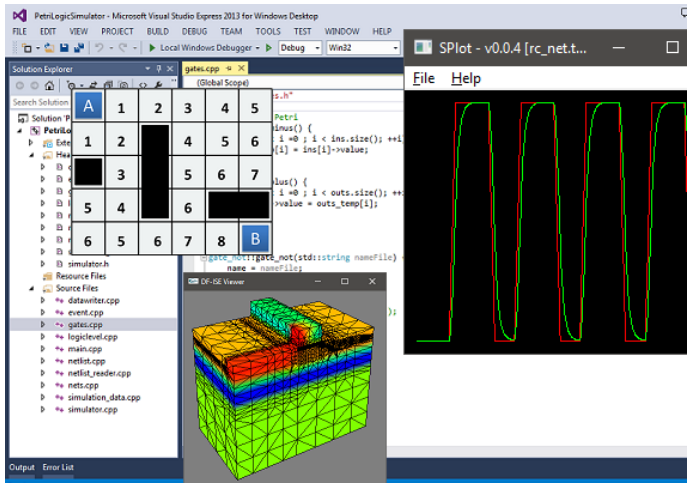
Национальный исследовательский университет «МИЭТ»

Институт интегральной электроники (группы ЭН-34-35, каф. ПКИМС)

Программные средства САПР

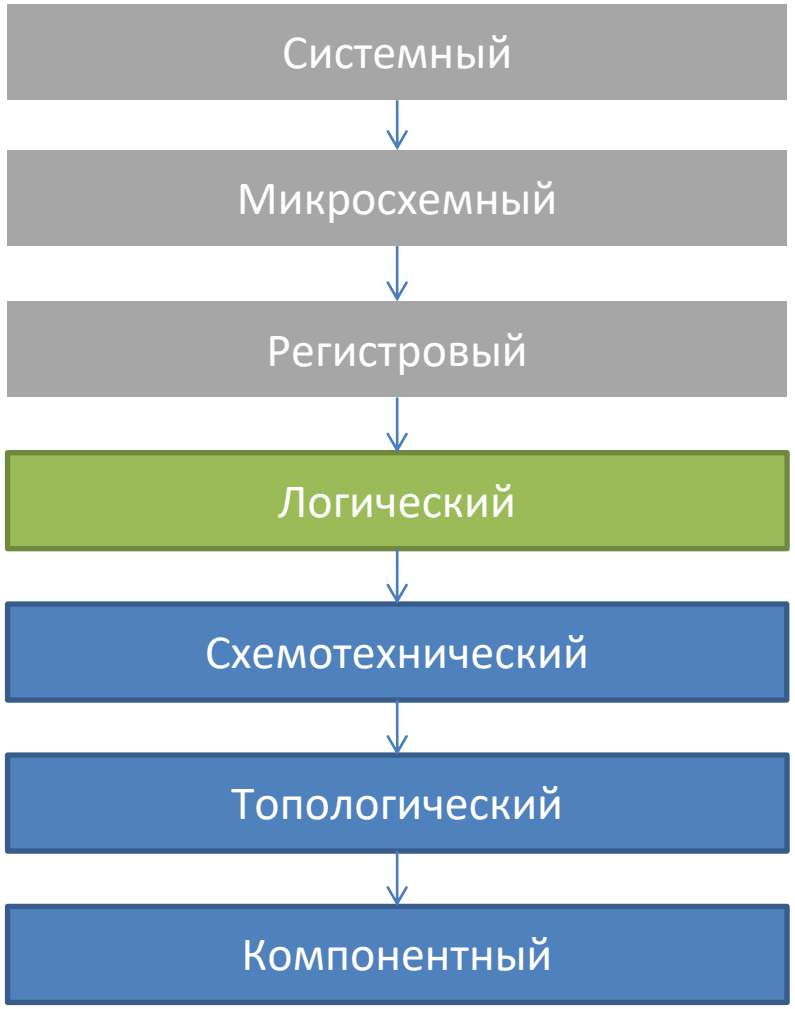


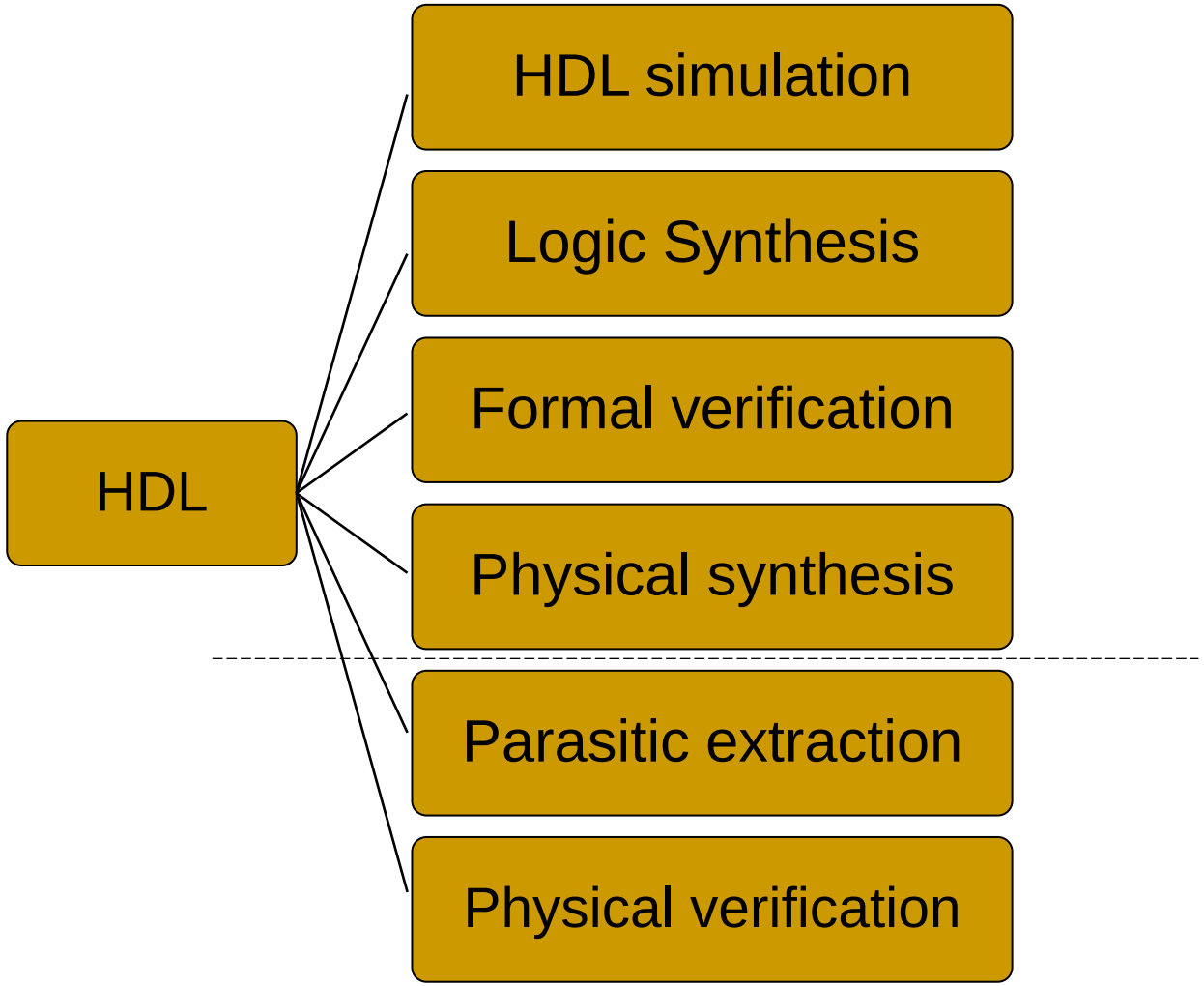
Лекция 5

Логический этап проектирования

Часть 2

Логический этап проектирования





Языки описания и моделирования цифровой аппаратуры

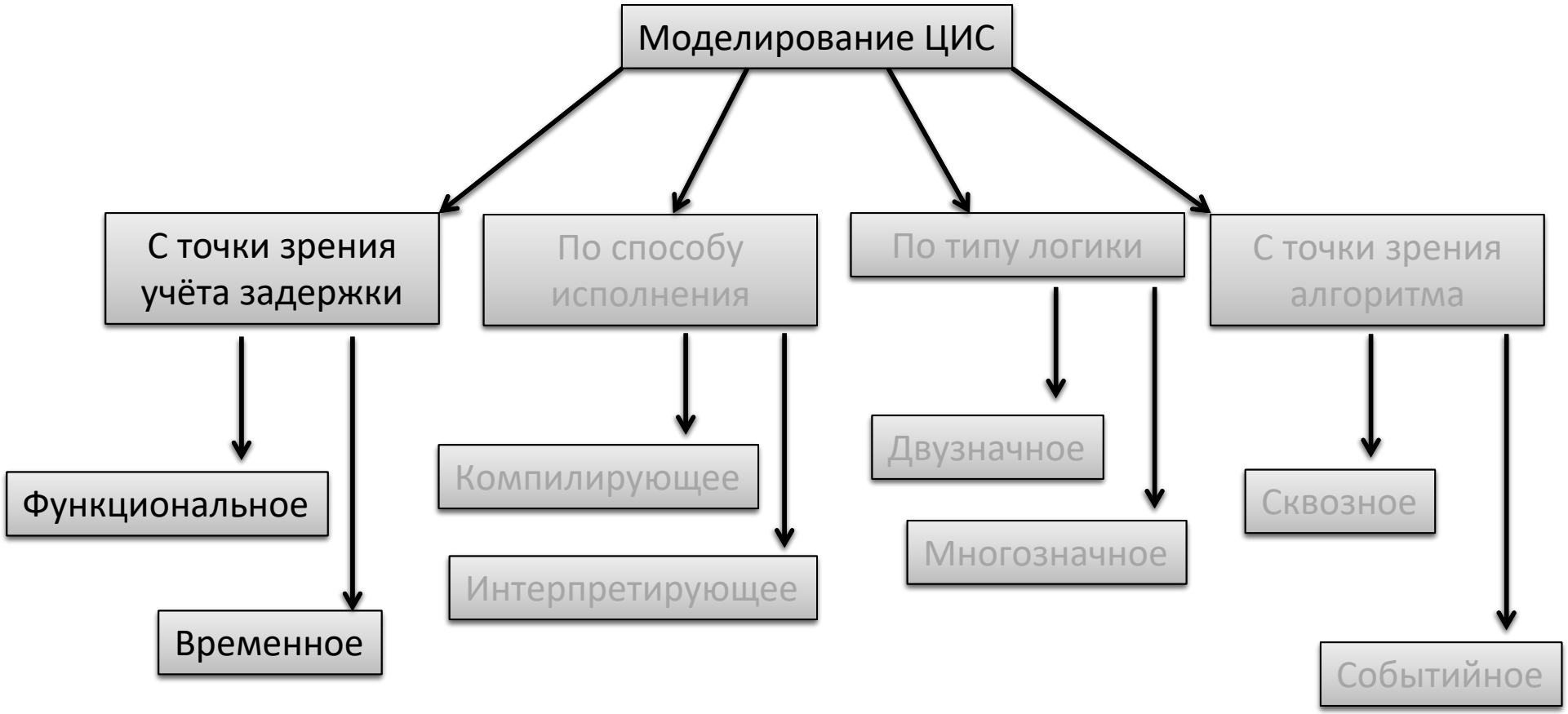
Verilog HDL

```
module device(x1, x2, y);
  input  x1, x2;
  output y;

  wire a, b;

  inverter i1(x2, a);
  and2     a1(x1, a, b);
  inverter i2(b, y);

endmodule
```

VHDL

```
architecture STR of DEVICE is
  component inv
    port(x: in STD_LOGIC; y: out STD_LOGIC);
  end component;
  component and2
    port(x1, x2: in STD_LOGIC; y: out STD_LOGIC);
  end component;

  signal a, b: bit;

begin

  p1 : inv port map(x2, a);
  p2 : and2 port map(x1, a, b);
  p3 : inv port map(b, y);

end STR;
```

Классификация алгоритмов логического моделирования

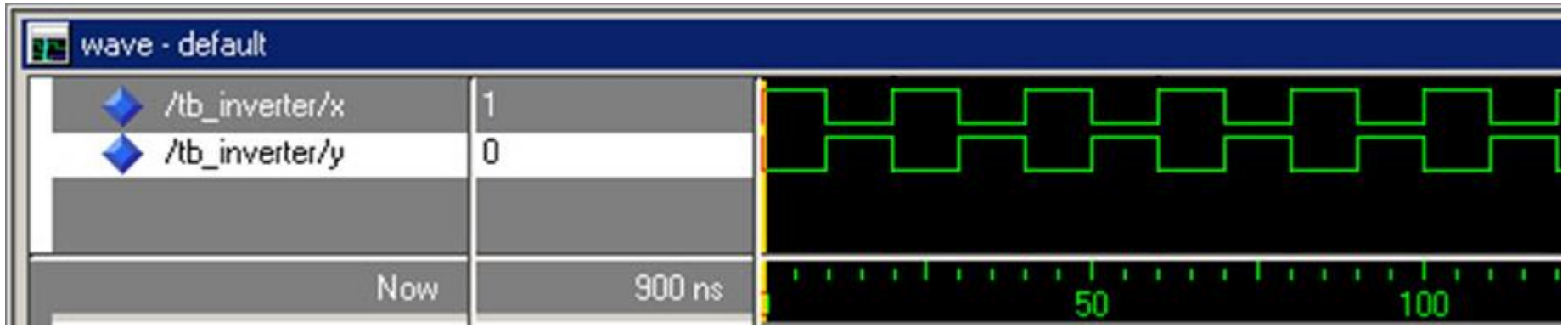


Функциональное моделирование

```
entity INV is
  port (X: in STD_LOGIC;
        Y: out STD_LOGIC;
end INV;

architecture RTL of INV is
begin
  Y <= not X;
end RTL;
```

Характеристики:

- не требует дополнительной информации, кроме информации об элементах;
- простое с точки зрения реализации;
- используется в качестве начального или промежуточного моделирования;
- не достоверно.

