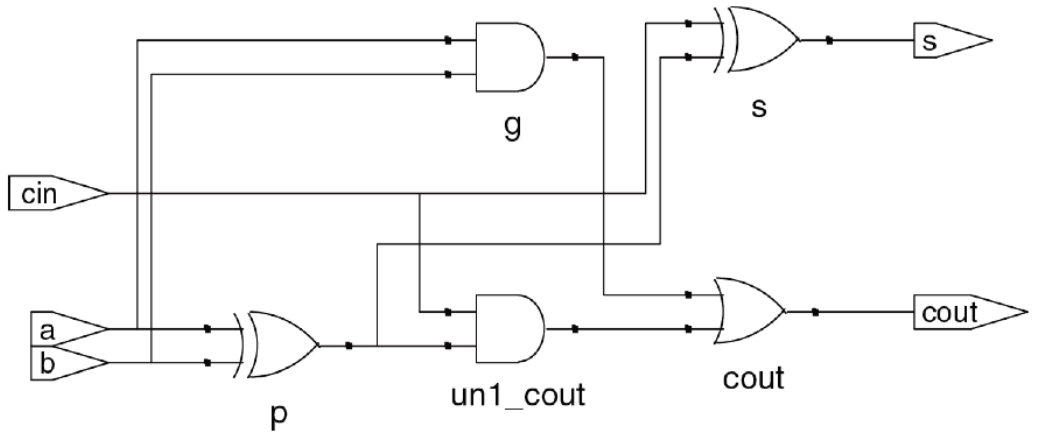
```
module top (
        input           a,
        input           b,
        input           clk,
        output          q
);

wire c;

assign c = a;



endmodule
```



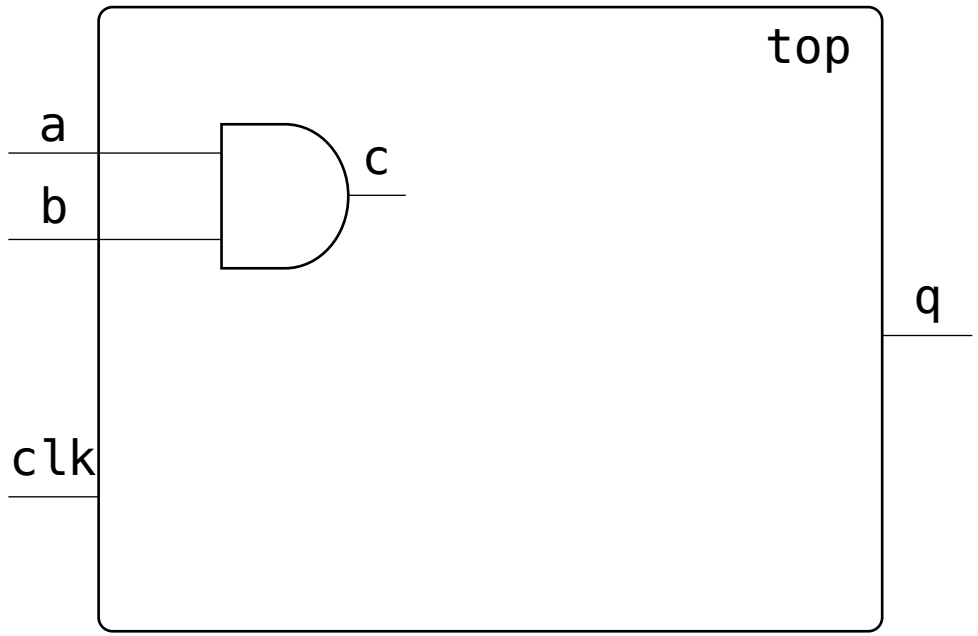
```
module top (
        input           a,
        input           b,
        input           clk,
        output          q
);

wire c;

assign c = a & b;



endmodule
```



