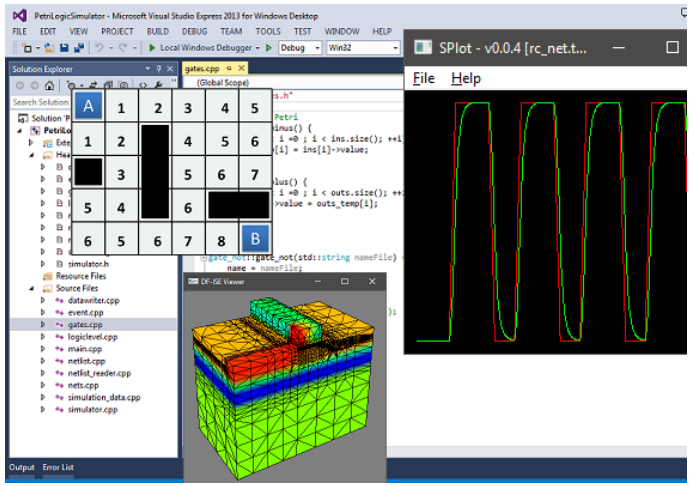
Национальный исследовательский университет «МИЭТ»

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

Программные средства САПР

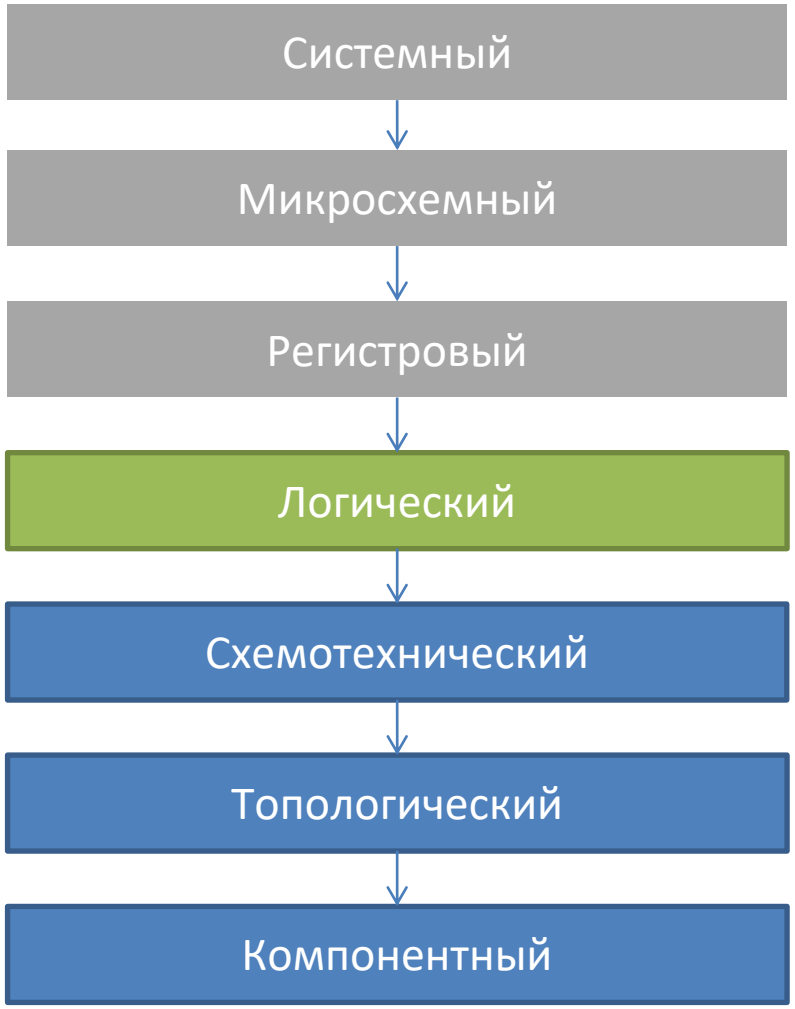


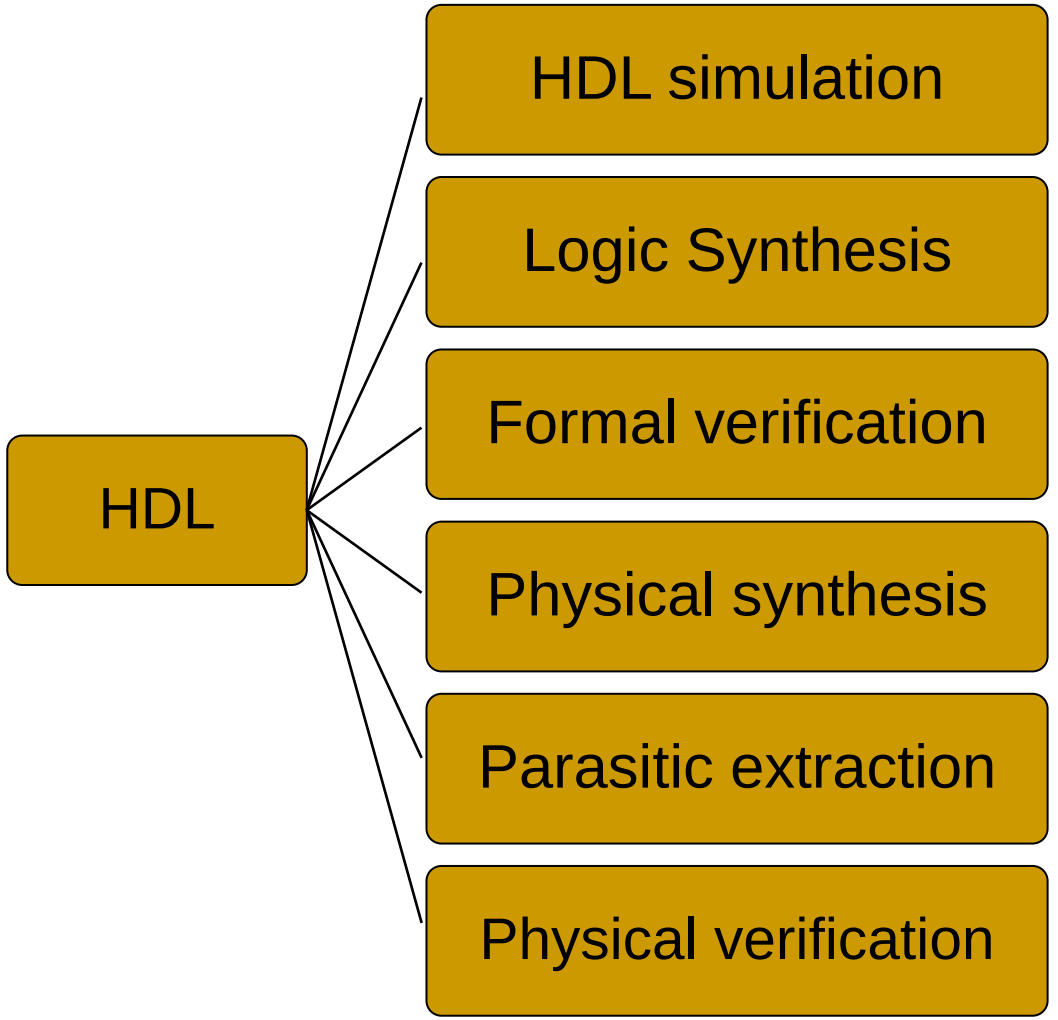
Лекция 4