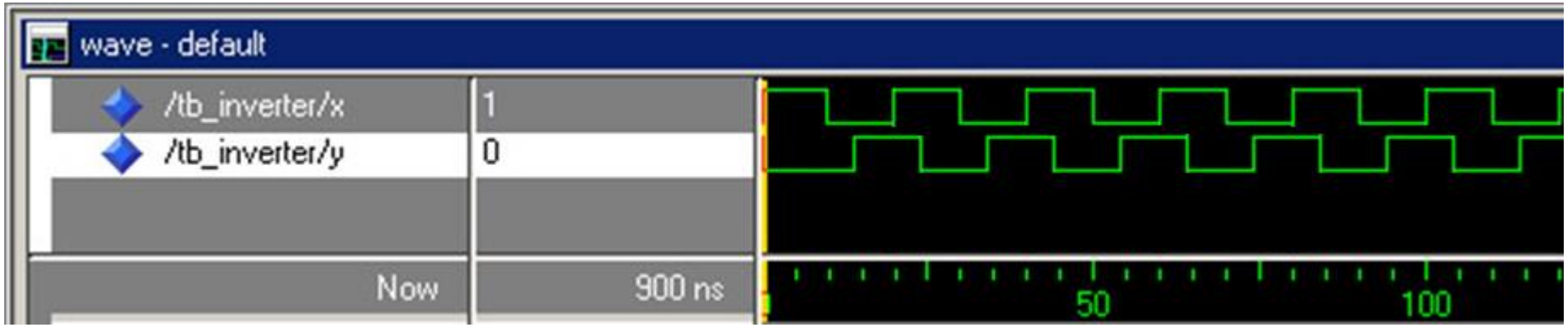


Временное моделирование

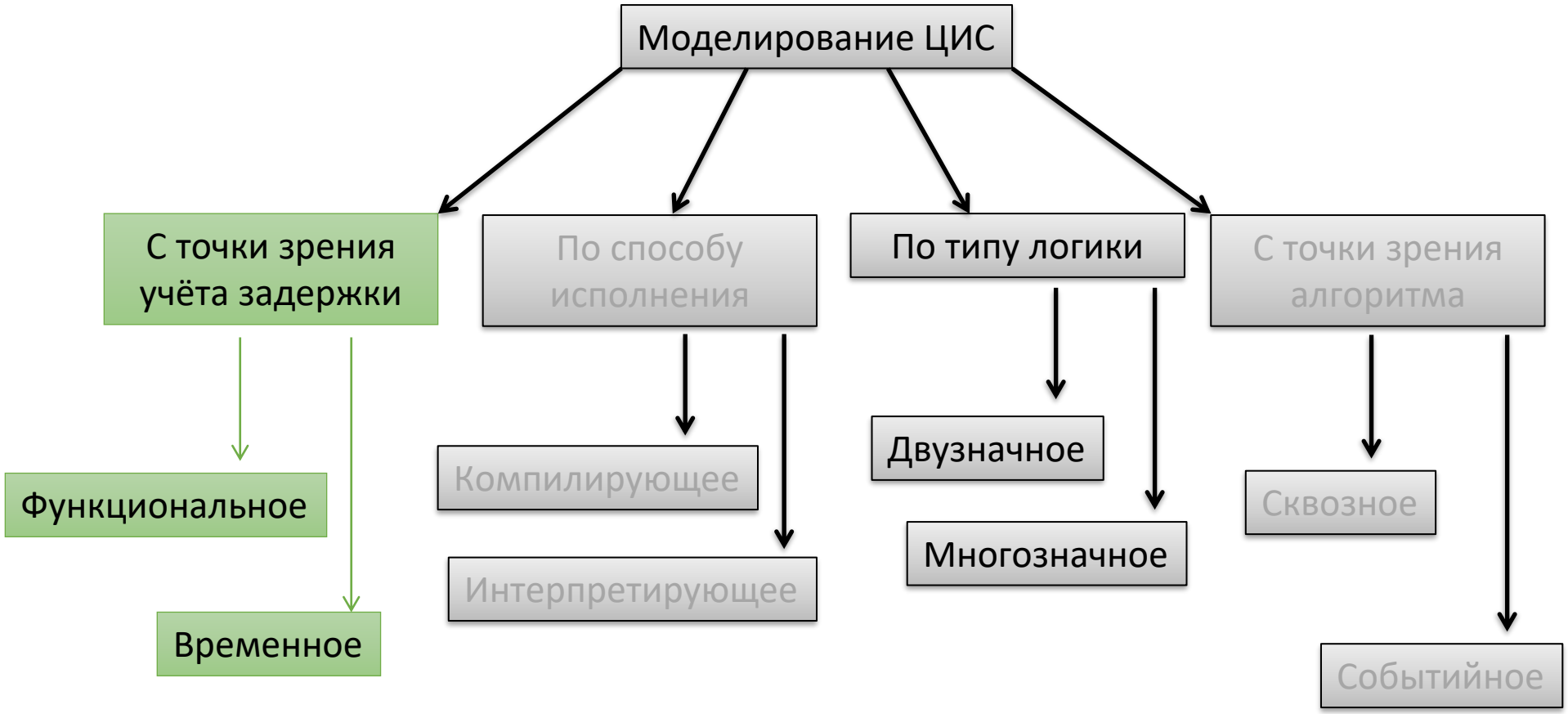
```
entity INV is
  port (X: in STD_LOGIC;
        Y: out STD_LOGIC;
end INV;

architecture RTL of INV is
begin
  Y <= not X after 3ns;
end RTL;
```

Характеристики:

- требует дополнительной информации, кроме информации об элементах;
- более сложное с точки зрения реализации;
- используется в качестве конечного моделирования;
- достоверно.



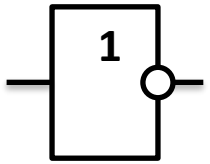
Классификация алгоритмов логического моделирования



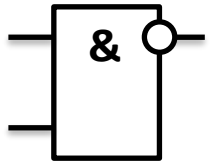
Типы логик: двузначная логика

Двузначная логика

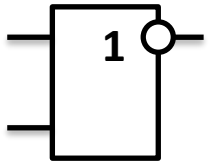
```
bit :          {'0', '1'};
```



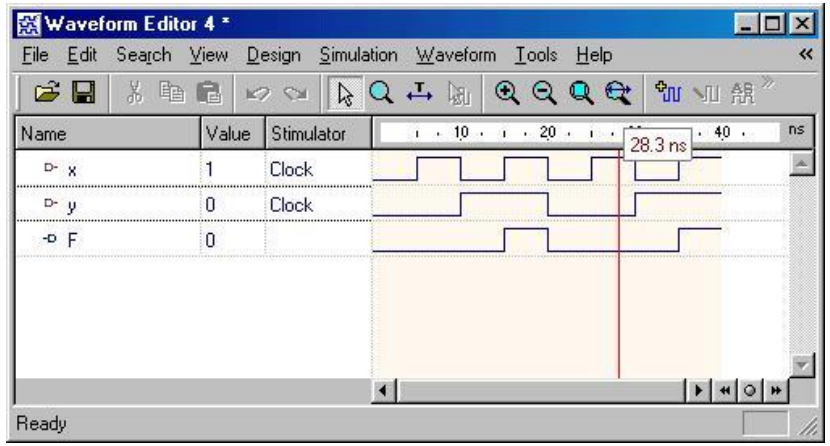
Инвертор



2И-НЕ



2ИЛИ-НЕ



NOT

x	y
0	1
1	0

NAND2

x1	x2	y
0	0	1
0	1	1
1	0	1
1	1	0

NOR2

x1	x2	y
0	0	1
0	1	0
1	0	0
1	1	0

Типы логик: многозначная логика

Многозначная логика в Verilog HDL

```
LOGIC : {
  0,   -- логический 0
  1,   -- логическая 1
  X,   -- неизвестное состояние
  Z,   -- высокий импеданс
}
```

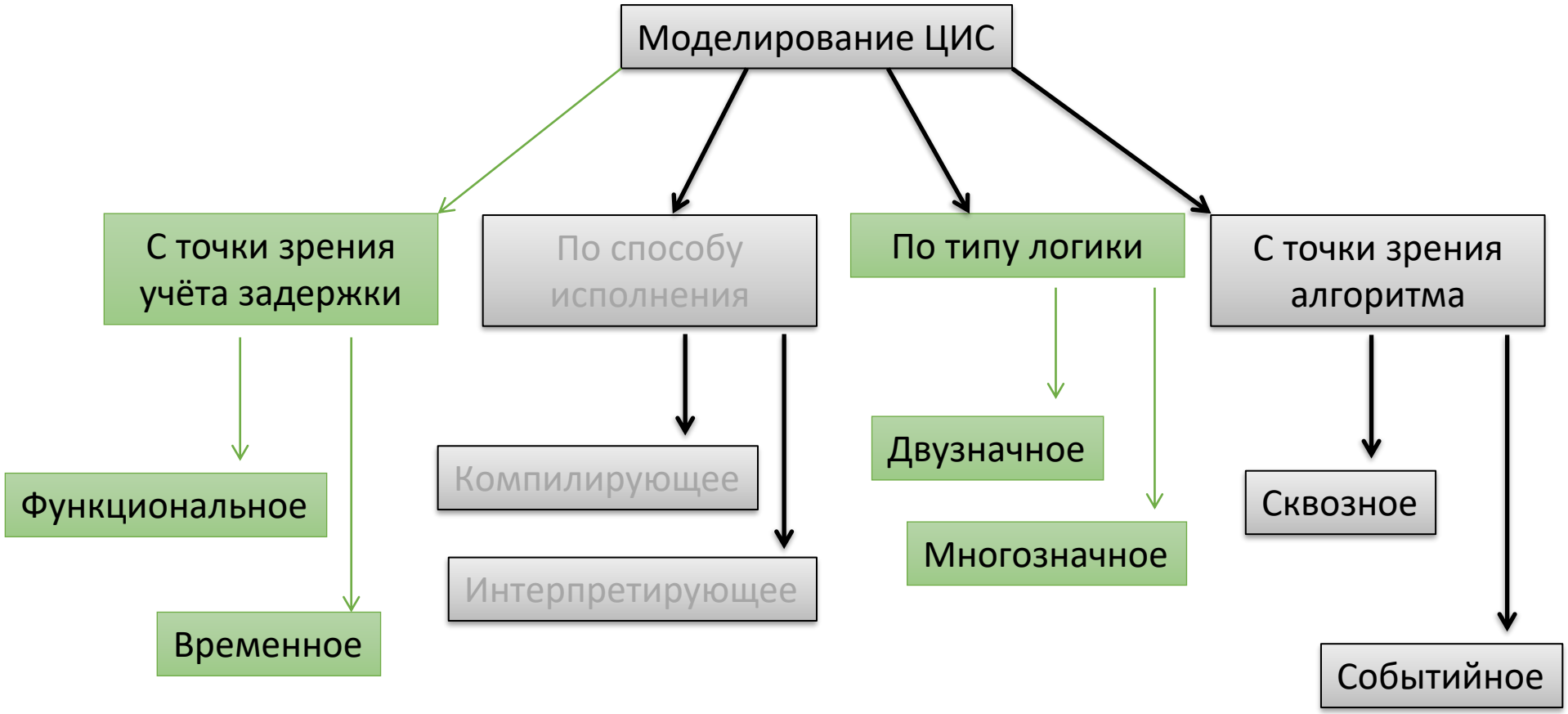
Многозначная логика в VHDL

```
STD_LOGIC : {
  'U',   -- неинициализированное значение
  'X',   -- неизвестное значение
  '0',   -- логический 0
  '1',   -- логическая 1
  'Z',   -- высокий импеданс
  'W',   -- слабый сигнал, непонятно, он 0 или 1
  'L',   -- слабый сигнал, подтянутый к 0
  'H',   -- слабый сигнал, подтянутый к 1
  '-'    -- безразличное состояние
}
```

Позволяет учитывать:

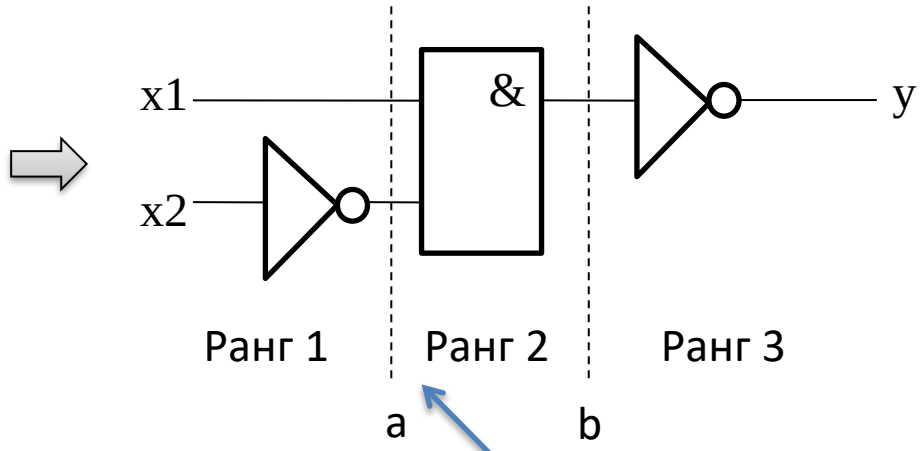
1. различные уровни сигналов;
2. реальные состояния узлов цифровых схем;