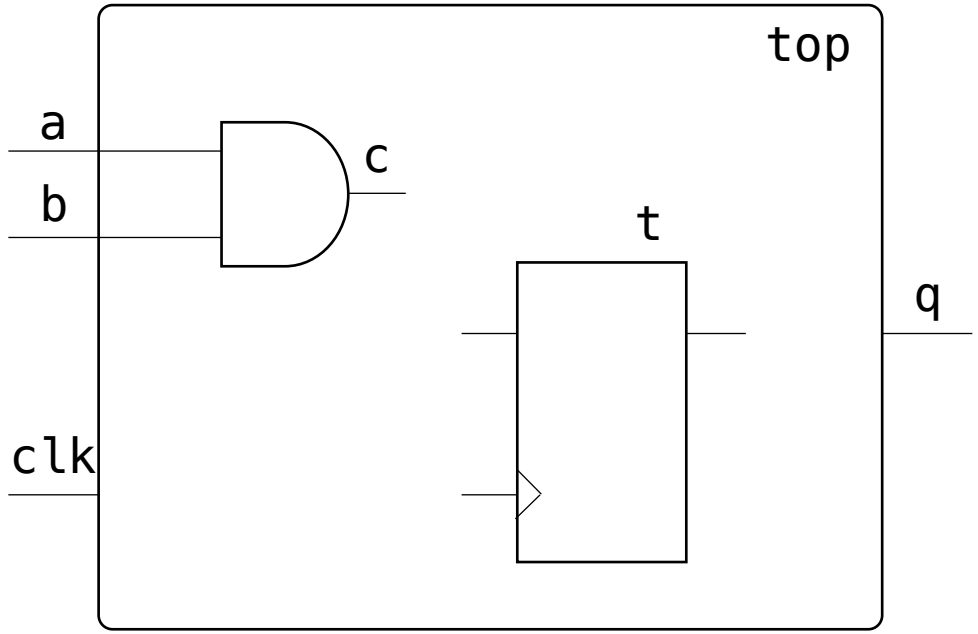
```
module fulladder (a, b, cin, s, cout);

       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
```



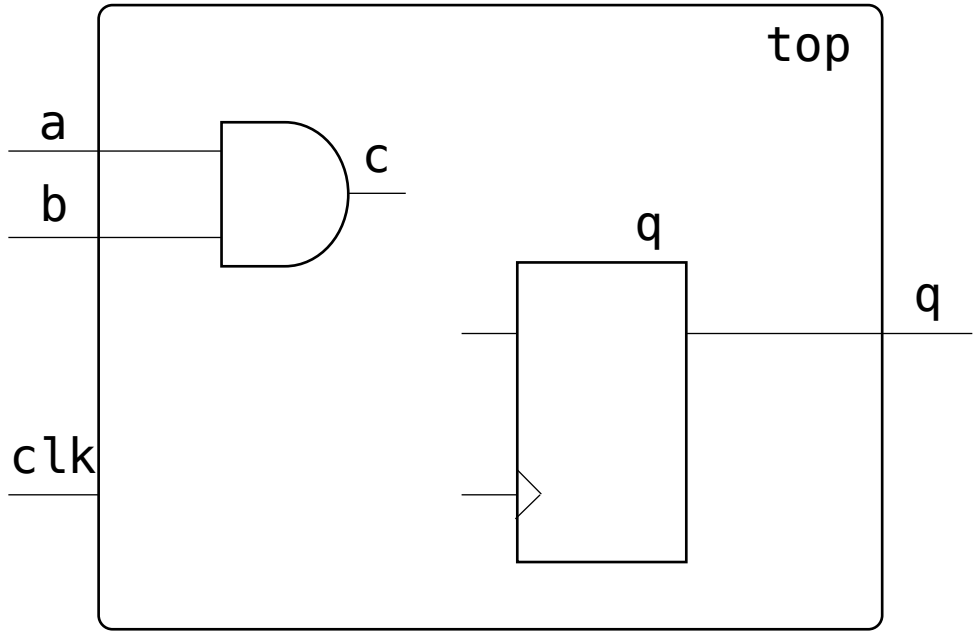
```
module top (
        input           a,
        input           b,
        input           clk,
        output          q
);

wire c;

assign c = a & b;



endmodule
```