Логический этап проектирования

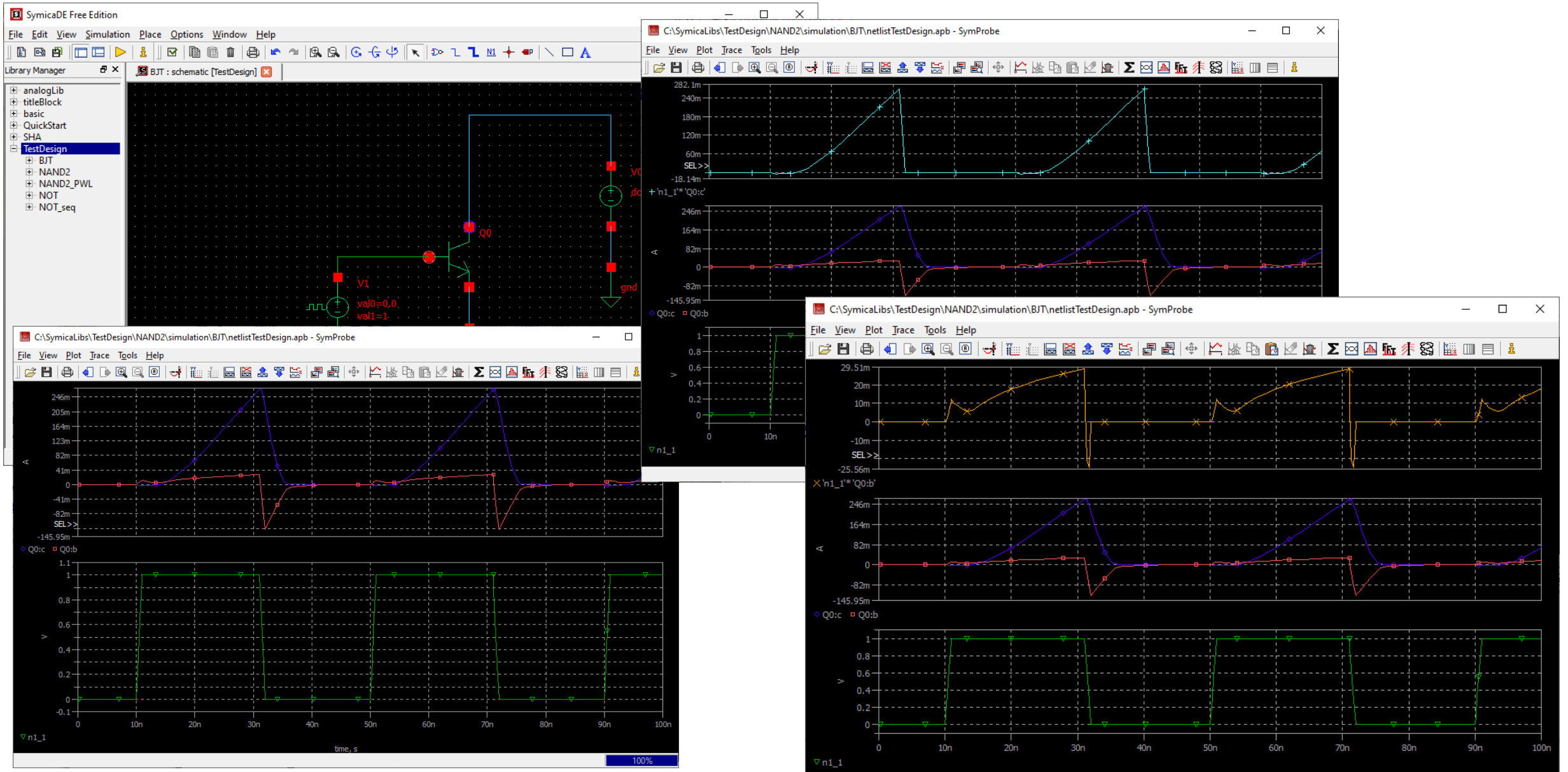
Часть 1

Этапы проектирования: логический

