AQUARIUS

Комбинационные схемы. Основы SystemVerilog.

Содержание лекции

Маршрут проектирования ИМС

Языки описания аппаратуры

HDL – Hardware Description Language – язык описания аппаратуры.

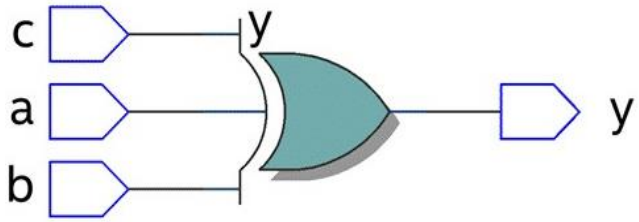
HDL Не являются языками программирования

Основное отличие – **параллельное** исполнение блоков кода, а не последовательное.

1 **RTL описание на языке Verilog HDL**

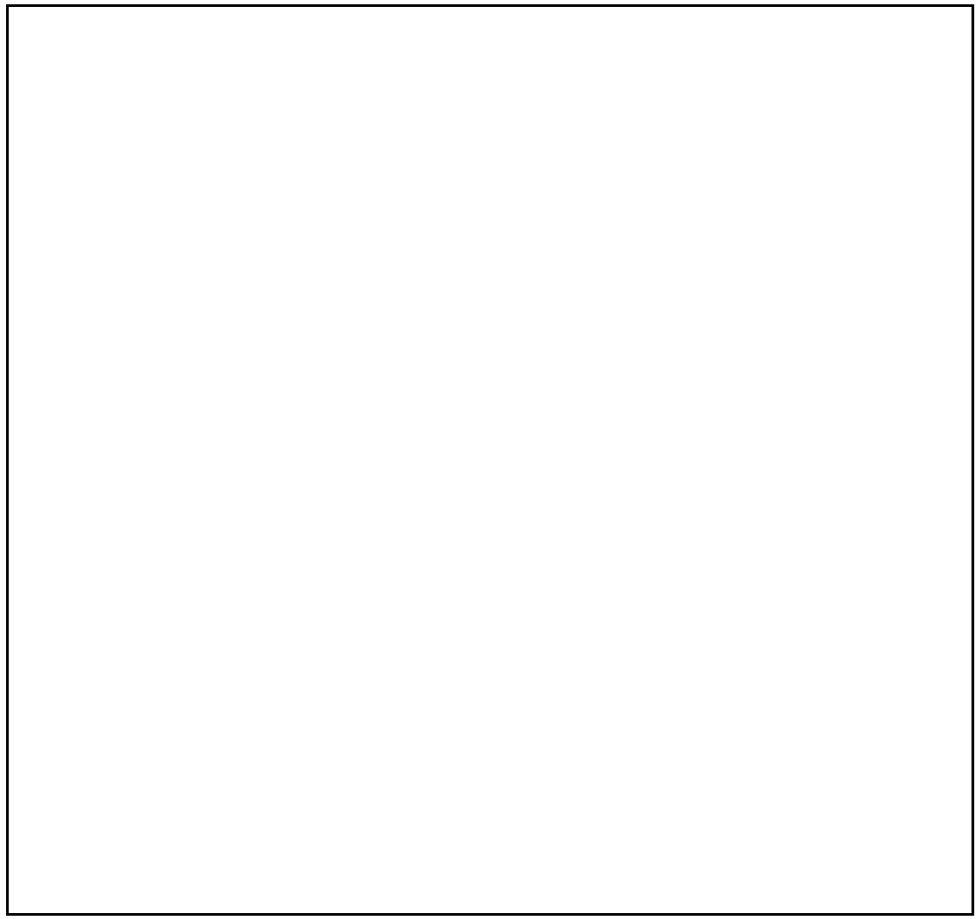
```
module xor3(
  input a, b, c,
  output reg y
);

  always @(a, b, c)
  begin
    y = a^b^c;
  end

endmodule
```