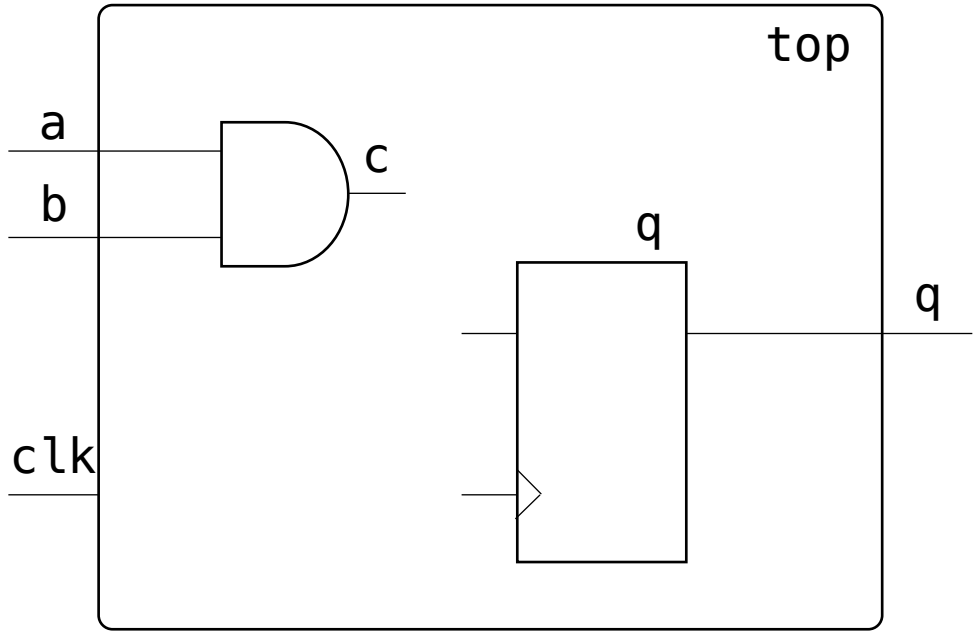
```
module top (
       input         a,
       input         b,
       input         clk,
       output        q
);

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



endmodule
```



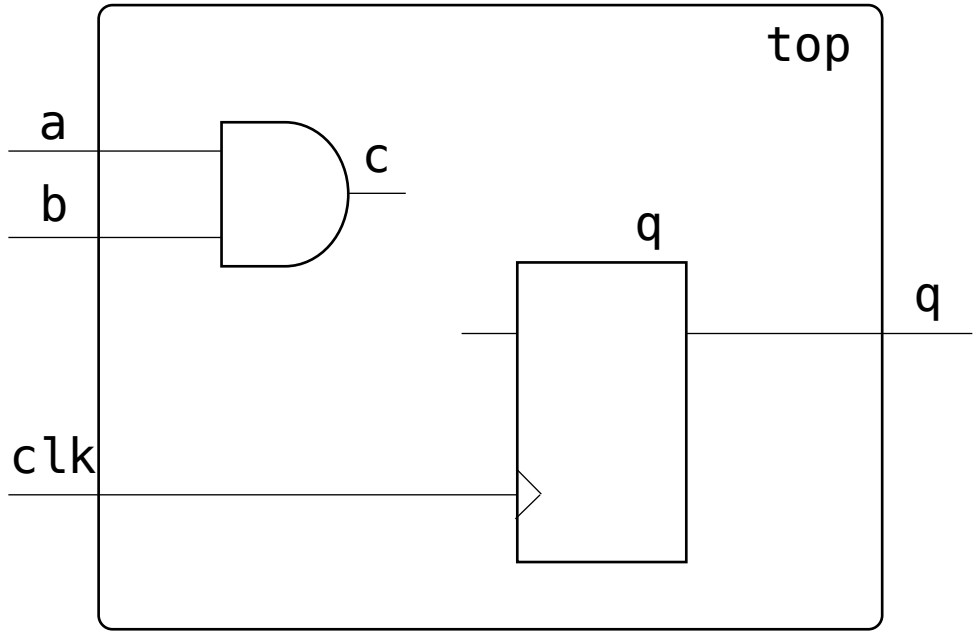
```
module top (
        input           a,
        input           b,
        input           clk,
        output  reg     q
);

wire c;

assign c = a & b;



endmodule
```