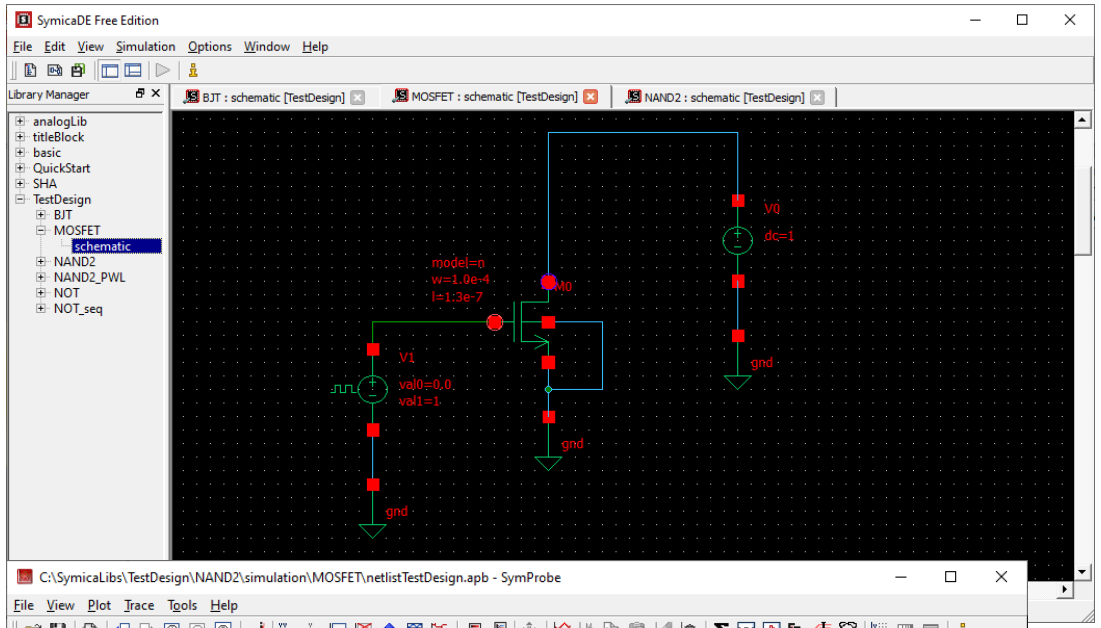


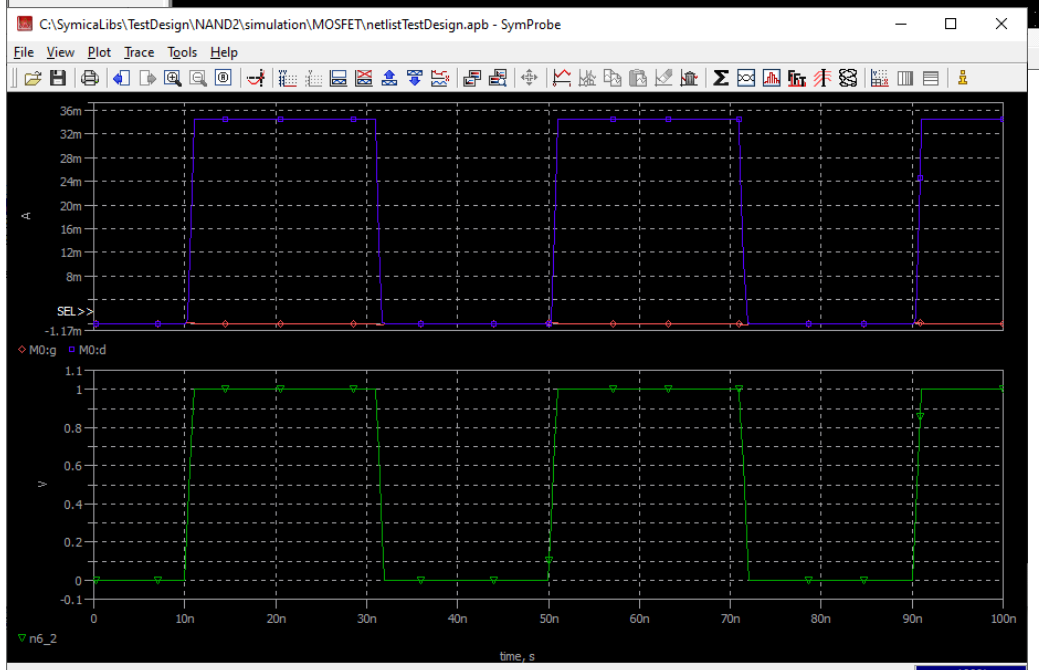


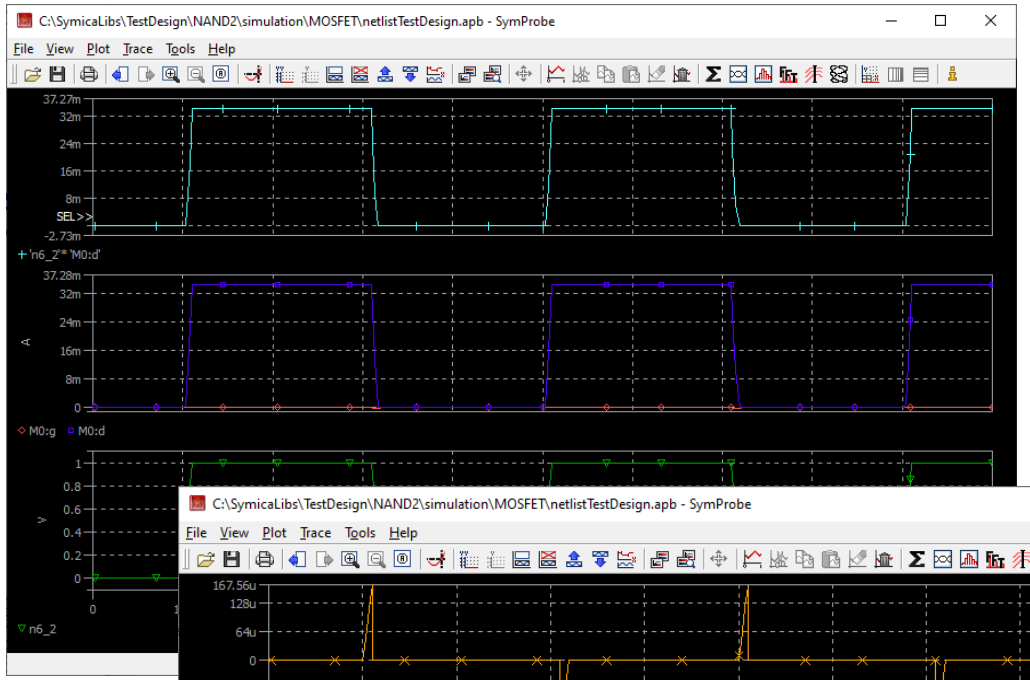
МДП транзистор vs биполярный