2 **Результат синтеза**



SystemVerilog HDL

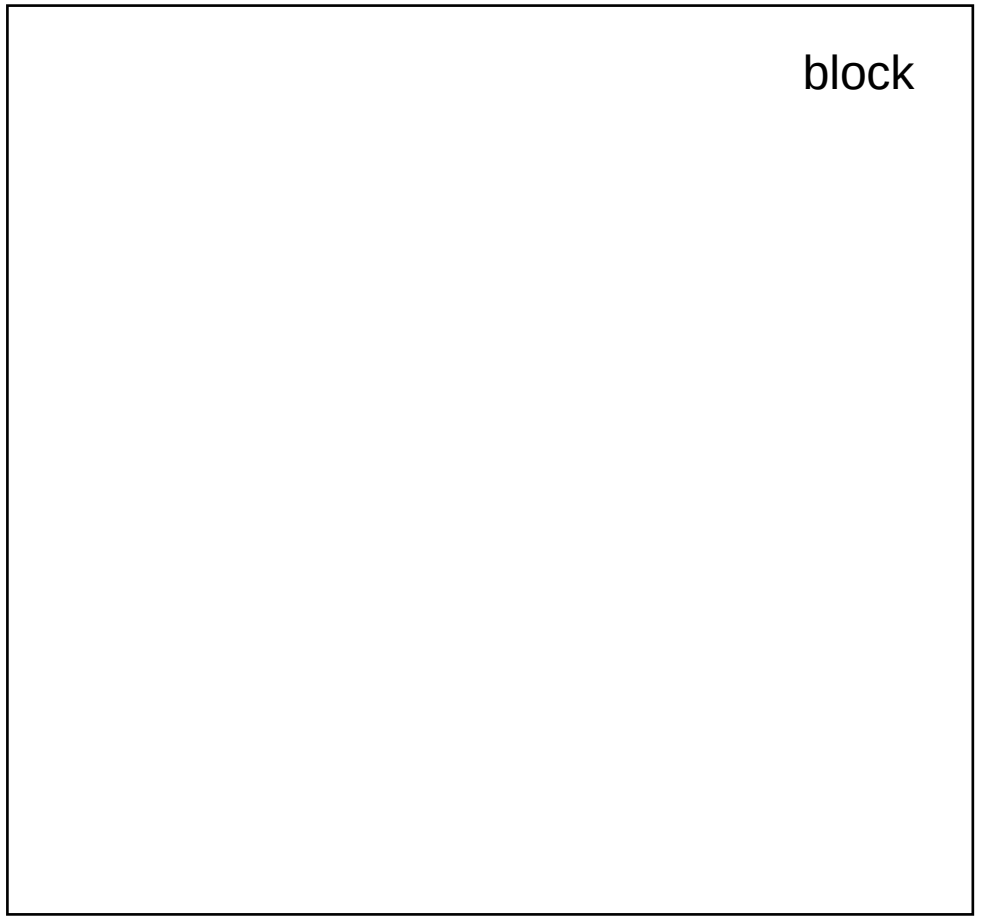
```
module
```



```
endmodule
```

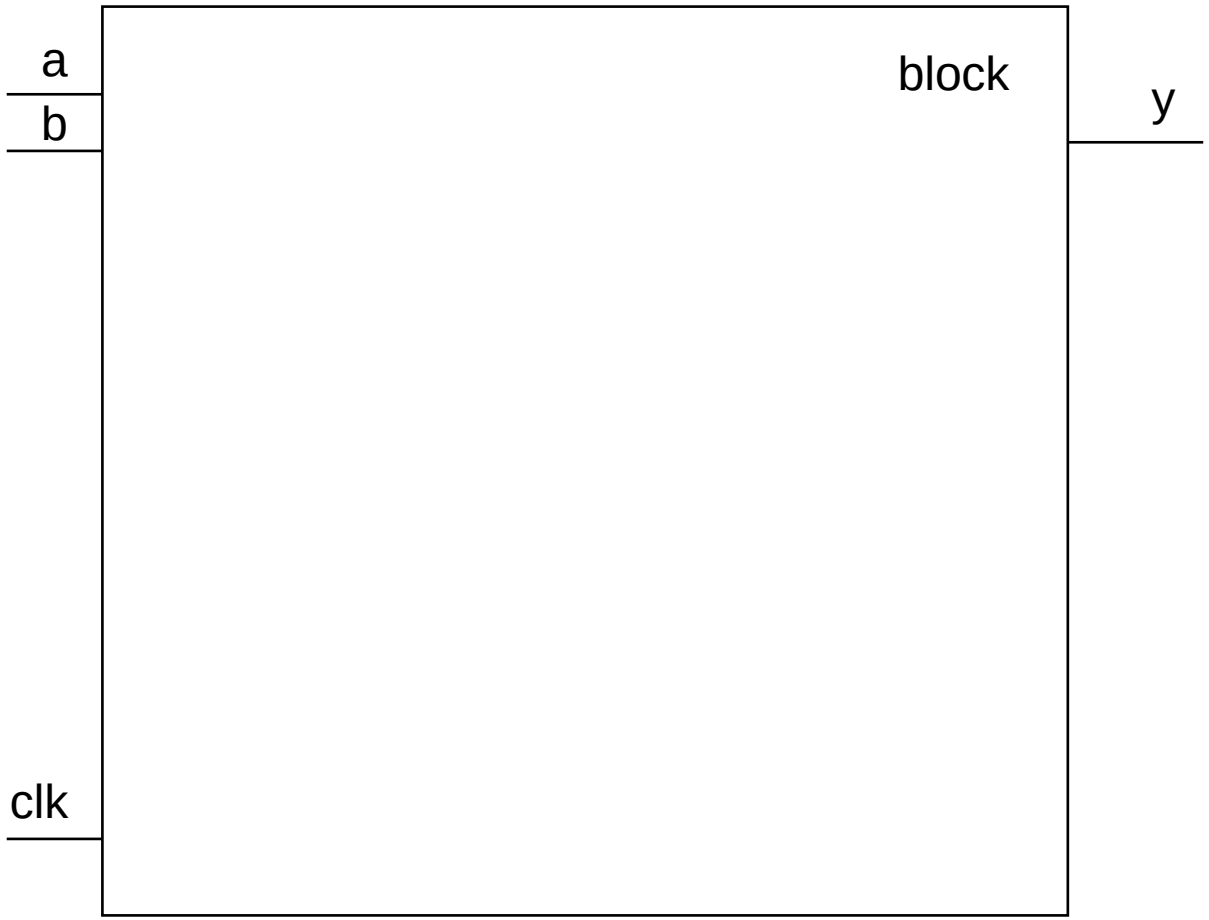
SystemVerilog HDL

```
module block

endmodule
```



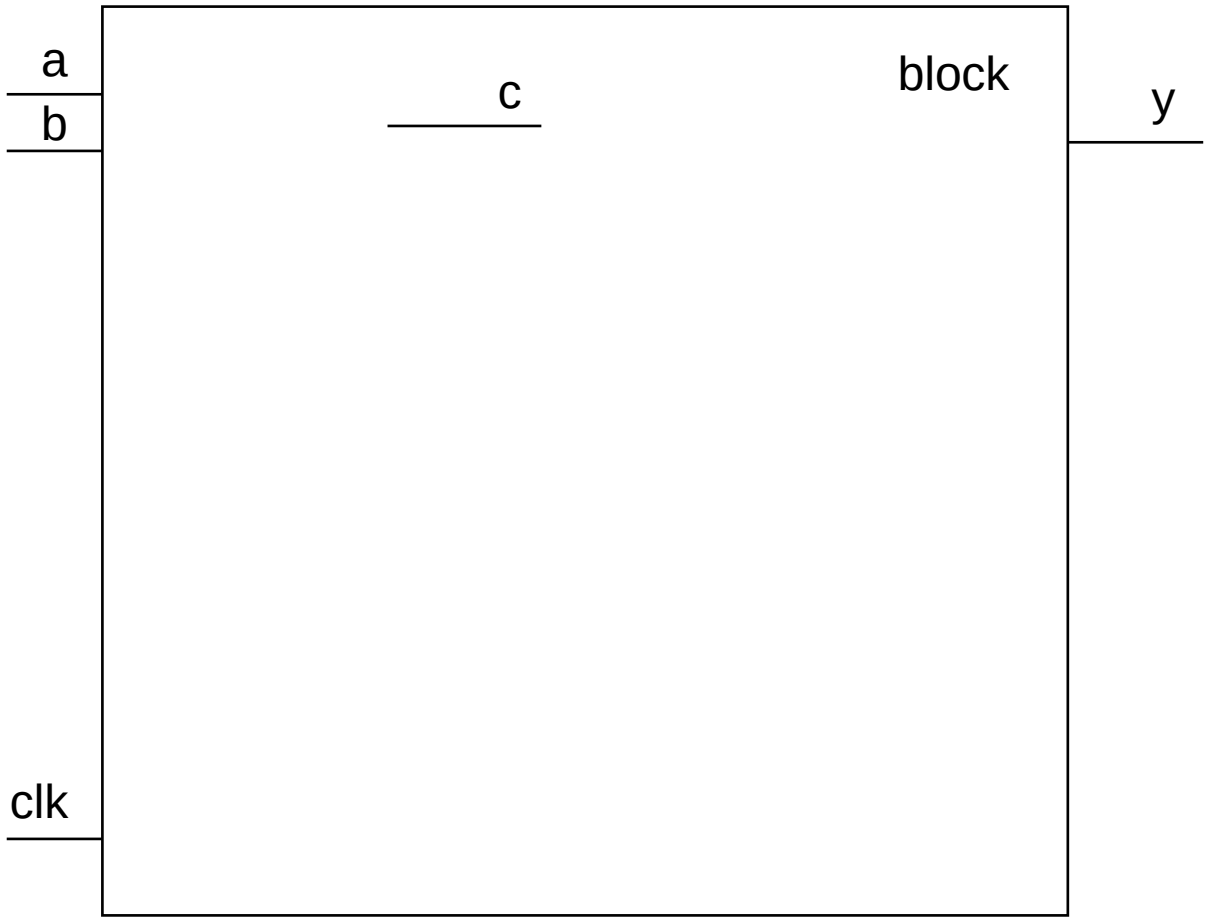
SystemVerilog HDL

```
module block (
  input clk,
  input a,
  input b,
  output y
);


endmodule
```