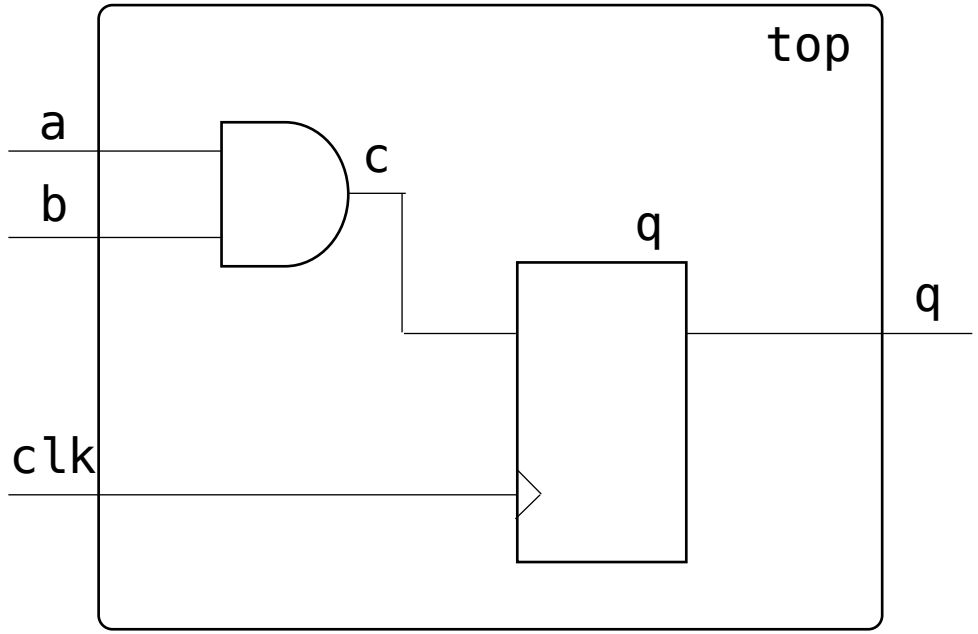
```
module top (
       input         a,
       input         b,
       input         clk,
       output  reg   q
);

wire c;

assign c = a & b;

always @ ()


endmodule
```