транзистор: потребление (1)

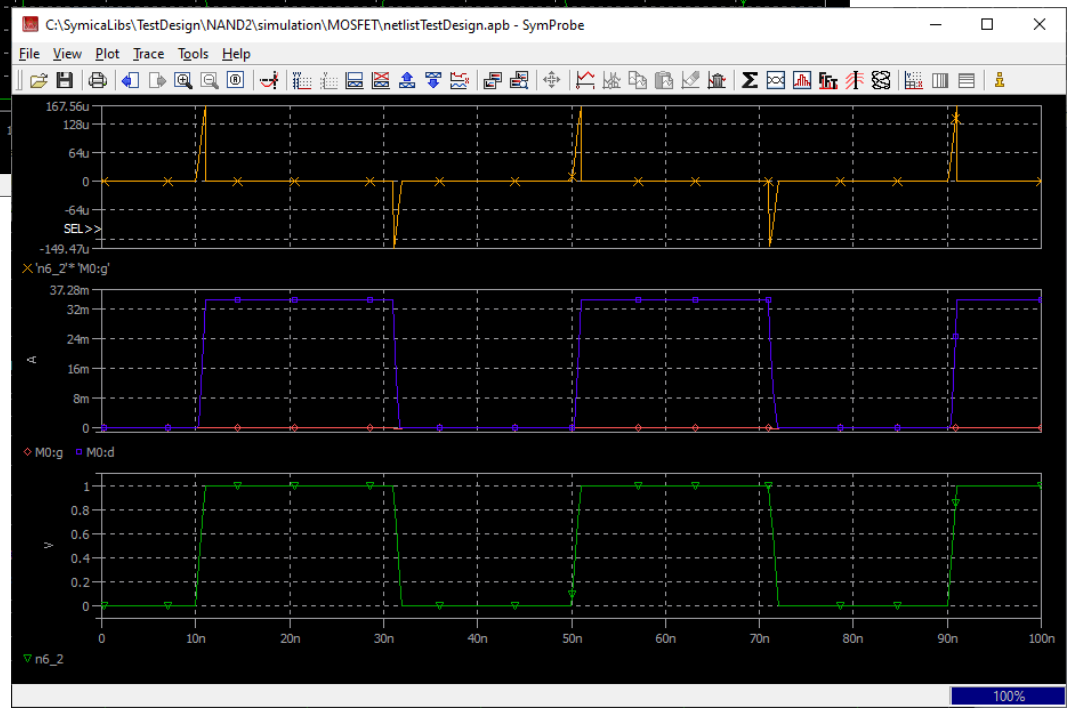


МДП транзистор vs биполярный транзистор: потребление (2)