Классификация алгоритмов логического моделирования



Алгоритмы моделирования: сквозное моделирование

Генератор входных воздействий




```
1. a = not x2
2. b = x1 and a
3. y = not b
```

Базовый сквозной алгоритм

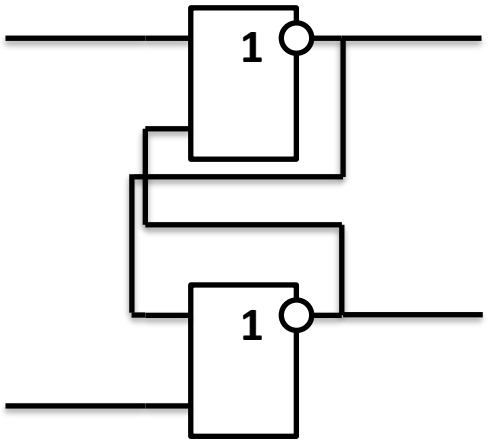
- Провести ранжирование схемы
- Сформировать ММ
- Провести моделирование (последовательно прорешать системы уравнений)

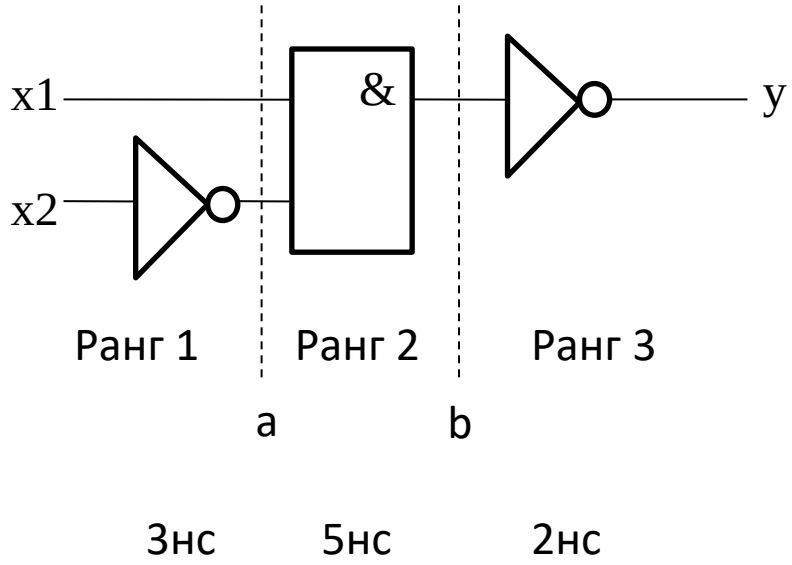
Недостатки сквозного моделирования

1. слишком объёмные вычисления;

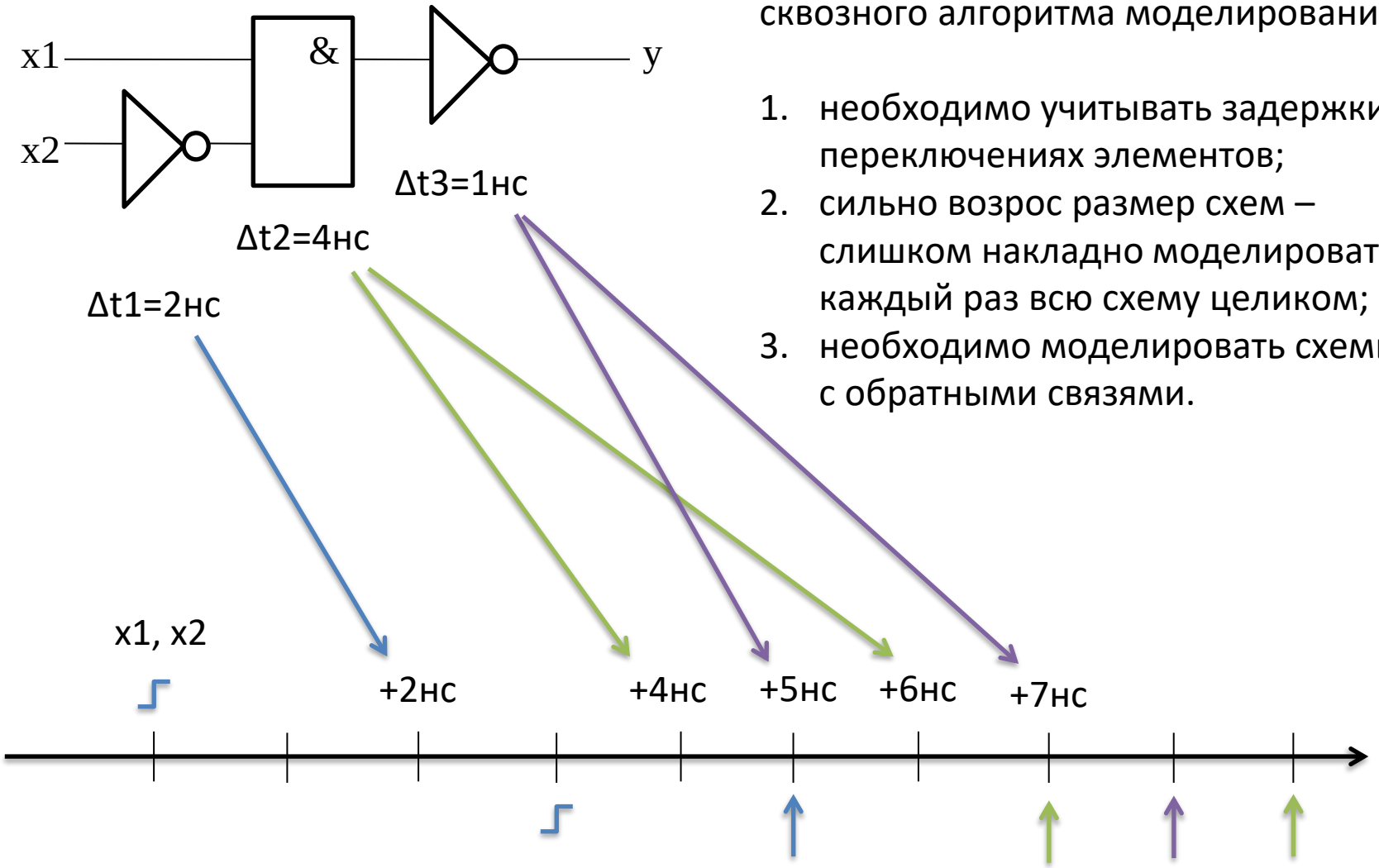
2. трудно реализуется алгоритм временного моделирования;

3. нельзя промоделировать схемы с обратными связями;





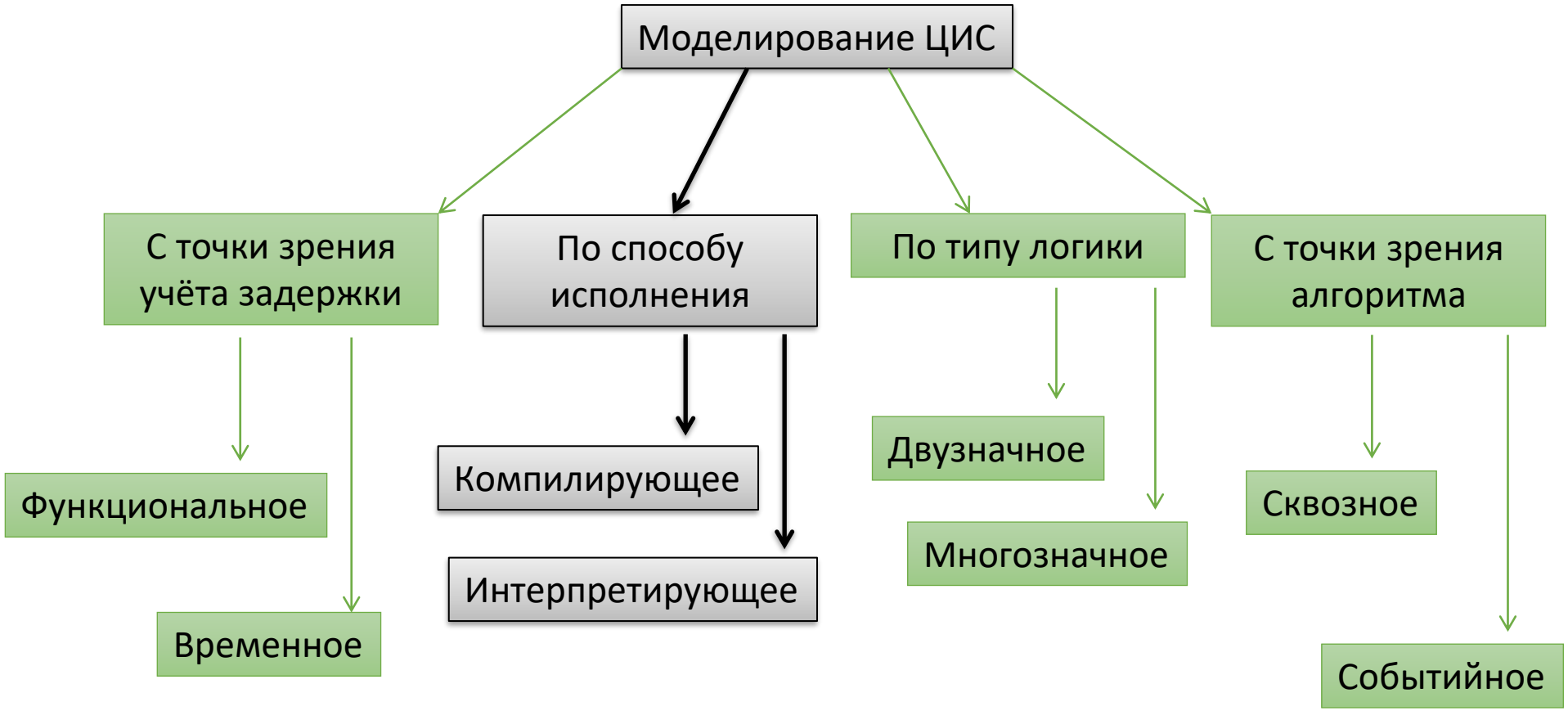
Алгоритм событийного моделирования



Основные предпосылки реализации сквозного алгоритма моделирования :

1. необходимо учитывать задержки в переключениях элементов;
2. сильно возрос размер схем – слишком накладно моделировать каждый раз всю схему целиком;
3. необходимо моделировать схемы с обратными связями.

Классификация алгоритмов логического моделирования



Компиляционное моделирование: симулятор Icarus Verilog (1)

Icarus Verilog

```
module tb_inv() ;
  reg x;
  wire y;

  inv i1 (x, y);

  initial
  begin
    $dumpfile("out.vcd");
    $dumpvars(0);

    #0   x = 1;
    #100 $finish;
  end

  always #10 x = ~x;
endmodule
```

```
`timescale 1ns/1ps

module inv(x, y);
  input  x;
  output y;

  assign y = ~x;
endmodule
```

Компиляционное моделирование: симулятор Icarus Verilog (3)

temp.dat

```
:vpi_module "C:\iverilog\lib\ivl\va_math.vpi";
S_0000010d6ce3d3b0 .scope module, "tb_inv" "tb_inv" 2 11;
 .timescale -9 -12;
v0000010d6ce3ee30_0 .var "x", 0 0;
v0000010d6ce3eed0_0 .net "y", 0 0, L_0000010d6d043270;  1 drivers
S_0000010d6d0759a0 .scope module, "il" "inv" 2 15, 2 3 0, S_0000010d6ce3d3b0;
 .timescale -9 -12;
    .port_info 0 /INPUT 1 "x";
    .port_info 1 /OUTPUT 1 "y";
L_0000010d6d043270 .functor NOT 1, v0000010d6ce3ee30_0, C4<0>, C4<0>, C4<0>;
v0000010d6ce3d540_0 .net "x", 0 0, v0000010d6ce3ee30_0;  1 drivers
v0000010d6d042ee0_0 .net "y", 0 0, L_0000010d6d043270;  alias, 1 drivers
    .scope S_0000010d6ce3d3b0;
T_0 ;
    %vpi_call 2 19 "$dumpfile", "out.vcd" {0 0 0};
    %vpi_call 2 20 "$dumpvars", 32'sb00000000000000000000000000000000 {0 0 0};
    %delay 0, 0;
    %pushi/vec4 1, 0, 1;
    %store/vec4 v0000010d6ce3ee30_0, 0, 1;
    %delay 100000, 0;
    %vpi_call 2 22 "$finish" {0 0 0};
    %end;
    .thread T_0;
    .scope S_0000010d6ce3d3b0;
T_1 ;
    %delay 10000, 0;
    %load/vec4 v0000010d6ce3ee30_0;
    %inv;
    %store/vec4 v0000010d6ce3ee30_0, 0, 1;
    %jmp T_1;
    .thread T_1;
# The file index is used to find the file name in the following table.
:file_names 3;
    "N/A";
    "<interactive>";
    "test.v";
```