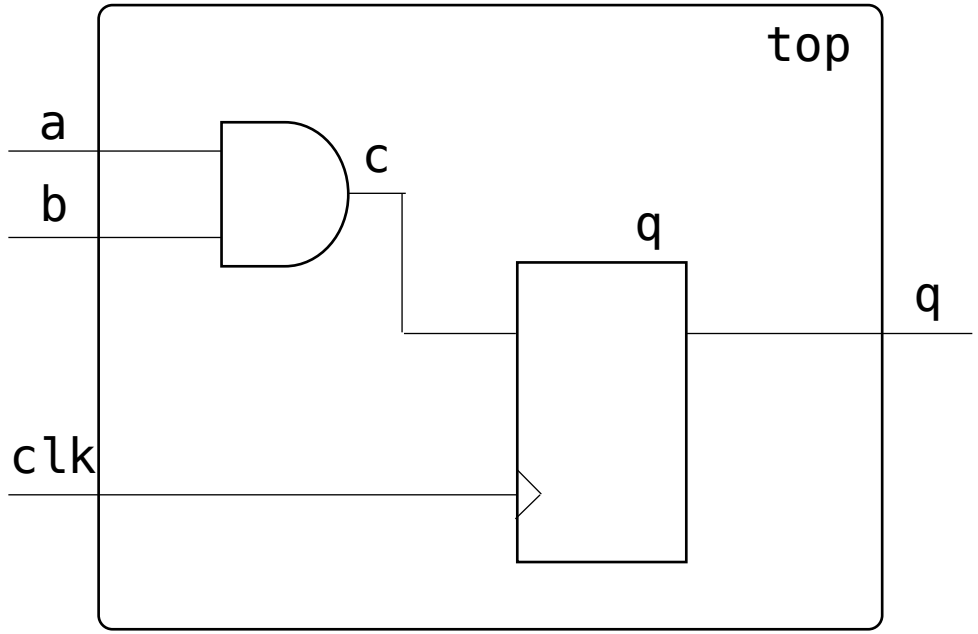
```
module top (
        input           a,
        input           b,
        input           clk,
        output  reg     q
);

wire c;

assign c = a & b;

always @ (posedge clk)


endmodule
```



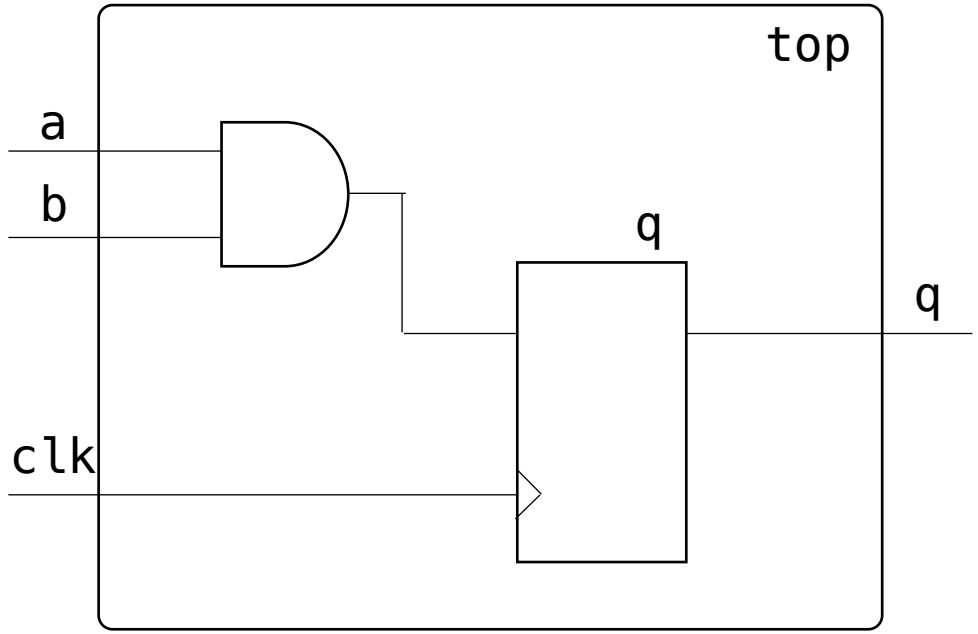
```
module top (
        input           a,
        input           b,
        input           clk,
        output  reg     q
);

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;

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
```