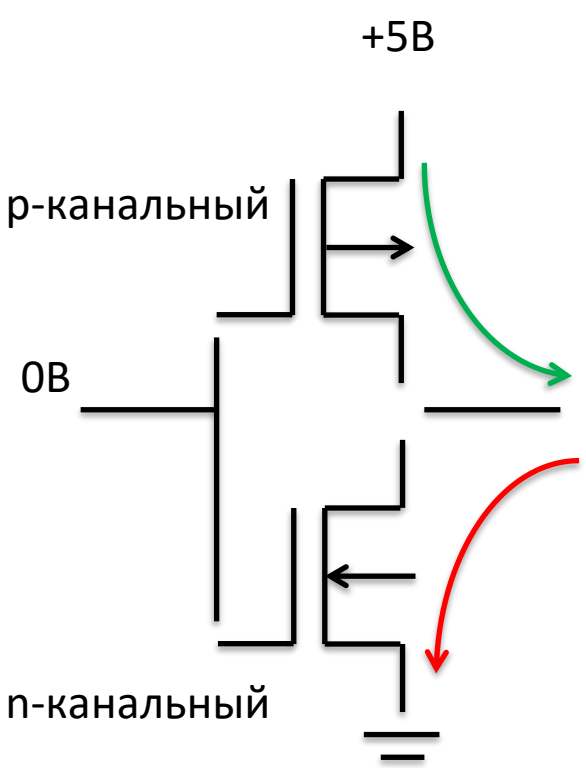








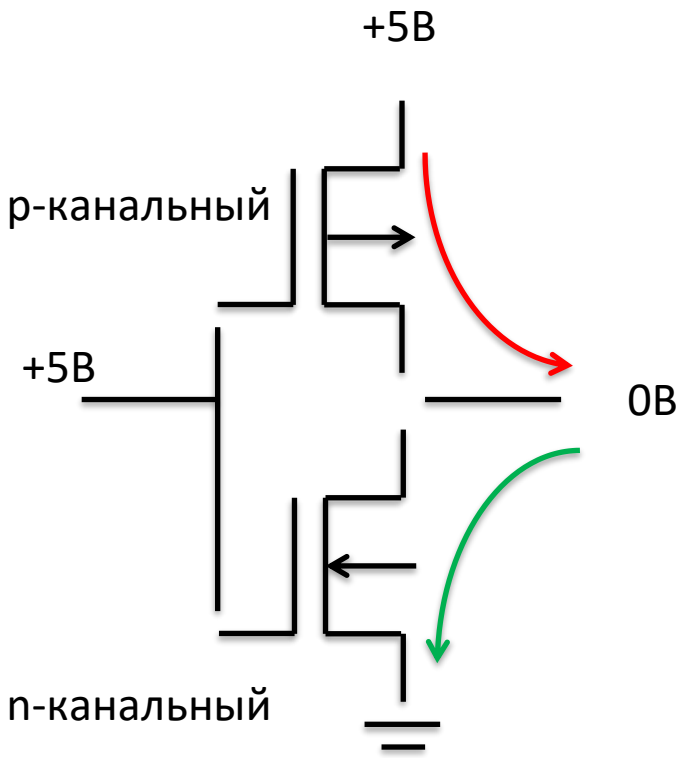
Принцип работы цифровых схем



без напряжения на затворе **пропускает** ток

+5В

без напряжения на затворе **не пропускает** ток



Минимальный логический базис

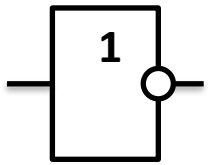
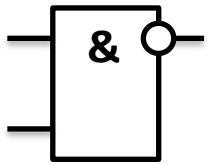
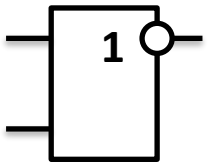
Графическое обозначение			
Название элемента	Инвертор	2И-НЕ	2ИЛИ-НЕ
Логическая функция	**NOT**	**NAND2**	**NOR2**