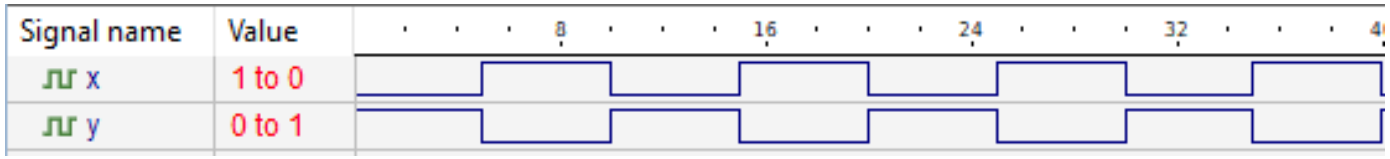
Пример компилирующего моделирования: библиотека SystemC

```
SC_MODULE(inverter) {
  sc_in   <bool>  x;
  sc_out  <bool>  y;

  void doOperate() {
    if(true == x)
      y = false;
    else
      y = true;
  }

  SC_CTOR(inverter) {
    SC_METHOD(doOperate);
    sensitive << x;
  }
};
```

```
struct inverter : public sc_module {
...
```

```
int main(int argc, char *args[]) {

    sc_clock clock("clk", 5);
    sc_signal <bool> y;

    display     dpl("echo_module");
    dpl.x(clock);
    dpl.y(y);

    inverter    inv("my_inverter");
    inv.x(clock);
    inv.y(y);

    sc_start(100, SC_NS);

    return 0;
}
```

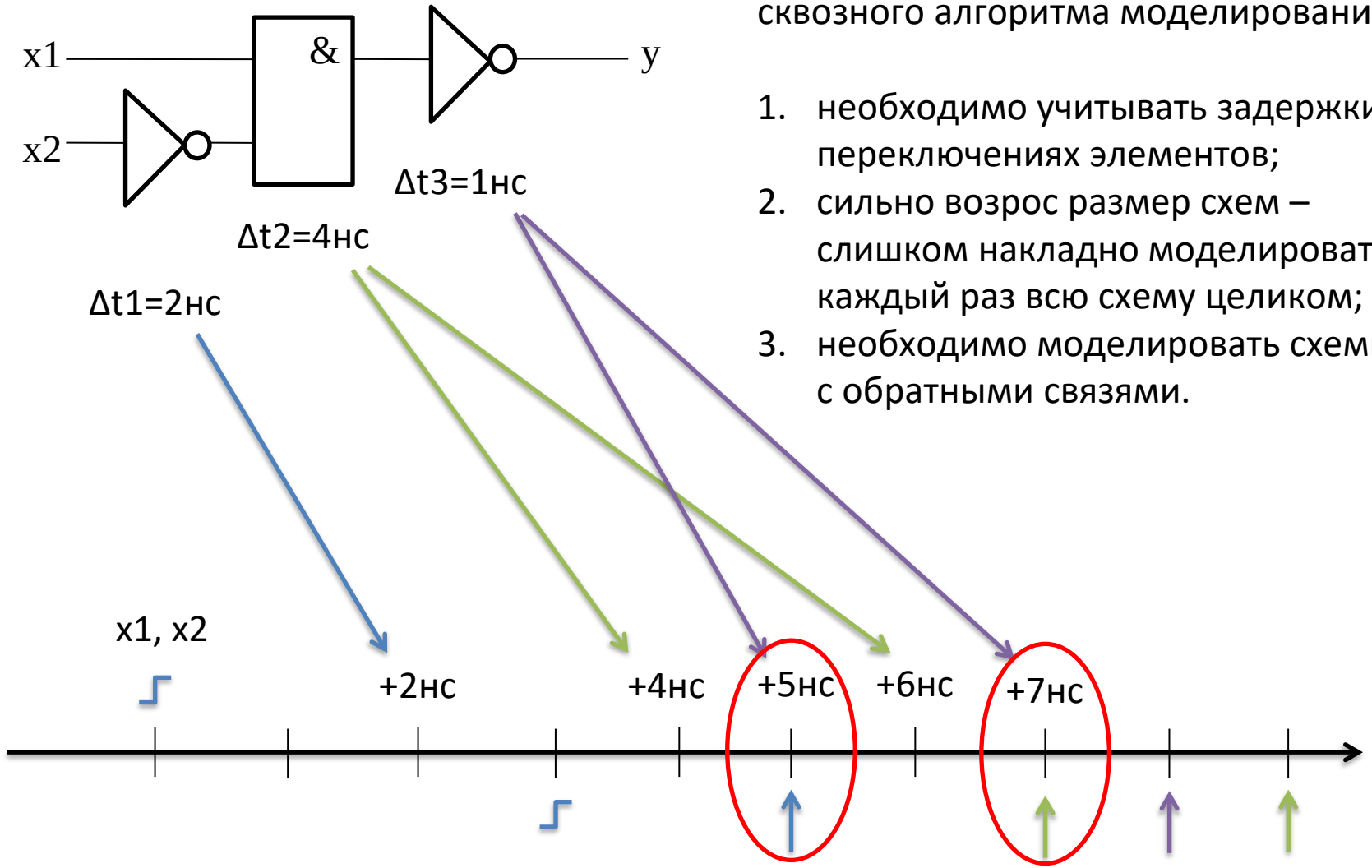
Представление результатов моделирования: формат VCD




```
$date
        Thu Apr 05 11:51:54 2018
$end
$version
        Icarus Verilog
$end
$timescale
        1ns
$end
$scope module inverter_tb $end
$var reg 1 ! x $end
$var wire 1 " y $end
$upscope $end
$enddefinitions $end
```

```
#0
$dumpvars
0!
1"
$end
#5
1!
0"
#10
0!
1"
#20
1!
0"
#30
0!
1"
#35
1!
0"
```

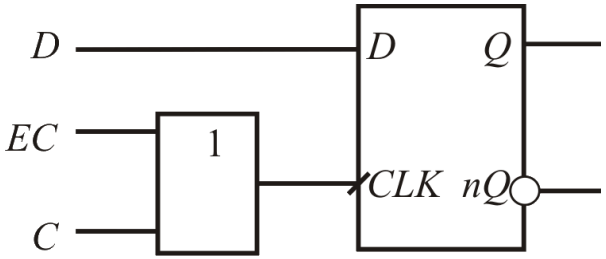
Проблемы событийного моделирования

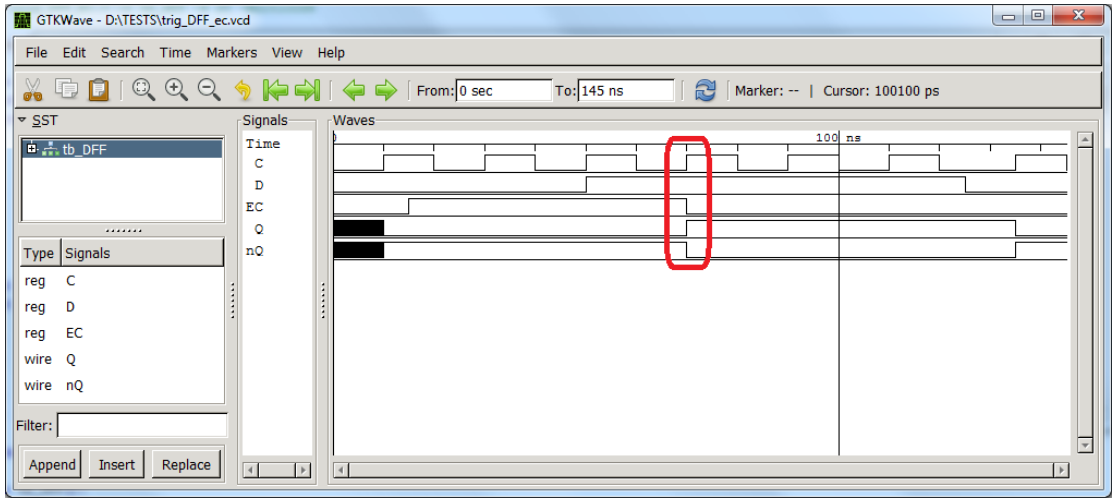


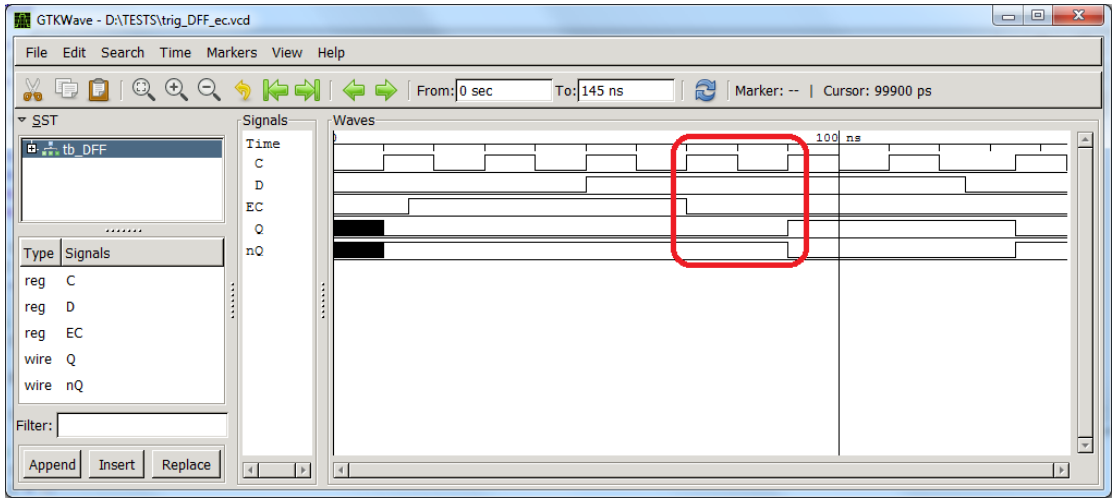
Основные предпосылки реализации сквозного алгоритма моделирования :

1. необходимо учитывать задержки в переключениях элементов;
2. сильно возрос размер схем – слишком накладно моделировать каждый раз всю схему целиком;
3. необходимо моделировать схемы с обратными связями.