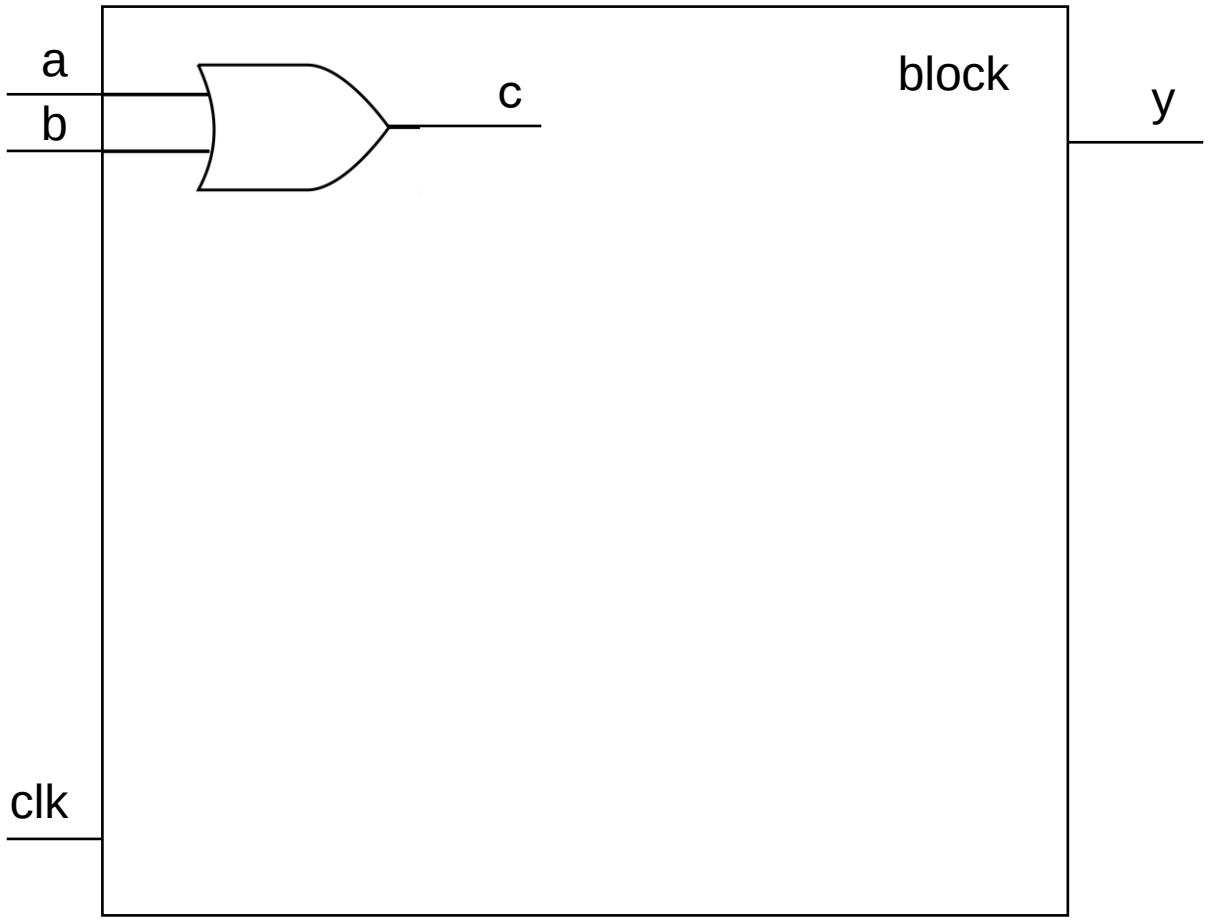
SystemVerilog HDL

```
module block (
  input clk,
  input a,
  input b,
  output y
);

  logic c; // комментарий



endmodule
```



SystemVerilog HDL

```
module block (
  input clk,
  input a,
  input b,
  output y
);

  logic c; // комментарий

  assign c = a|b;

endmodule
```



Вектор в SystemVerilog

Вектором называется одномерный массив элементов

Почему «вектор»? Потому что имеет упорядоченный набор элементов (компонент/координат) – битов

Синтаксис

```
<type> [msb:lsb] signal_name;
```

*msb – most significant bit (старший бит)

*lsb – less significant bit (младший бит)

Запись чисел в SystemVerilog

Вектор в SystemVerilog

Синтаксис

<type> [msb:lsb] signal_name;

*msb – most significant bit (старший бит)
*lsb – less significant bit (младший бит)

1 **Little-endian [bits]**

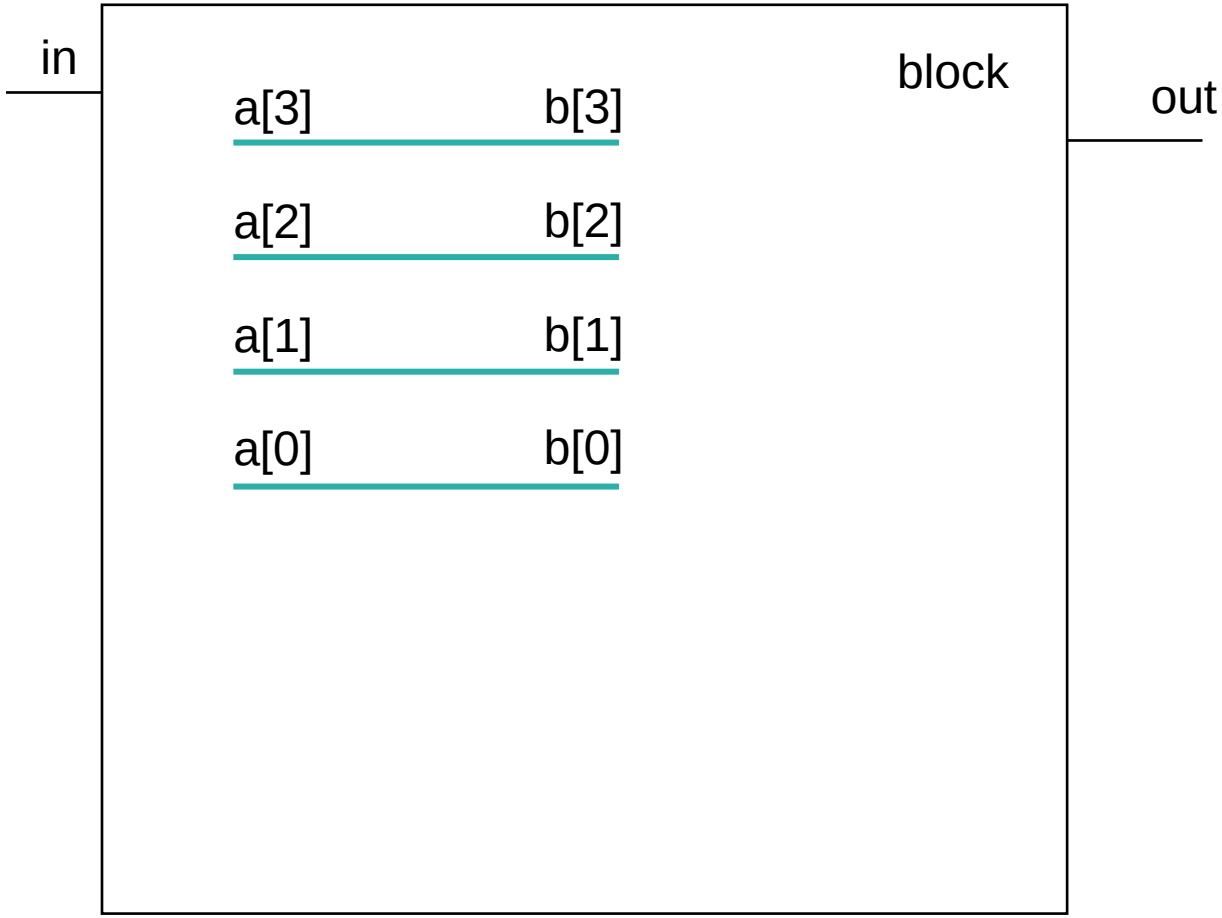
```
logic [7:0] signal_name;

assign signal_name = 8'b1;
// 8'b1 = 0 0 0 0 0 0 0 1

// signal_name[0] = 1
// signal_name[1..7] = 0
```