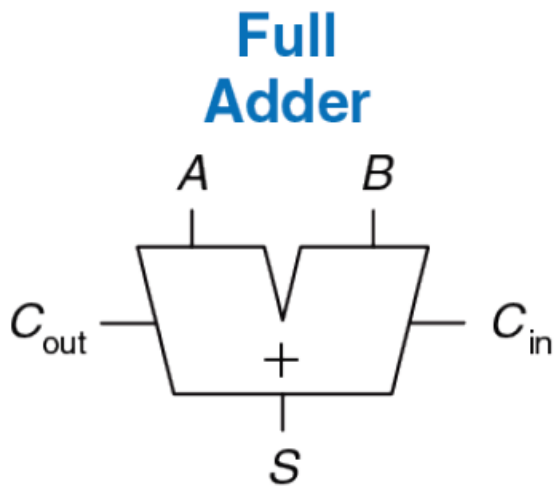
План лабораторной работы

- 1 пара
  - ~~О лабораторных работах (**T**)~~
  - ~~Введение в FPGA и Verilog HDL (**T**)~~
  - Тренинг по Vivado и Verilog HDL (**TS**)
  - Обзор отладочного стенда (**TS**)
- 2 пара
  - Теория Сумматор (**T**)
  - Описание сумматора на Verilog HDL (**S**)
  - Реализация сумматора на отладочном стенде (**S**)

План лабораторной работы

- ~~1 пара~~
  - ~~О лабораторных работах (**T**)~~
  - ~~Введение в FPGA и Verilog HDL (**T**)~~
  - ~~Тренинг по Vivado и Verilog HDL (**TS**)~~
  - ~~Обзор отладочного стенда (**TS**)~~
- 2 пара
  - Теория Сумматор (**T**)
  - Описание сумматора на Verilog HDL (**S**)
  - Реализация сумматора на отладочном стенде (**S**)

Сумматор

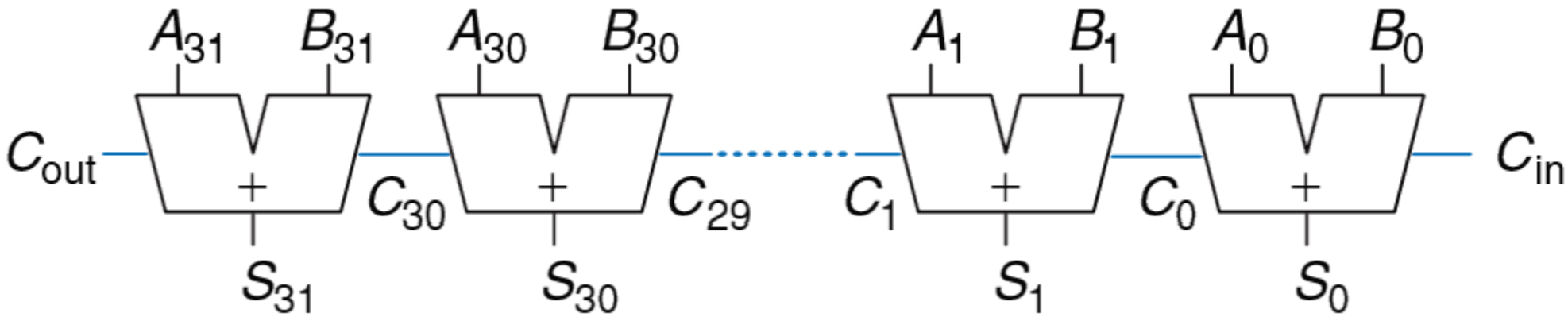


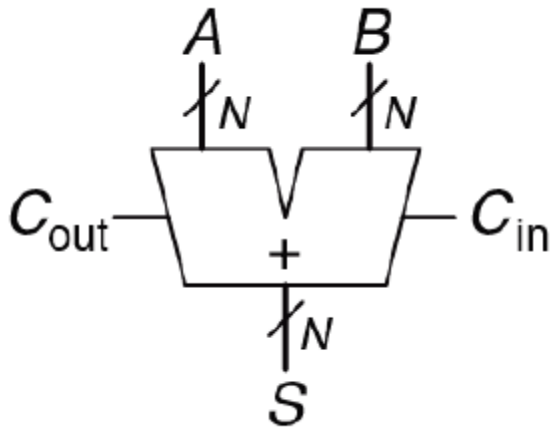
C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + AC_{in} + BC_{in}$$