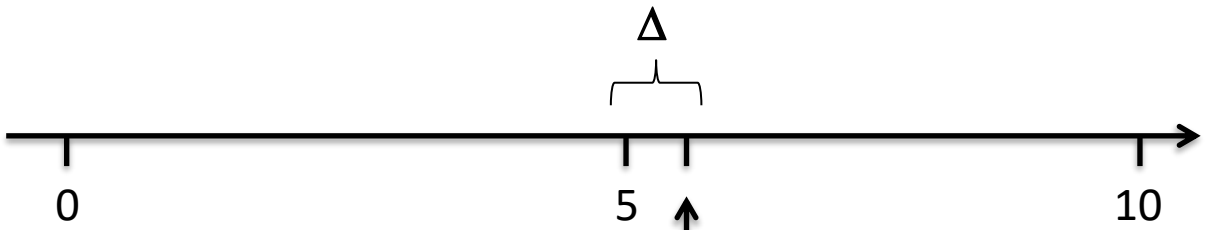
Проблемы событийного моделирования



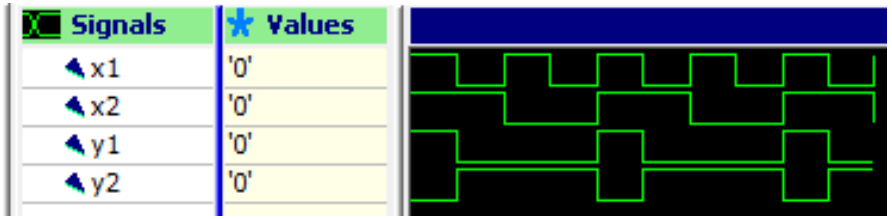


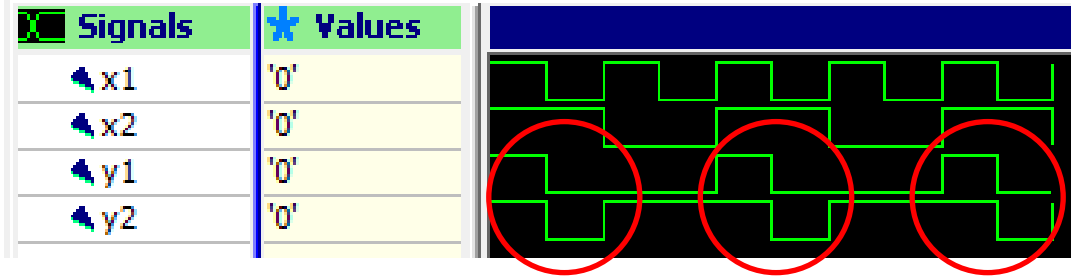


Δ-задержка

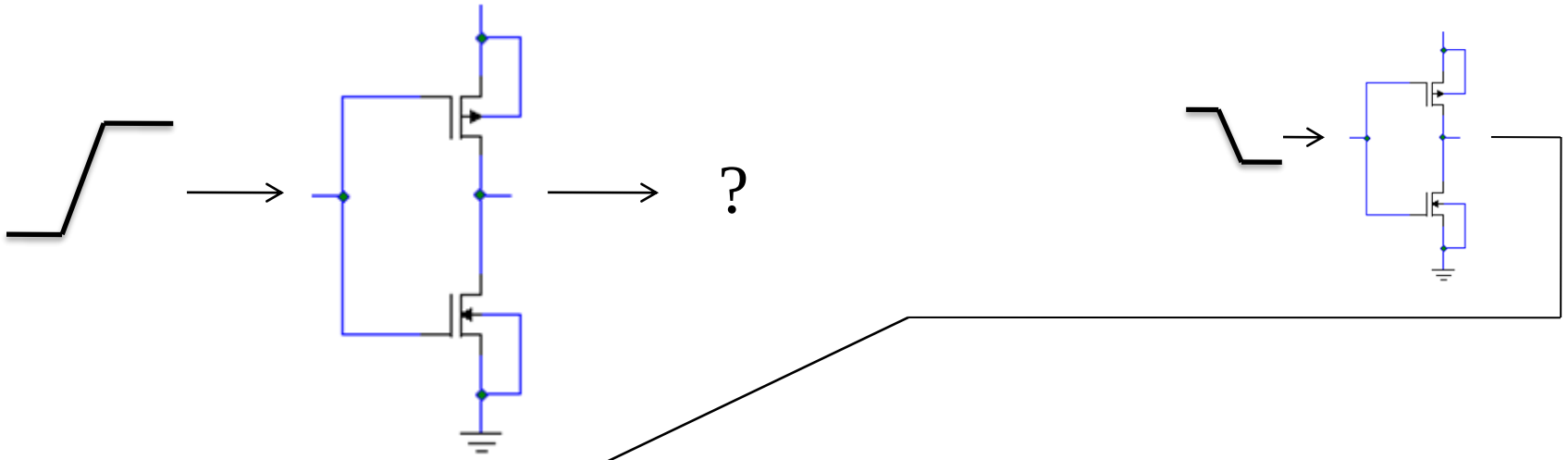


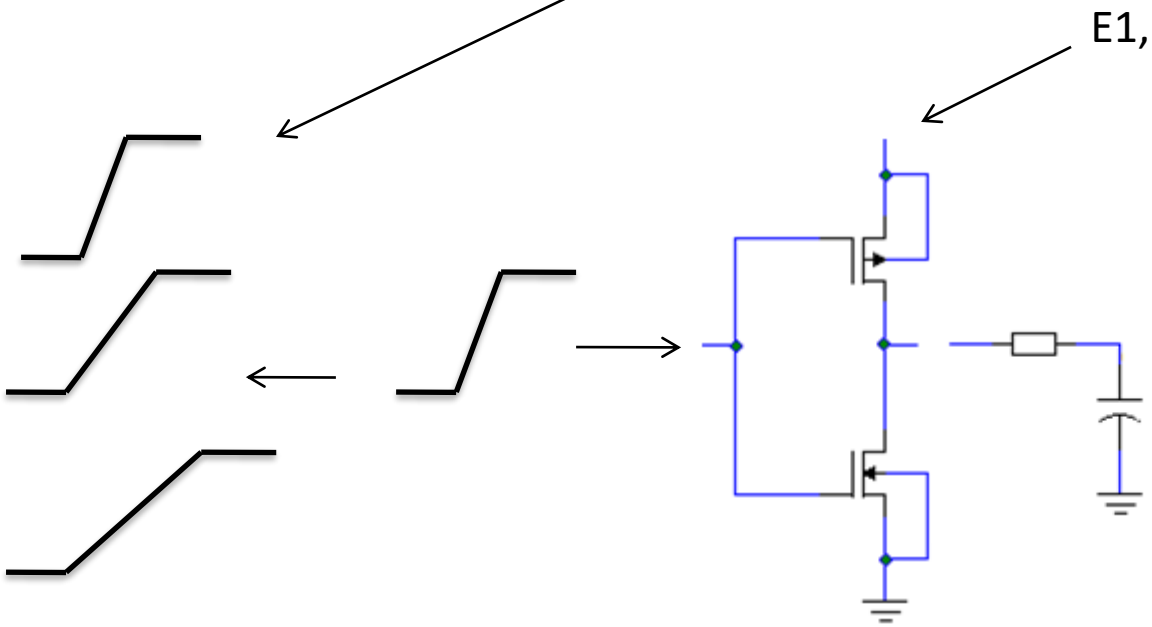

```
process(x1, x2)
begin
  if(x1 = '1' and x2 = '1') then
    y1 <= '1';
  else
    y1 <= '0';
  end if;
  y2 <= not y1;
end process;
```





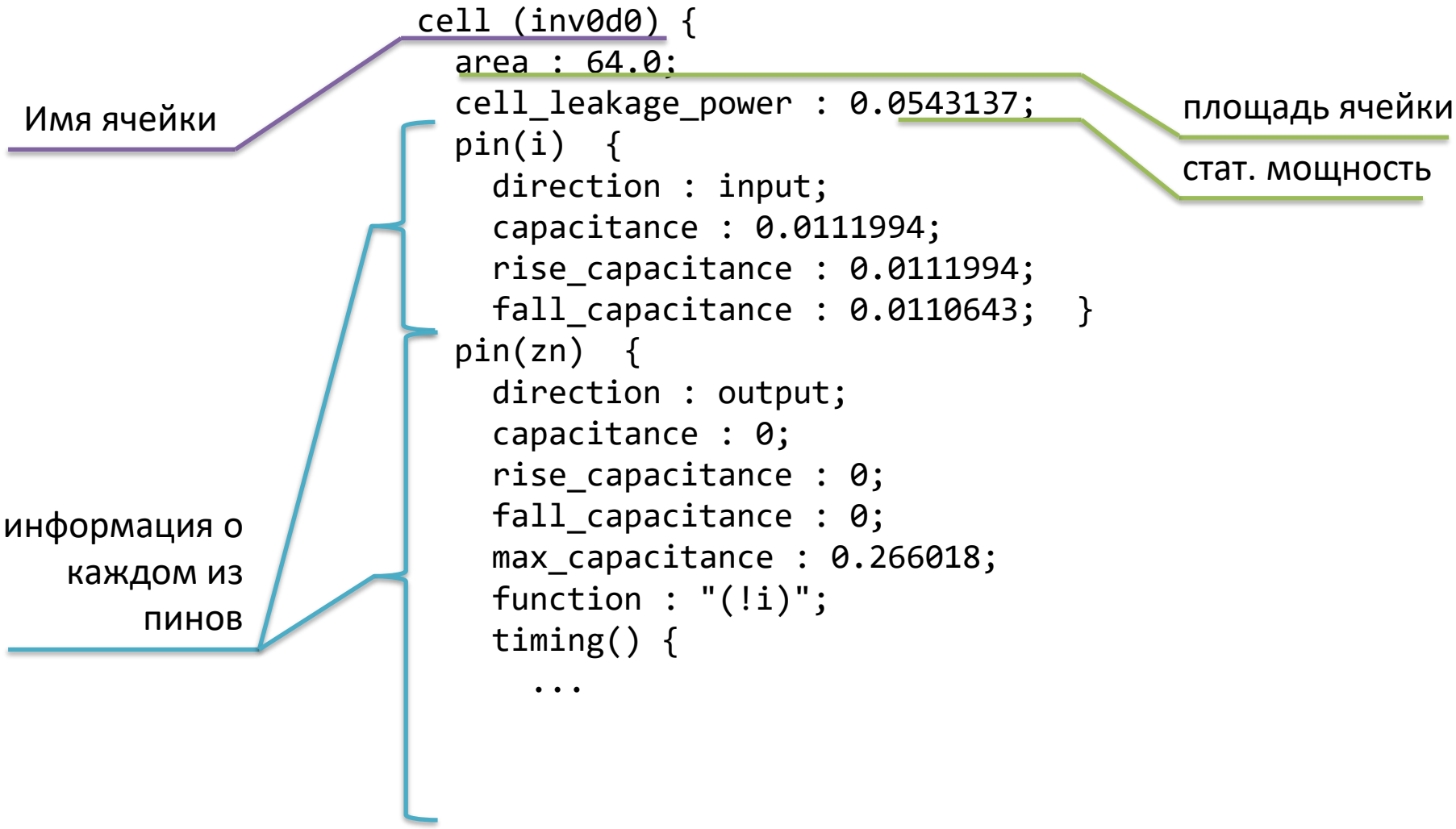
Характеризация библиотечных элементов





R\C	C1	C2	C3
R1	t1	t2	t3
R2	t4	t5	t6
R3	t7	t8	t9

Результат характеризации: формат Synopsys Liberty (1)




```
cell (inv0d0) {
  area : 64.0;
  cell_leakage_power : 0.0543137;
  pin(i)  {
    direction : input;
    capacitance : 0.0111994;
    rise_capacitance : 0.0111994;
    fall_capacitance : 0.0110643;  }
  pin(zn)  {
    direction : output;
    capacitance : 0;
    rise_capacitance : 0;
    fall_capacitance : 0;
    max_capacitance : 0.266018;
    function : "(!i)";
    timing() {
      ...
```

Результат характеризации: формат Synopsys Liberty (2)

От какого пина смотрим зависимость

«Форма» реакции

Время нарастания фронта

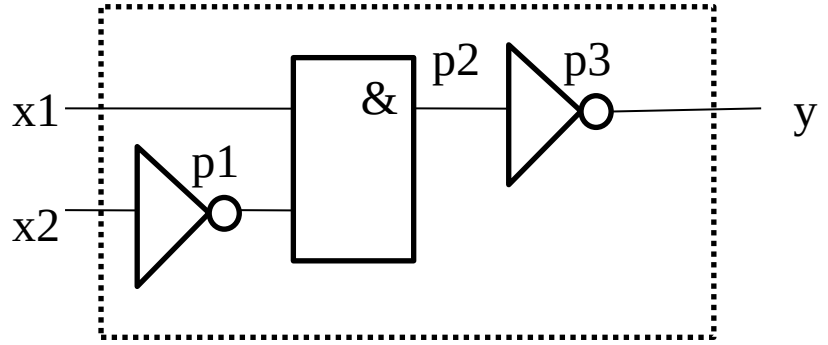
Выходная нагрузка

значения задержки

```
timing() {
     related_pin : "i";
     timing_sense : negative_unate;
     cell_rise(delay_template_6x6) {
       index_1 ("0.016, 0.032, 0.064, 0.128, 0.256, 0.512");
       index_2 ("0.08, 0.16, 0.32, 0.48, 0.64, 0.8");
       values ( \
         "0.055434, 0.069821, 0.082992, 0.096737, 0.108544, 0.114477", \
         "0.089193, 0.096673, 0.131445, 0.153615, 0.170875, 0.184564", \
         "0.162878, 0.178567, 0.197923, 0.243968, 0.275384, 0.301804", \
         "0.358053, 0.362199, 0.386151, 0.415653, 0.429929, 0.490506", \
         "0.811713, 0.819202, 0.832069, 0.867784, 0.894318, 0.91146", \
         "1.81723, 1.82331, 1.83447, 1.86957, 1.89758, 1.90694");
     }
```

Формат хранения информации о задержках в схеме: SDF (Standard Delay Format) (1)

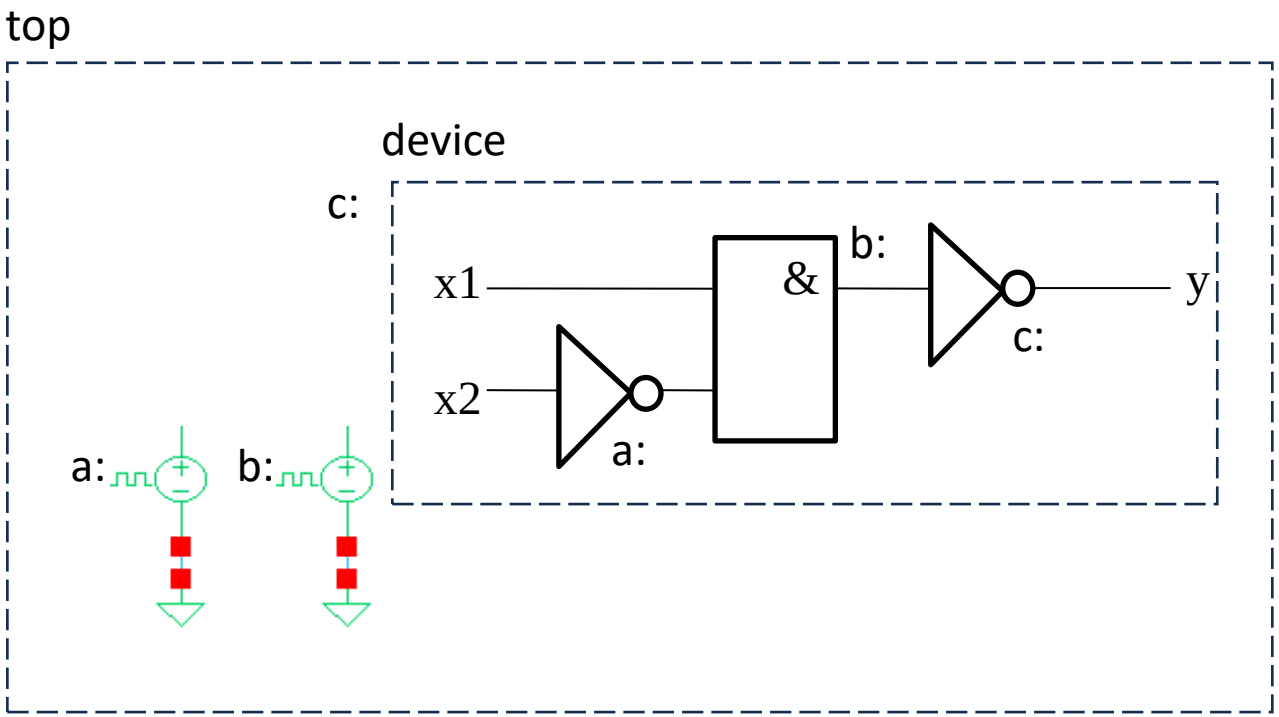
```
entity device is
  port(x1, x2: in STD_LOGIC; y: out STD_LOGIC);
end device;

architecture STR of device is
  component inv
    port(x: in STD_LOGIC; y: out STD_LOGIC);
  end component;
  component and2
    port(x1, x2: in STD_LOGIC; y: out STD_LOGIC);
  end component;

  signal a, b: bit;

begin
  a : inv port map(x2, a);
  b : and2 port map(x1, a, b);
  c : inv port map(b, y);
end STR;
```




```
entity top is
end top;

architecture TEST of top is
  component device
    port(x1, x2: in STD_LOGIC; y: out STD_LOGIC);
  end component;

  signal x1, x2: STD_LOGIC := '0';
  signal y: STD_LOGIC;
begin
  a : process
       begin
         x1 <= not x1;
         wait for 5 ns;
       end process;
       ...
  c : device port map(x1, x2, y);

end TEST;
```