2 **Big-endian [bits] (лучше не использовать)**

```
logic [0:7] signal_name;

assign signal_name = 8'b1;
// 8'b1 = 0 0 0 0 0 0 0 1

// signal_name[0..6] = 0
// signal_name[7] = 1
```

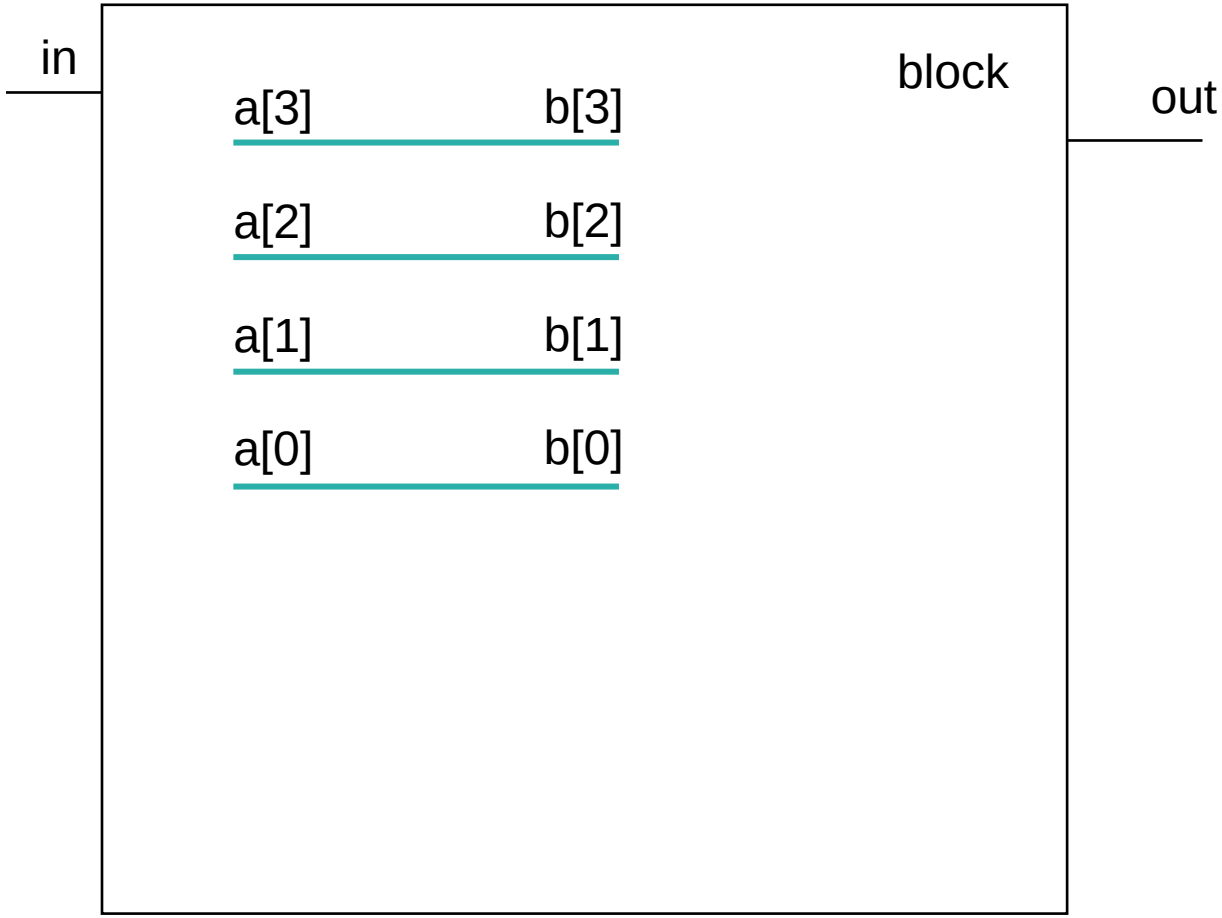
Вектор в SystemVerilog

```
module block(
  input in,
  output out
);
  logic [3:0] a; //4-х битная шина
  logic [3:0] b; //4-х битная шина

  assign a = b;
endmodule
```