Сумматор

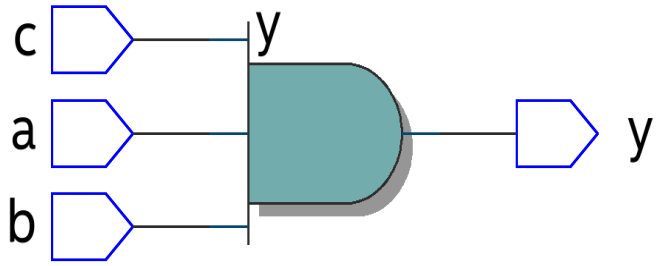

A_{31}
B_{31}
A_{30}
B_{30}
A_1
B_1
A_0
B_0
C_{out}
C_{30}
C_{29}
C_1
C_0
C_{in}
S_{31}
S_{30}
S_1
S_0

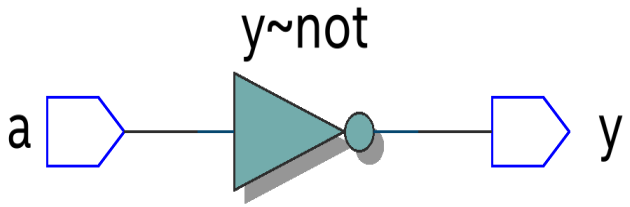

A
B
N
N
C_{out}
C_{in}
N
S

Иерархия модулей в Verilog

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

assign y = a & b & c;

endmodule
```

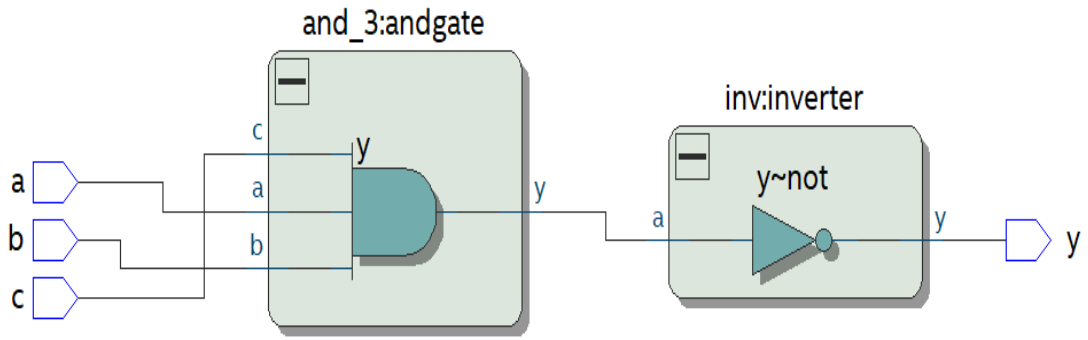
```
module inv (
       input  a
       output y
);

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

Generate

```
module add_gen (
        input   [2:0]   A,
        input   [2:0]   B,
        output  [2:0]   S
);

genvar i;
generate
        for (i=0; i<3; i=i+1) begin : newgen
                adder new (
                        .a(A[i]),
                        .b(B[i]),
                        .s(S[i])
                );
        end
endgenerate

endmodule
```

≡

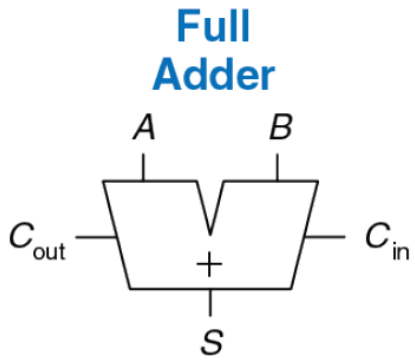
```
adder new0 (
        .a(A[0]),
        .b(B[0]),
        .s(S[0])
);

adder new1 (
        .a(A[1]),
        .b(B[1]),
        .s(S[1])
);

adder new2 (
        .a(A[2]),
        .b(B[2]),
        .s(S[2])
);
```