Формат хранения информации о задержках в схеме: SDF (Standard Delay Format) (2)

```
(DELAYFILE
  (SDFVERSION "3.0")
  (DESIGN "TEST")
  (DATE "March 12, 2022 09:46")
  (TIMESCALE 100 ps)
  (CELL
    (CELLTYPE "AND2")
    (INSTANCE top/c/b)
    (DELAY
      (ABSOLUTE
        (IOPATH x1 y (1.5:2.5:3.4) (2.5:3.6:4.7))
        (IOPATH x2 y (1.4:2.3:3.2) (2.3:3.4:4.3))
      )
    )
  )
  (CELL
    (CELLTYPE "INV")
    (INSTANCE top/c/a)
    (DELAY
      (ABSOLUTE
        (IOPATH x y (2:3:4) (5:6:7))
      )
    )
  )
```

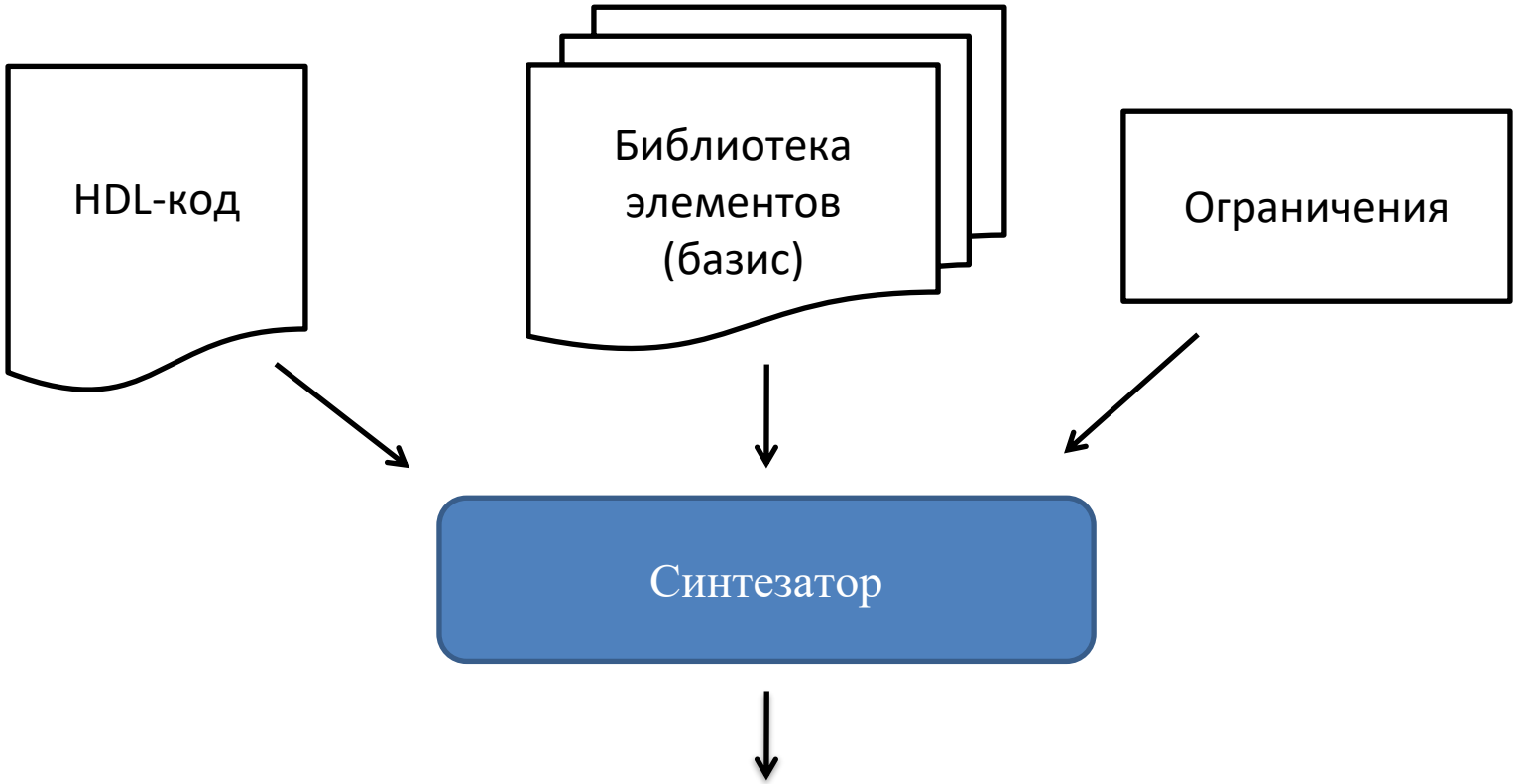


Синтез цифровых схем: цифровой синтез

```
module device(x1, x2, y);
  input  x1, x2;
  output y;

  always @(x1 or x2)
    if (x1 == 1 && x2 == 0)
      y <= 0;
    else
      y <= 1;

endmodule
```

```
module device(x1, x2, y);
  input  x1, x2;
  output y;

  wire a, b;

  not i1(x2, a);
  and a1(x1, a, b);
  not i2(b, y);

endmodule
```

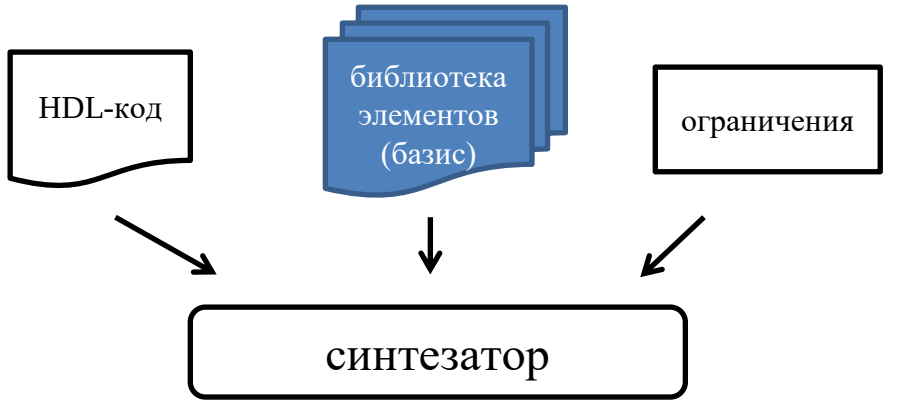
Работа программы синтеза

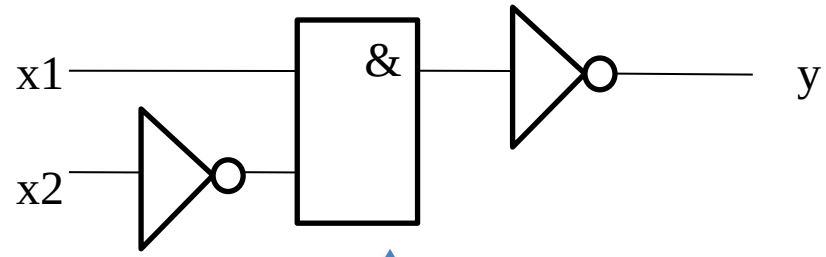


1. RTL HDL-описание;
2. нетлист на аналоговом языке – spice, spectre;
3. топологическое представление схемы.

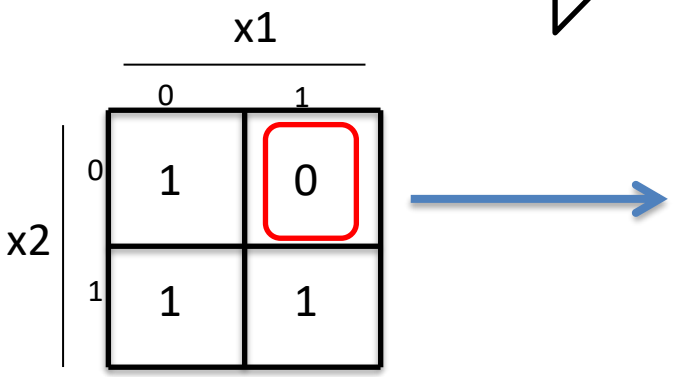
Синтез комбинационных схем методом Карт Карно (1)

```
module device(x1, x2, y);
  input  x1, x2;
  output y;

  wire a, b;

  not i1(x2, a);
  and a1(x1, a, b);
  not i2(b, y);

endmodule
```





x1	x2	y
0	0	1
0	1	1
1	0	0
1	1	1

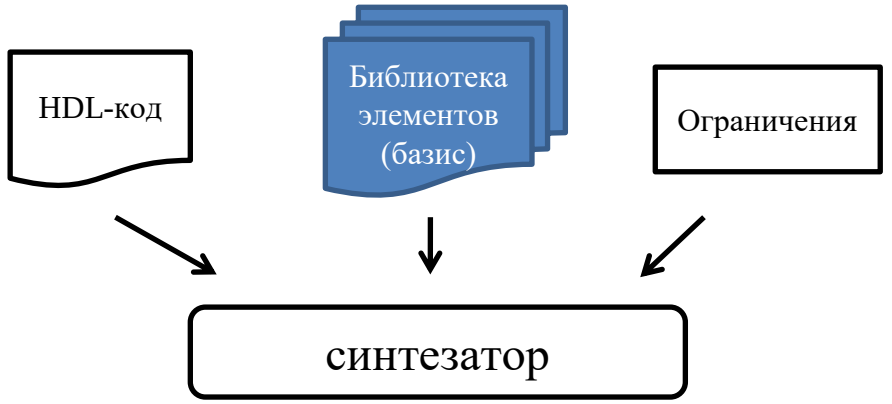


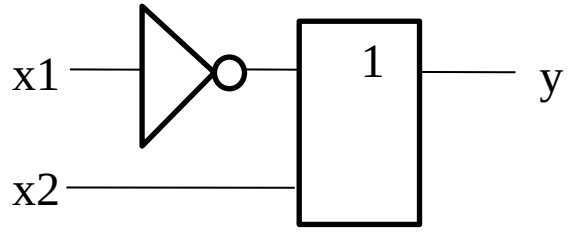
$x1 * \overline{x2}$

Базис – «И» - смотрим по нулям, выход инвертируем

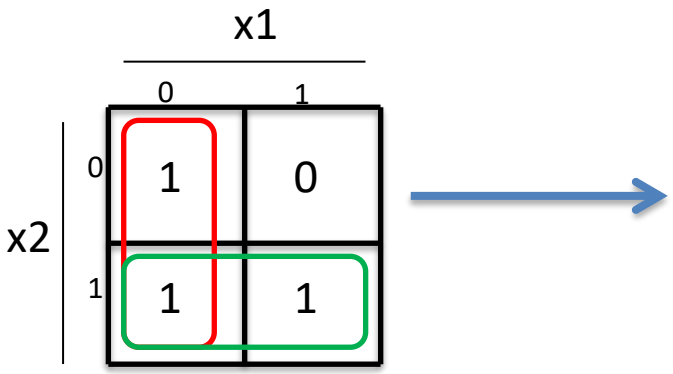
Синтез комбинационных схем методом Карт Карно (2)

```
module device(x1, x2, y);
  input  x1, x2;
  output y;

  wire a, b;

  not i1(x2, a);
  and a1(x1, a, b);
  not i2(b, y);

endmodule
```





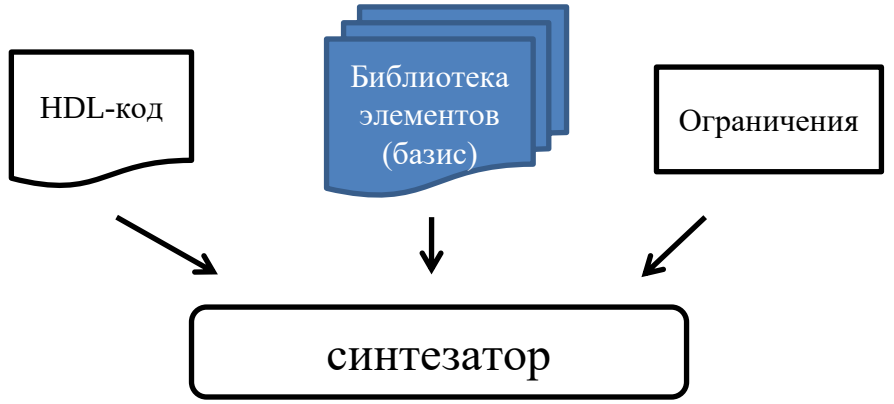
x1	x2	y
0	0	1
0	1	1
1	0	0
1	1	1

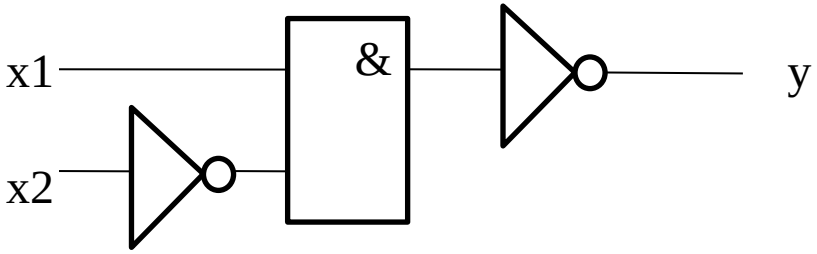


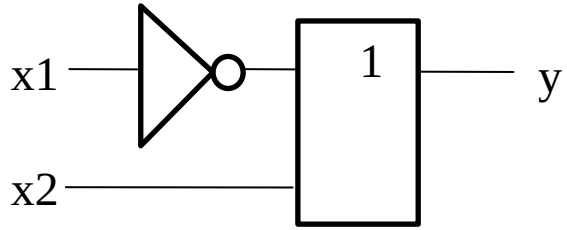
$\overline{x1} + x2$

Базис – «ИЛИ» - смотрим по единицам

Важность наполнения библиотеки для синтеза (1)