Вектор в SystemVerilog

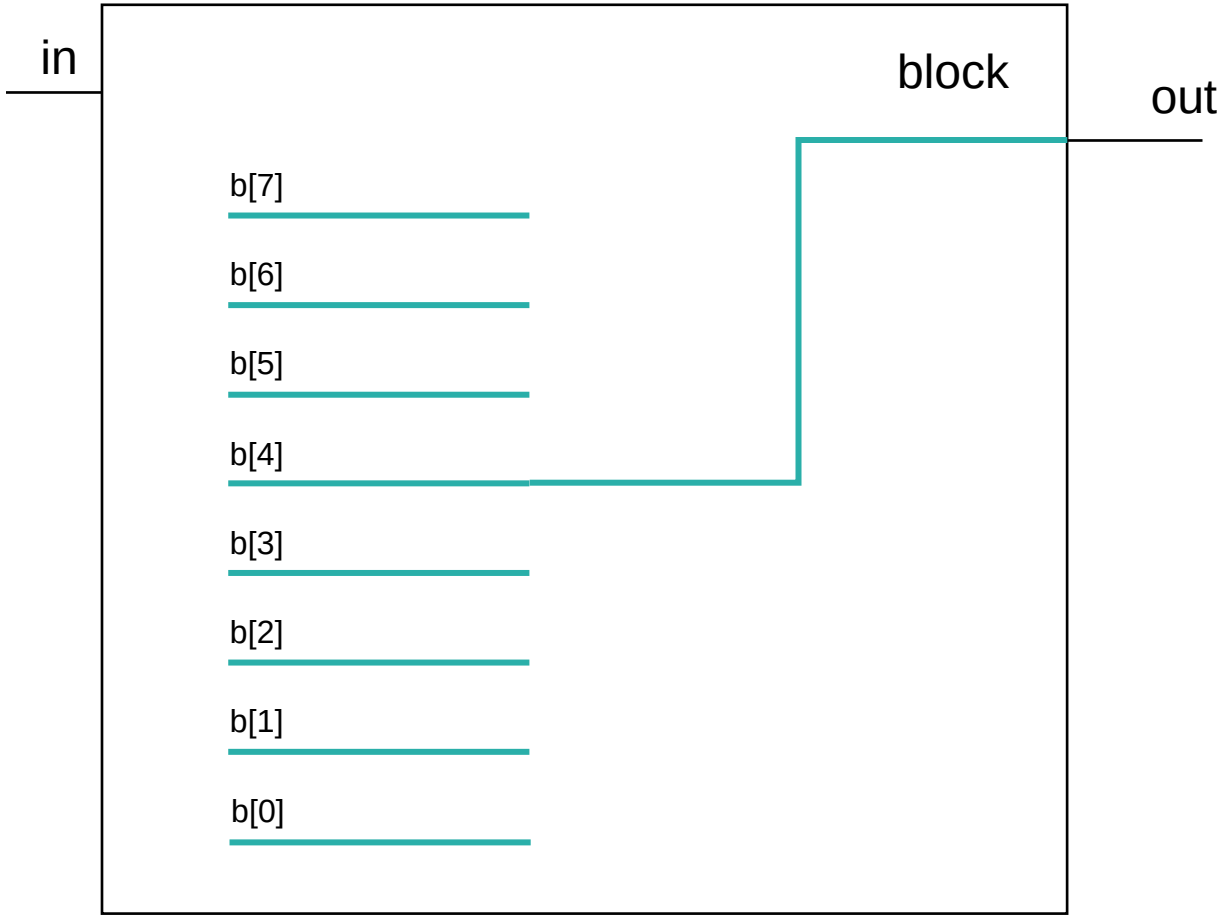
```
module block(
  input in,
  output out
);
  logic [3:0] a, b; //Две 4-х битные шины

  assign a = b;
endmodule
```



Вектор в SystemVerilog

Подключение одного провода из шины

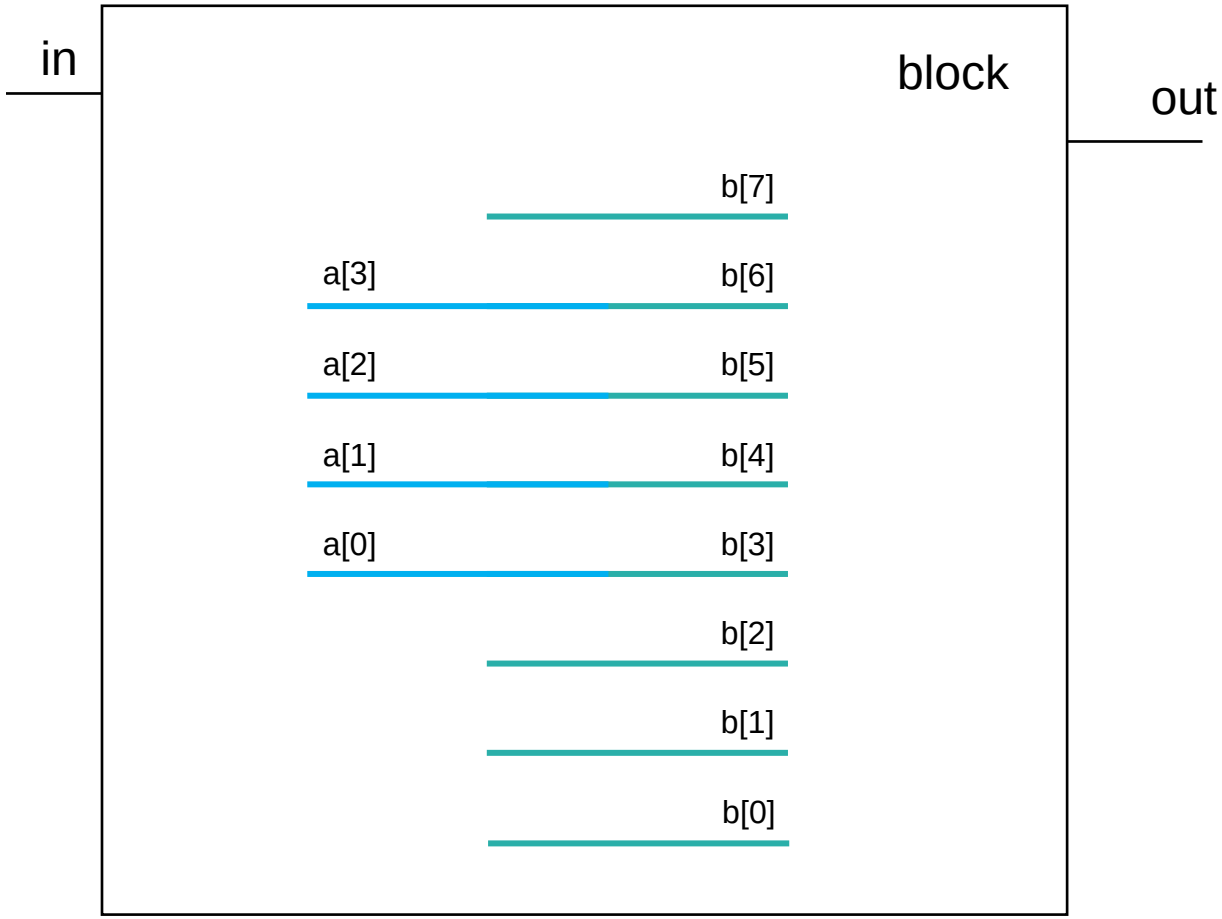
```
module block(
  input in,
  output out
);
  logic [3:0] a;
  logic [7:0] b;

  assign out = b[4];
endmodule
```