План лабораторной работы

- ~~1 пара~~
  - ~~О лабораторных работах (**T**)~~
  - ~~Введение в FPGA и Verilog HDL (**T**)~~
  - ~~Тренинг по Vivado и Verilog HDL (**TS**)~~
  - ~~Обзор отладочного стенда (**TS**)~~
- 2 пара
  - ~~Теория Сумматор (**T**)~~
  - Описание сумматора на Verilog HDL (**S**)
  - Реализация сумматора на отладочном стенде (**S**)

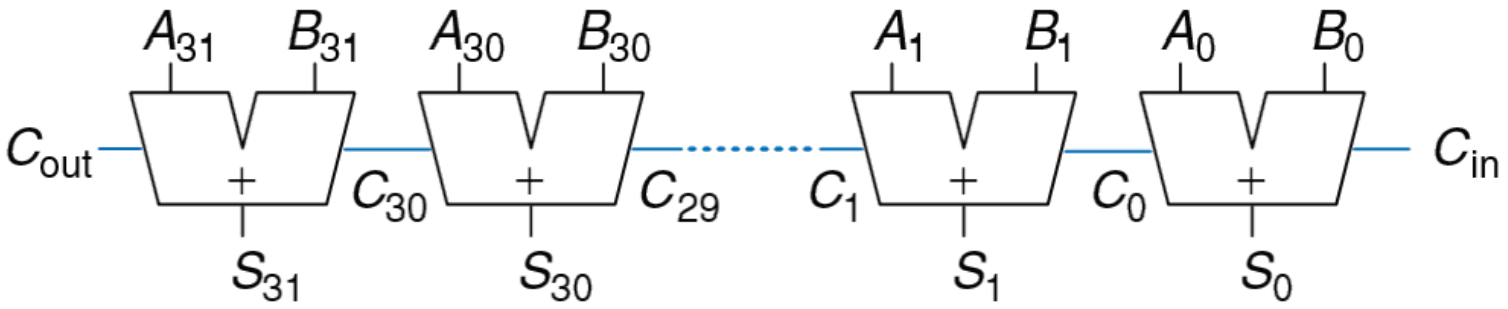
Задание

1. Реализовать полный сумматор на Verilog HDL
2. На основе разработанного полного сумматора реализовать 32-битный сумматор



$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + AC_{in} + BC_{in}$$



План лабораторной работы

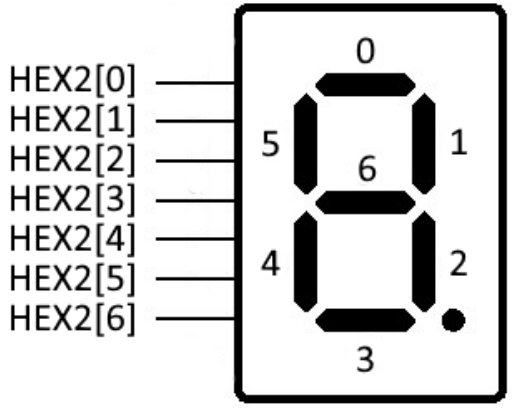
- ~~1 пара~~
  - ~~О лабораторных работах (**T**)~~
  - ~~Введение в FPGA и Verilog HDL (**T**)~~
  - ~~Тренинг по Vivado и Verilog HDL (**TS**)~~
  - ~~Обзор отладочного стенда (**TS**)~~
- 2 пара
  - ~~Теория Сумматор (**T**)~~
  - ~~Описание сумматора на Verilog HDL (**S**)~~
  - Реализация сумматора на отладочном стенде (**S**)

Задание

- Внедрить сумматор в отладочный стенд. В качестве входных данных и сигналов управления использовать переключатели switch на стенде. Результат выводить на светодиоды

Задание со звездочкой

```
module mission_1 (
        input    [9:0] SW,
        output   [6:0] HEX2
);

// реализовать модуль управления
// семисегментными индикаторами


endmodule
```



SW[3:0] = 4'b0101 ⟶ HEX2 = 7'b0010010