Число транзисторов: 10

А если бы у нас был NAND2: 6

Число транзисторов: 8

Способы представления функций комбинационных логических схем

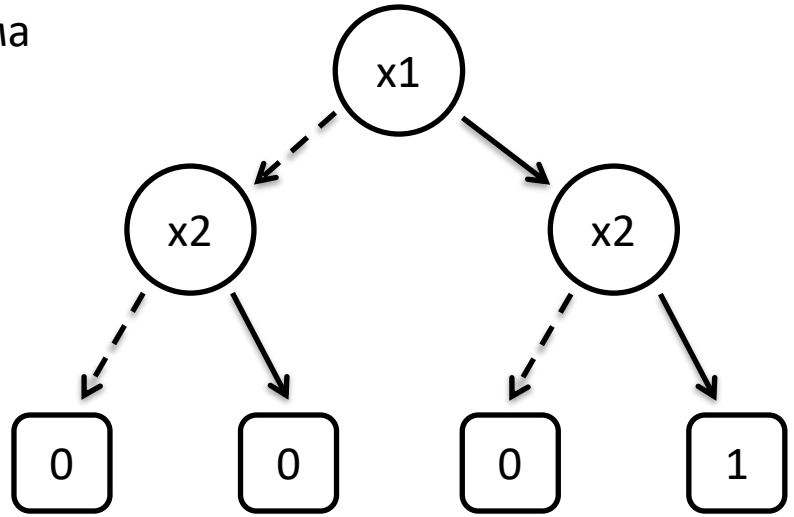
Таблица истинности

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

Карта Карно

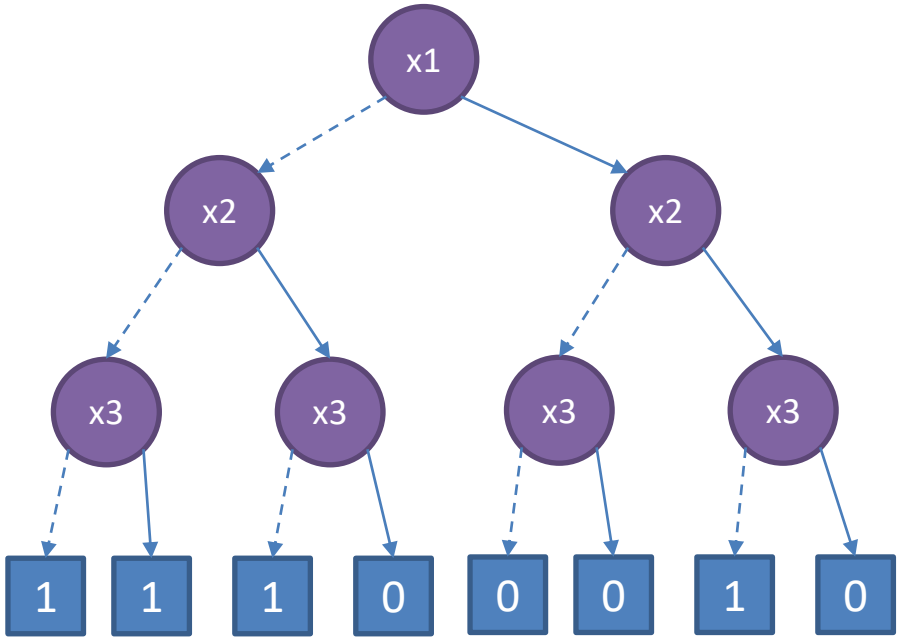
x1 \ x2	0	1
0	0	0
1	0	1

Диаграмма двоичных решений



Оптимизация логической схемы на основе BDD (1)

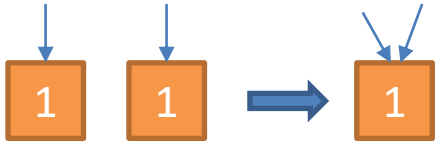
x1	x2	x3	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

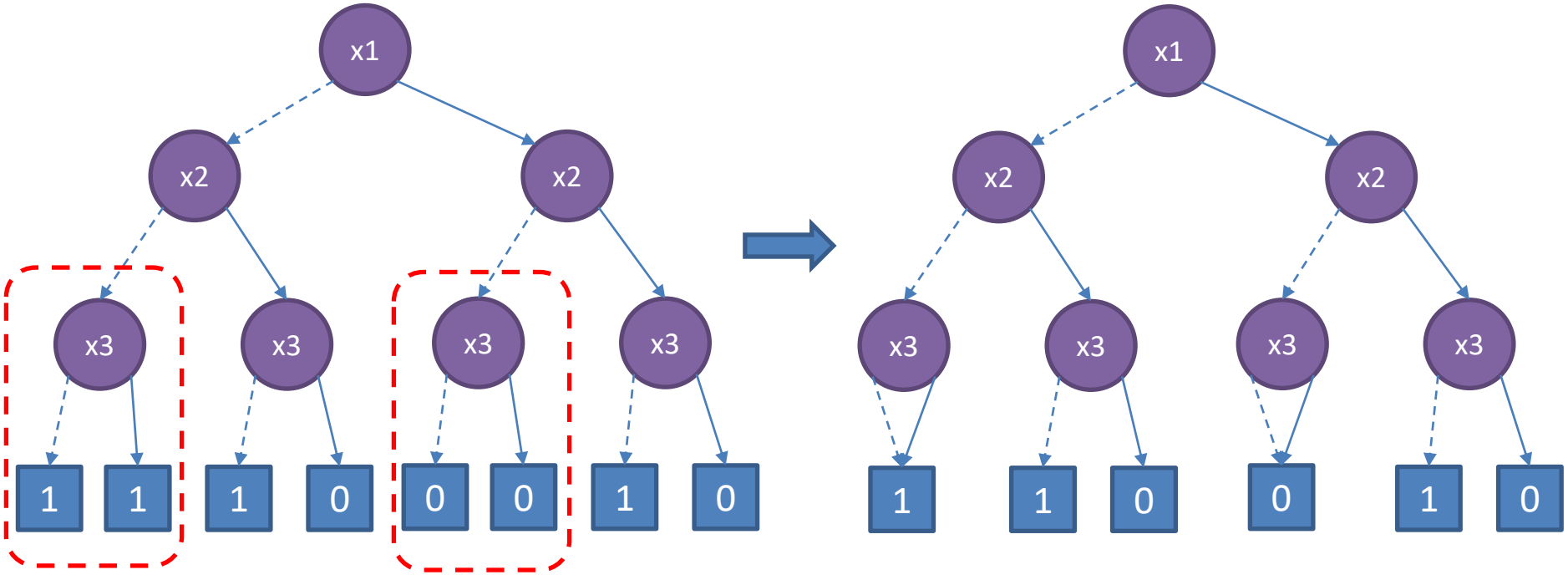


f(x1, x2, x3) = !x1*!x2*!x3 + !x1*!x2*x3 + !x1*x2*!x3 + x1*x2*!x3

Оптимизация логической схемы на основе BDD (2)

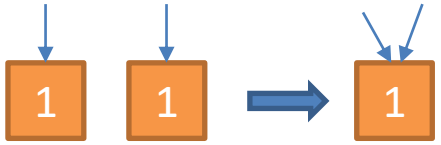
Правило 1. Эквивалентные узлы сливаются

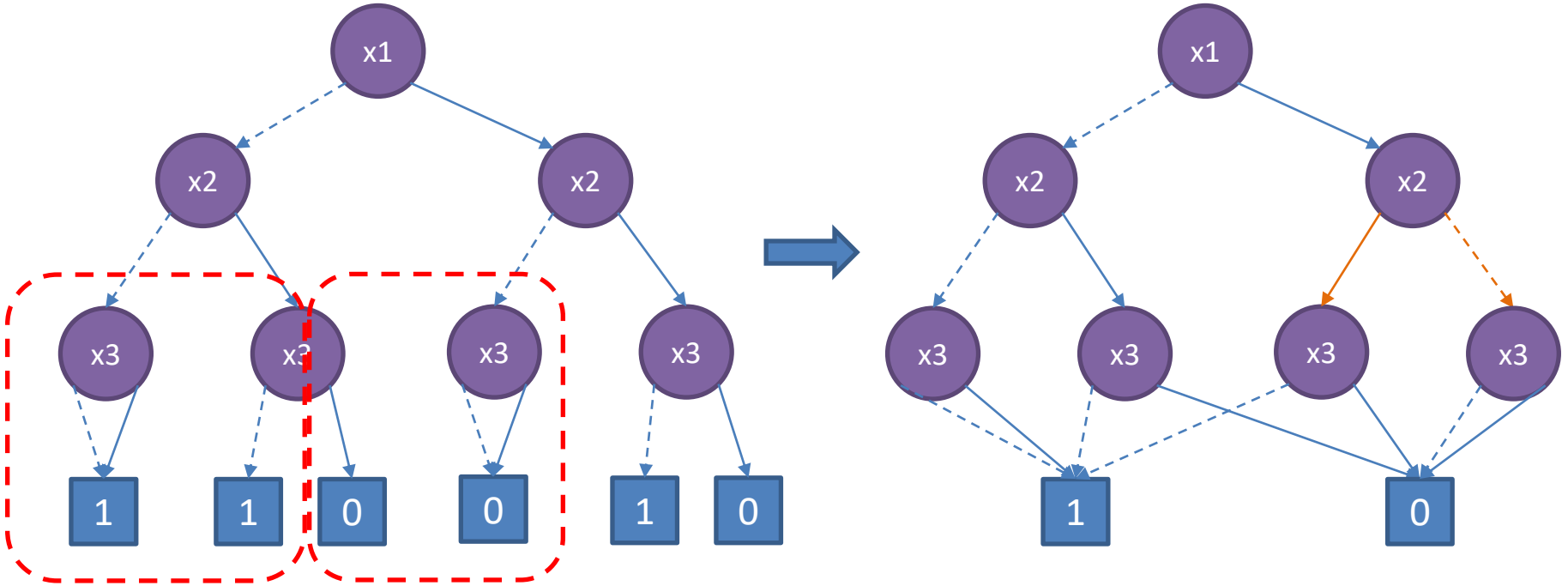




Оптимизация логической схемы на основе BDD (3)