Вектор в SystemVerilog

Присвоение диапазона значений

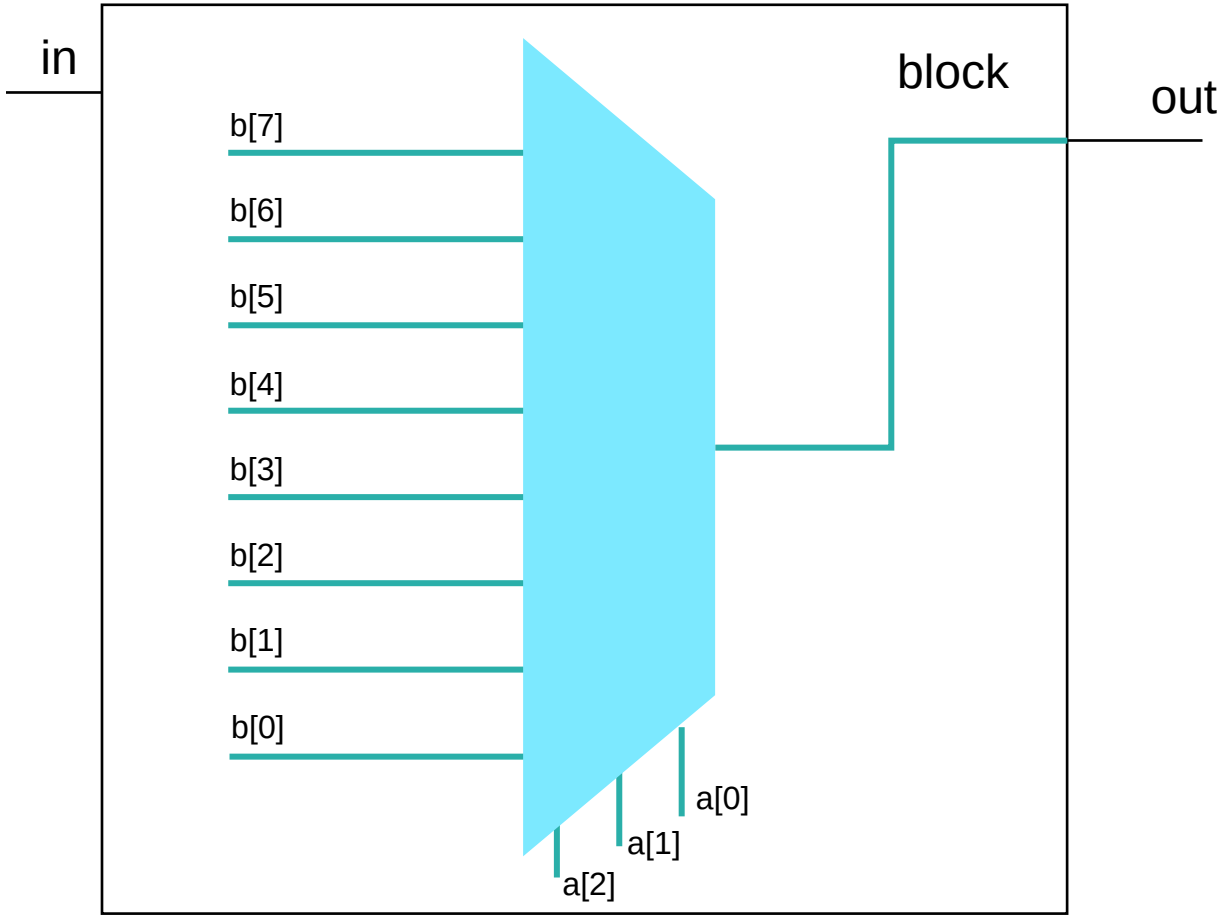
```
module block(
  input in,
  output out
);
  logic [3:0] a;
  logic [7:0] b;

  assign a = b[6:3];
endmodule
```



Вектор в SystemVerilog

Присвоение значения, зависящего от другого

```
module block(
  input in,
  output out
);
  logic [2:0] a;
  logic [7:0] b;

  assign out = b[a];
endmodule
```



Векторы и массивы в SystemVerilog

Многоразрядные входы/выходы и двумерный массив

```
module block (
  input clk,
  input [7:0] a, // многоразрядный вход
  input b,
  output [3:0] y // многоразрядный выход
);

  logic [3:0] c [22];
  logic [7:0] d [0:21]; // двумерный массив 8-ми разрядных шин

  assign y = a[7:4];

endmodule
```

Основные операции в Verilog

Побитовые операции

Синтаксис	Операция
~	Инверсия
&	И
|	ИЛИ
^	Исключающее ИЛИ (XOR)
~^ или ^~	Инверсное исключающее ИЛИ
<<	Логический сдвиг влево (пустые биты заполняются нулями)
>>	Логический сдвиг вправо (пустые биты заполняются нулями)

```
logic [7:0] a, b, c;
assign a = ~b;   // Инверсия каждого бита шины b
assign a = b&c;  // Операция И над каждым соответствующим
                 // битом b и c
assign a = b|c;  // Операция ИЛИ над каждым соответствующим
                 // битом b и c
assign a = b^c;  // Побитовый XOR
assign a = b~^c; // Побитовый XNOR
assign a = b<<3; // Сдвиг b на 3 бита влево
                 // b = 8'b0000_1111 -> a = 8'b0111_1000
assign a = b>>2; // Сдвиг b на 2 бита вправо
                 // b = 8'b0000_1111 -> a = 8'b0000_0011
```

Основные операции в Verilog

Операции свертки

Синтаксис	Операция
&	И над всеми битами вектора (1-битный результат)
~&	И-НЕ над всеми битами вектора (1-битный результат)
|	ИЛИ над всеми битами вектора (1-битный результат)
~|	ИЛИ не над всеми битами вектора (1-битный результат)
^	Исключающее ИЛИ над всеми битами вектора (1-битный результат)
~^ или ~^	Инверсное исключающее ИЛИ над всеми битами вектора (1-битный результат)

```
logic a;
logic [7:0] b;
assign a = &b;  // a = b[0]&b[1]&b[2]&...&b[7]
assign a = ~&b;
assign a = |b;  // a = b[0]|b[1]|b[2]|...|b[7]
assign a = ~|c;
assign a = ^c;
assign a = ~^c;
```

Основные операции в Verilog

Булевы операции

Синтаксис	Операция
!	Отрицание
&&	Булева И
||	Булева ИЛИ

```
logic a;
logic [7:0] b,c;
assign a = !b; // ИСТИНА, если все биты в b равны 0, иначе ЛОЖЬ
assign a = b && c; // ИСТИНА, если побитовая опреация И над
                   // векторами b и c дает все единицы, иначе ЛОЖЬ
assign a = b || c; // ИСТИНА, если побитовая опреация ИЛИ над
                   // векторами b и c дает все единицы, иначе ЛОЖЬ
```

Основные операции в Verilog

Операции логического сравнения

Синтаксис	Операция
==	Равенство
!=	Неравенство
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно

```
logic a;
bit [7:0] b,c;
assign a = (b == c);
assign a = (b != c);
assign a = (b < c);
assign a = (b > c);
assign a = (b <= c);
assign a = (b >= c);
```

*Примечание:
Результаты данных операций
справедливы для переменных,
имеющих определенное состояние.

Основные операции в Verilog

Арифметические операции

Синтаксис	Операция
+	Сложение
-	Вычитание
-	Отрицательное число (когда знак стоит перед числом)
*	Умножение
/	Деление
%	Остаток от деления
**	Возведение в степень
<<<	Арифметический сдвиг влево (заполнение пустых битов нулями)
>>>	Арифметический сдвиг вправо (заполнение пустых битов битом знака)

Арифметические операции

```
logic [7:0] b;
/************************/
/*   Логический сдвиг   */
/************************/
assign a = b>>2; // Сдвиг b на 2 бита вправо
                 // b = 8'b0000_1111 -> a = 8'b0000_0011
                 // b = 8'b1111_0000 -> a = 8'b0011_1100
/************************/
/* Арифметический сдвиг */
/************************/
assign a = b>>>2; // Сдвиг b на 2 бита вправо
                  // b = 8'b0000_1111 -> a = 8'b0000_0011
                  // b = 8'b1111_0000 -> a = 8'b1111_1100
```

Арифметические операции

Сложение и вычитание чисел без знака

1 Числа без знака

```
logic [7:0] a, b;
logic [8:0] sum, dif;

assign sum = a+b;
assign dif = a-b;
```

Какое максимальное значение может иметь вектор a? 255

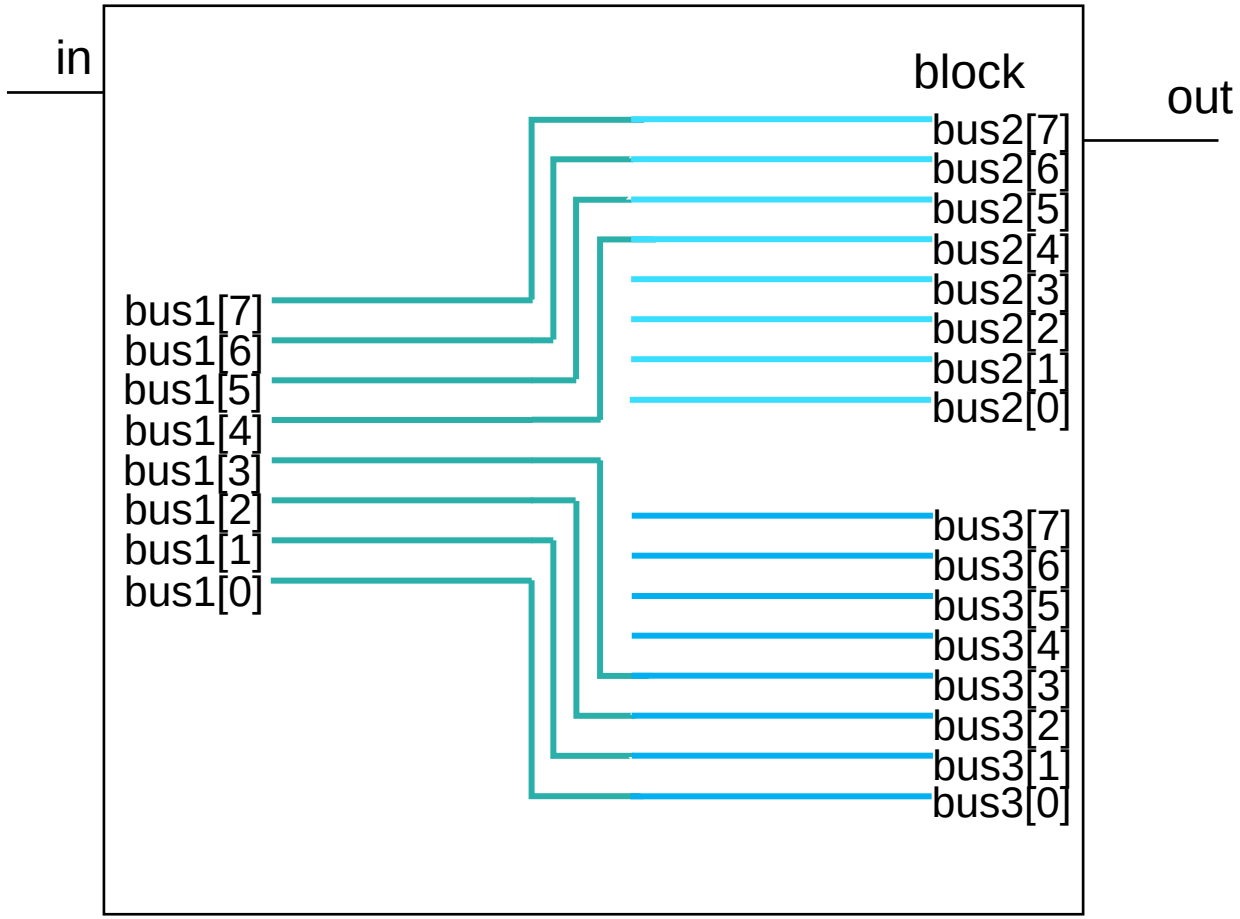
2 Числа со знаком

```
logic signed [7:0] a, b;
logic signed [8:0] sum, dif;

assign sum = a+b;
assign dif = a-b;
```

Какое максимальное значение может иметь вектор a? 127

Конкатенация {}

```
logic[7:0] bus1, bus2, bus3;
assign bus1 = {bus2[7:4], bus3[3:0]};

logic[7:0] busC;
logic [3:0] busA, busb;

assign busC = {busA, busB}; // busC должна
                            //иметь 8 бит
assign busC = {4'b0000, busA};
```

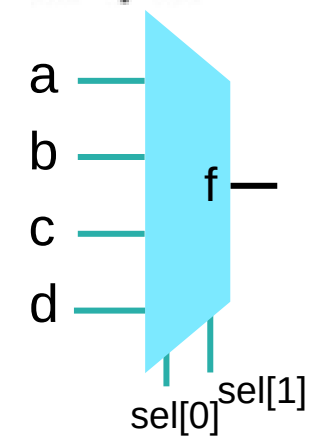


Тернарный оператор « ? : »

Синтаксис

```
assign signal = (condition) ? if_true : if_false;
```

Sel	F
“00”	A
“01”	B
“10”	C
“11”	D

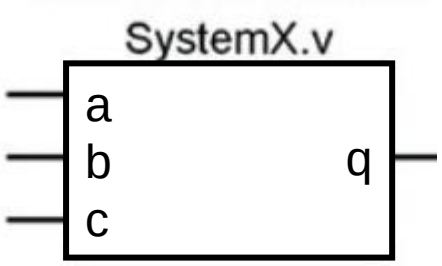



```
module mux_4to1 (
  output f,
  input a,b,c,d,
  input [1:0] sel
);
  assign f = (sel == 2'b00) ? a :
             (sel == 2'b01) ? b :
             (sel == 2'b10) ? c :
             (sel == 2'b11) ? d :
             1'bx;
endmodule
```

Блок always. Присвоение значения без assign

```
module top(
  input logic [7:0] a, // Входы и выходы
  output logic [3:0] q // с явным указанием
                       // типа
);

  always @(*) begin
    q = ~a[3:0];
  end

endmodule
```