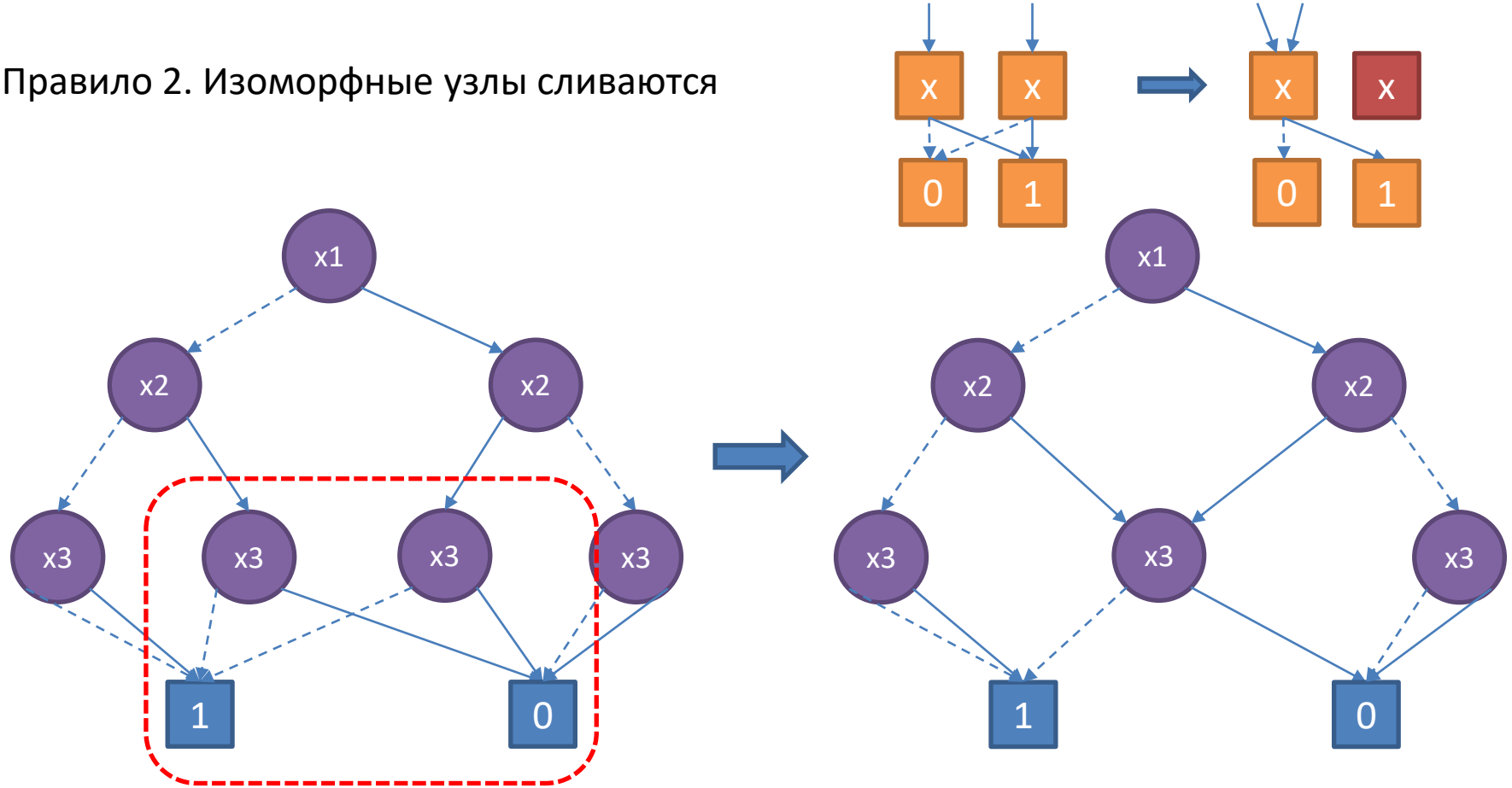
Правило 1. Эквивалентные узлы сливаются





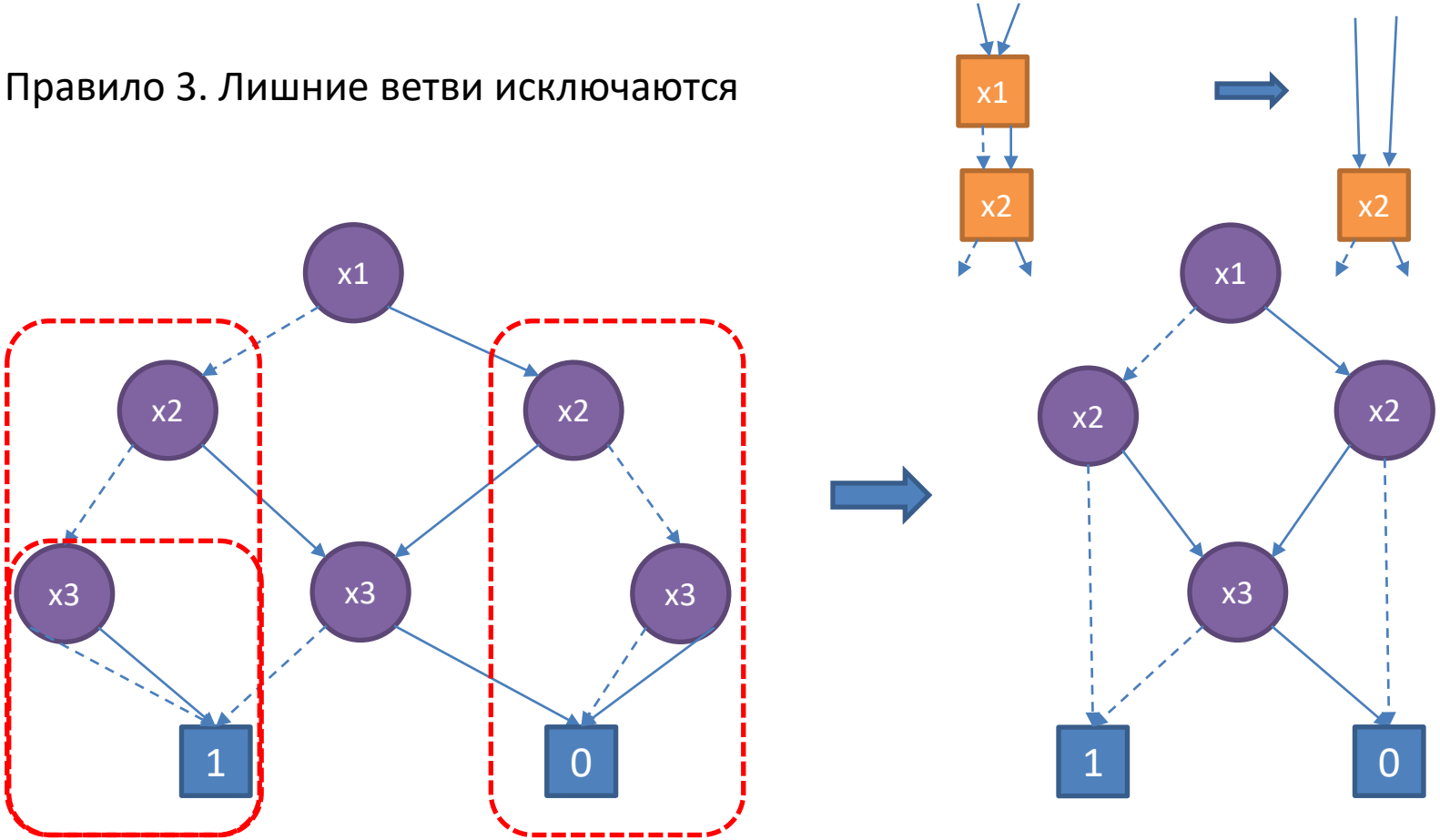
Оптимизация логической схемы на основе BDD (4)

Правило 2. Изоморфные узлы сливаются



Оптимизация логической схемы на основе BDD (5)

Правило 3. Лишние ветви исключаются



f(x1, x2, x3) = !x1*!x2 + !x1*x2*!x3 + x1*x2*!x3

f(x1, x2, x3) = !x1*!x2*!x3 + !x1*!x2*x3 + !x1*x2*!x3 + x1*x2*!x3

Библиотека элементов: файл формата Synopsys Liberty

```
library(mylib) {
  cell(BUF) {
    area: 1;
    pin(A) { direction: input; }
    pin(Y) { direction: output;
             function: "A"; }
  }
  cell(NOT) {
    area: 1;
    pin(A) { direction: input; }
    pin(Y) { direction: output;
             function: "A'"; }
  }
  cell(AND) {
    area: 1;
    pin(A) { direction: input; }
    pin(B) { direction: input; }
    pin(Y) { direction: output;
             function: "(A&B)"; }
  }
```

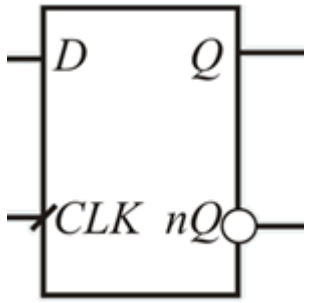
```
cell(NOR) {
  area: 1;
  pin(A) { direction: input; }
  pin(B) { direction: input; }
  pin(Y) { direction: output;
           function: "(A+B)'"; }
}
```

```
cell(DFF) {
    area: 10;
    ff(IQ, IQN) { clocked_on: C;
                  next_state: D; }
    pin(C) { direction: input;
                 clock: true; }
    pin(D) { direction: input; }
    pin(Q) { direction: output;
             function: "IQ"; }
  }
```

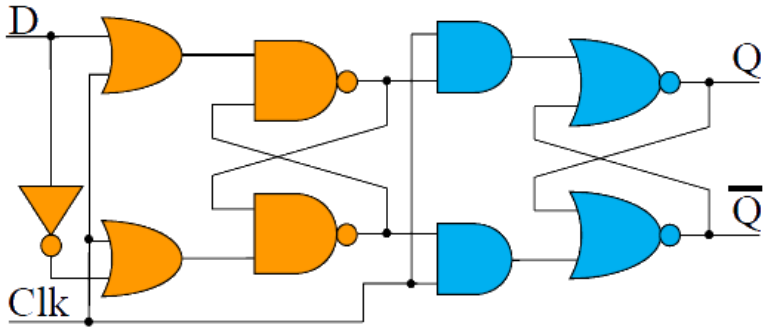
Фрагмент библиотеки NanGate

```
cell (AND2_X1) {
        drive_strength      : 1;
        area                : 1.064000;
        pg_pin(VDD) {
                voltage_name        : VDD;
                pg_type             : primary_power;
        }
        pg_pin(VSS) {
                voltage_name        : VSS;
                pg_type             : primary_ground;
        }
        pin (A1) {
                direction           : input;
                related_power_pin   : "VDD";
                related_ground_pin  : "VSS";
        }
        pin (A2) {
                direction           : input;
                related_power_pin   : "VDD";
                related_ground_pin  : "VSS";
        }
        pin (ZN) {
                direction           : output;
                related_power_pin   : "VDD";
                related_ground_pin  : "VSS";
                function            : "(A1 & A2)";
        }
 }
```

Перевод описания в вентильный уровень

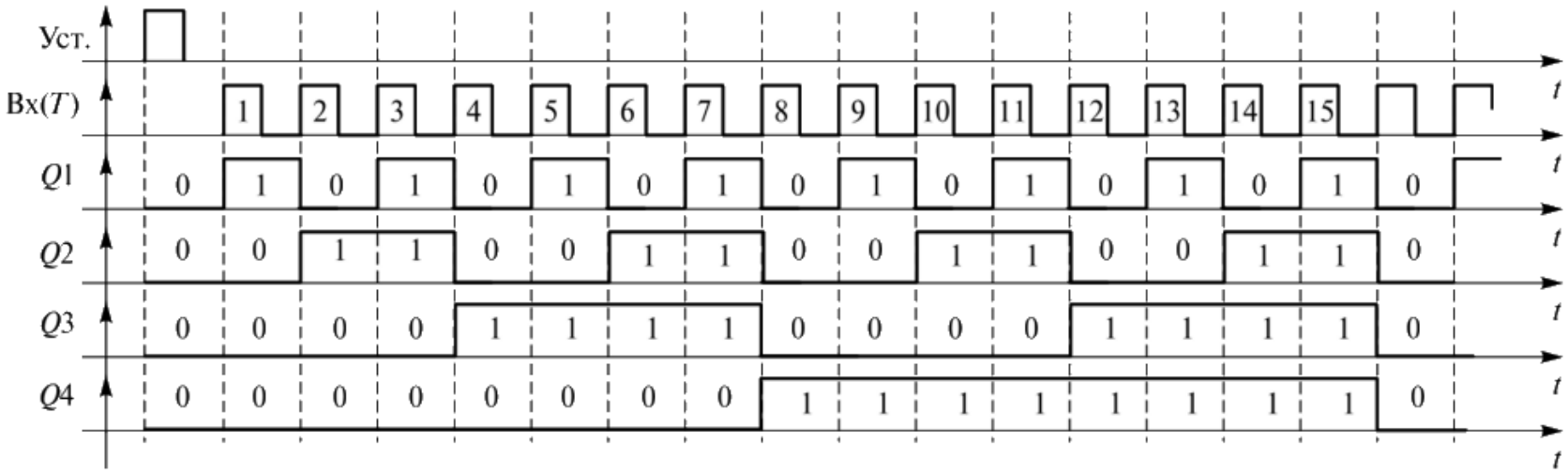



```
module dff_BEH(D, C, Q, nQ);
  input D, C;
  output Q, nQ;
  reg Q, nQ;

  always @(posedge C)
  begin
    Q = D;
    nQ = ~D;
  end
endmodule
```




```
module dff_RTL(D, C, Q, nQ);
  input  D, C;
  output Q, nQ;
  wire   nD, a1, a2, b1, b2, c1, c2;

  not   i1(nD, D);
  or    i2(a1, D, C);
  or    i3(a2, nD, C);
  nand  i4(b1, a1, b2);
  nand  i5(b2, a2, b1);
  and   i6(c1, b1, C);
  and   i7(c2, b2, C);
  nor   i8(Q,  c1, nQ);
  nor   i9(nQ, c2, Q);
endmodule
```

Синтез схемы счётчика в САПР Synopsys (1)




```
module Johnson_count(clk, r, out);
  input clk;
  input r;
  output reg [0:3]out;
  always @ (negedge clk or negedge r)
    if (r == 0)
      out = 4'b1111;
    else
      out = out+1'b1;
endmodule
```

Синтез схемы счётчика в САПР Synopsys (2)

```
module Johnson_count ( clk, r, out );
  output [0:3] out;
  input clk, r;

  wire  N2, N3, N4, n1, n3, n4, n5;

  DFFARX1_RVT out_reg3 (n5, clk, n1, out[3], n5);
  DFFARX1_RVT out_reg2 (N2, clk, n1, out[2]);
  DFFARX1_RVT out_reg1 (N3, clk, n1, out[1]);
  DFFARX1_RVT out_reg0 (N4, clk, n1, out[0]);
  INVX0_RVT   U3       (r, n1);
  NAND2X0_RVT U8       (out[1], n4, n3);
  AND2X1_RVT  U9       (out[2], out[3], n4);
  XNOR2X1_RVT U10      (n3, out[0], N4);
  XOR2X1_RVT  U11      (out[1], n4, N3);
  XNOR2X1_RVT U12      (n5, out[2], N2);
endmodule
```

Верификация и отладка цифровых схем (6)

- Функциональная (Functional Verification)
- Формальная (Formal Verification)
- Статический анализ кода (Static Code Analysis)
- **Физическая (Physical Verification)**