If-else



a	b	c	q
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

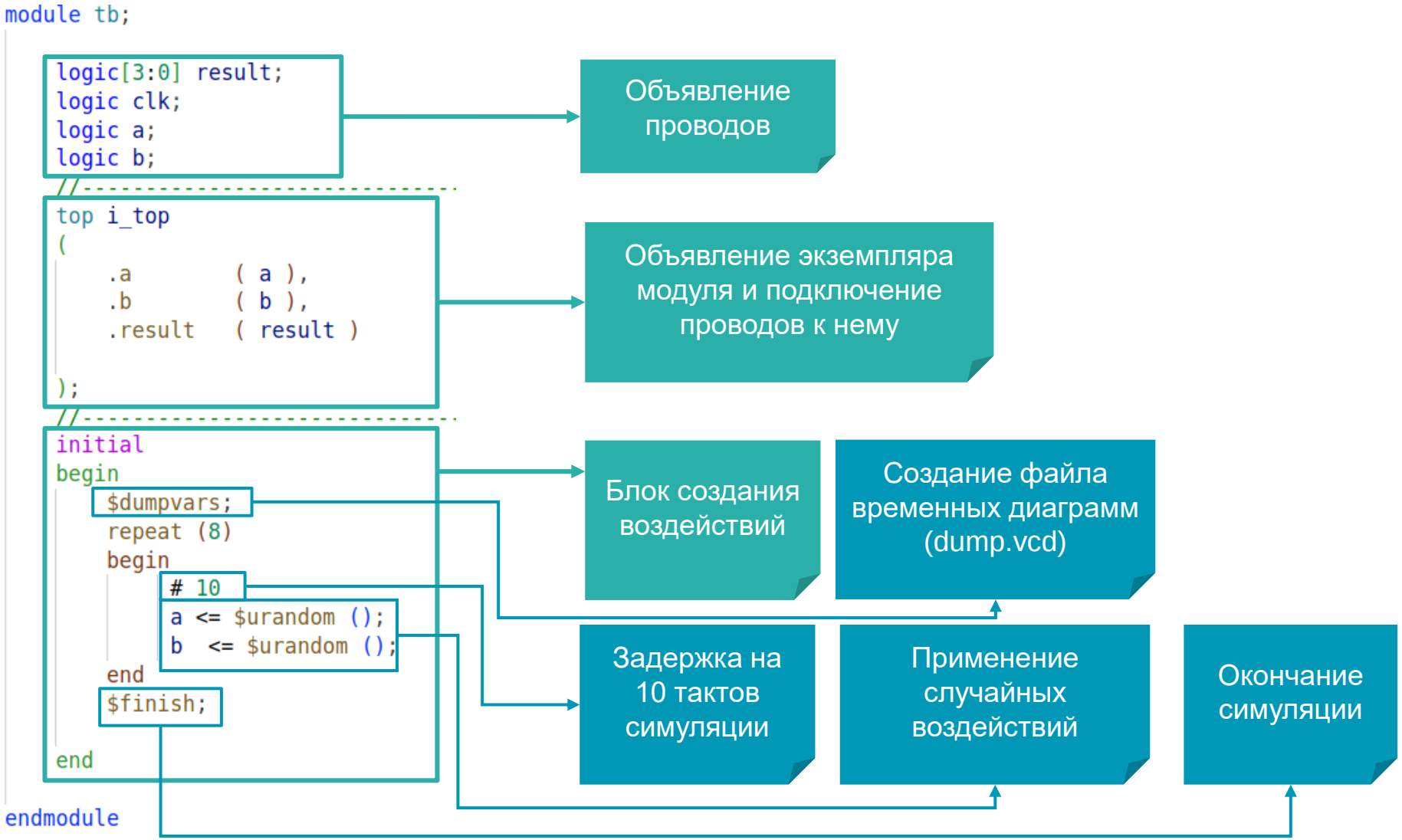
```
module SystemX(
  output logic q,
  input a,b,c
);
  always @(a,b,c) // Указан перечень сигналов чувствительности блока
                  // Обычно для обозначения всех сигналов применяется (*)
    begin
      if ((a == 1'b0) && (b == 1'b0) && (c == 1'b0))
        q = 1;
      else if ((a == 1'b0) && (b == 1'b1) && (c == 1'b0))
        q = 1;
      else if ((a == 1'b1) && (b == 1'b1) && (c == 1'b0))
        q = 1;
      else
        q = 0;
    end
endmodule
```

Конструкция case

Дешифраторы и мультиплексоры

```
module block (
  output[7:0] f,
  input [7:0] a,b,c,d,
  input [1:0] sel
);

  always_comb begin
    case (sel)
      2'b00: f = a;
      2'b01: f = b;
      default : f = c;
    endcase
  end
endmodule
```

Тестовое окружение



Особенность языка Verilog

Обратите внимание!

Код, кроме **assign** и объявления переменных, может исполняться только **внутри процедурных блоков.**

Вне процедурных блоков исполняемые части кода могут находиться в **task** и **function**, однако их исполнение происходит только в случае их вызова из процедурного блока.

```
module block (
    input in,
    output out
)

    logic a, b, c, d;

    a = 1;
    b = 0;

....

endmodule
```

```
module block (
    input in,
    output out
)

    logic a, b, c, d;

    assign a = 1;
    assign b = 0;

....

endmodule
```

```
module block (
    input in,
    output out
)

    logic a, b, c, d;

    initial begin
        a = 1;
        b = 0;
    end

....

endmodule
```

```
module block (
    input in,
    output out
)

    logic a, b, c, d;

    always @(*) //always_comb
    begin
        a = 1;
        b = 0;
    end

....

endmodule
```

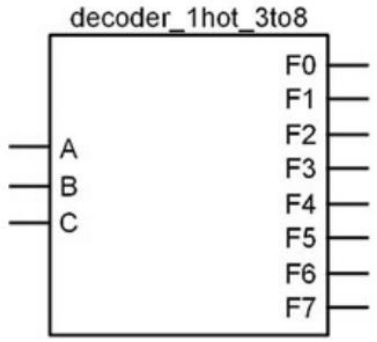
```
module block (
    input in,
    output out
)

    logic a, b, c, d;

    task set_a_b(logic set_a,logic set_b);
        a = set_a;
        b = set_b;
        #10;
    endtask

....
    initial begin
        set_a_b(1,0);
    end
endmodule
```

Упражнения с декодерами

Опишите декодер 3 в 8, используя конструкцию assign и логические операторы

1 Таблица истинности

A	B	C	F7	F6	F5	F4	F3	F2	F1	F0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0



2 ДНФ

$$F0 = \sum_{A,B,C}(0) = A'\cdot B'\cdot C'$$
$$F1 = \sum_{A,B,C}(1) = A'\cdot B'\cdot C$$
$$F2 = \sum_{A,B,C}(2) = A'\cdot B\cdot C'$$
$$F3 = \sum_{A,B,C}(3) = A'\cdot B\cdot C$$
$$F4 = \sum_{A,B,C}(4) = A\cdot B'\cdot C'$$
$$F5 = \sum_{A,B,C}(5) = A\cdot B'\cdot C$$
$$F6 = \sum_{A,B,C}(6) = A\cdot B\cdot C'$$
$$F7 = \sum_{A,B,C}(7) = A\cdot B\cdot C$$

3 RTL описание на языке Verilog HDL

```
module decoder_1_hot_3to8(
  output logic F0, F1, F2, F3, F4, F5, F6, F7,
  input logic A, B, C);

  assign F0= ~A & ~B & ~C;
  assign F1= ~A & ~B &  C;
  assign F2= ~A &  B & ~C;
  assign F3= ~A &  B &  C;
  assign F4=  A & ~B & ~C;
  assign F5=  A & ~B &  C;
  assign F6=  A &  B & ~C;
  assign F7=  A &  B &  C;

endmodule
```