Таблица истинности

x	y
0	1
1	0

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

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

Представление логики работы схем

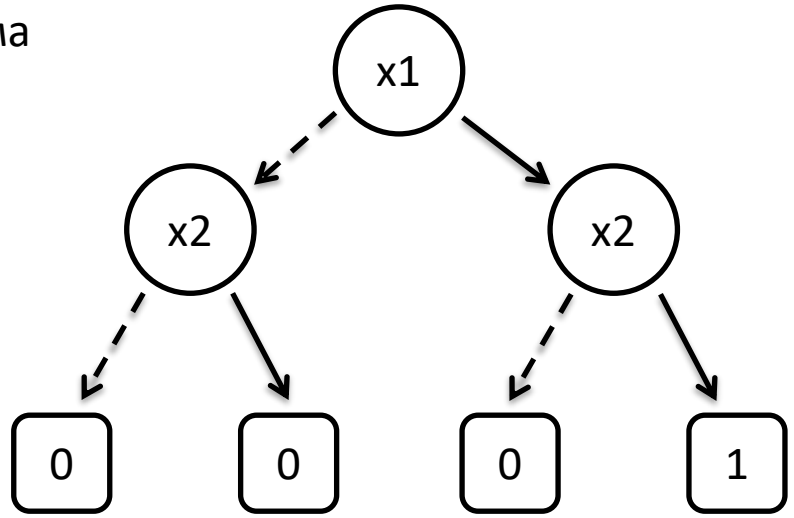
Таблица истинности

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

Карта Карно

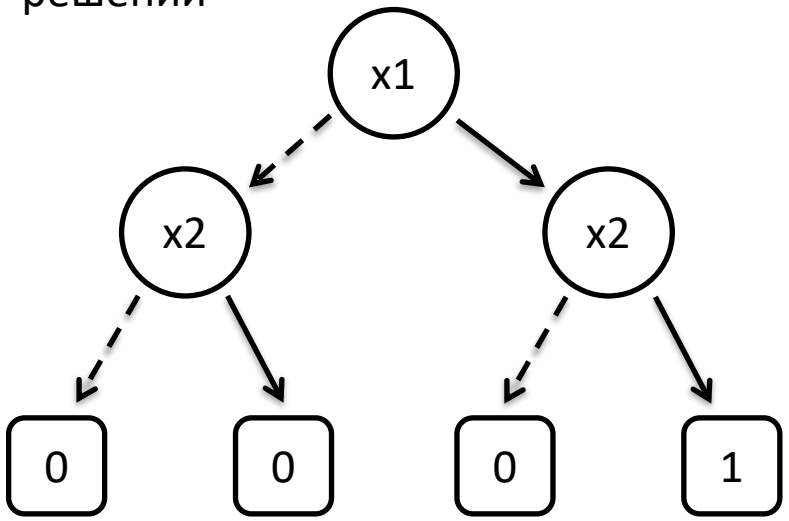
x1 \ x2	0	1
0	0	0
1	0	1

Диаграмма двоичных решений

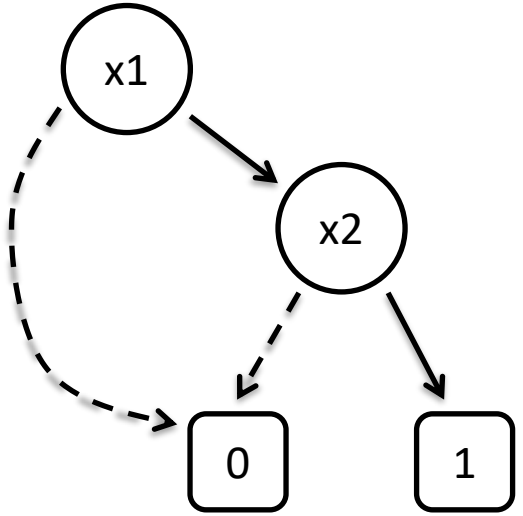


Формы диаграмм двоичных решений

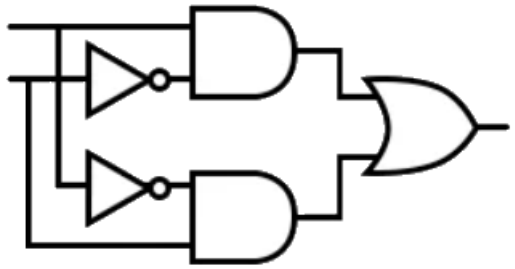
Полная диаграмма двоичных решений



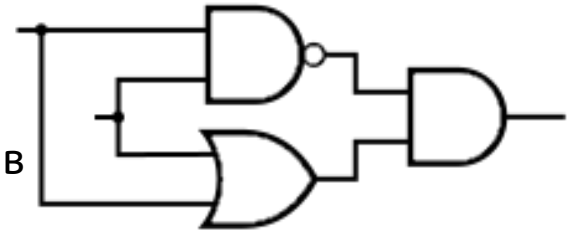
Сокращённая диаграмма двоичных решений



Разница между представлениями вентилей в различных базисах

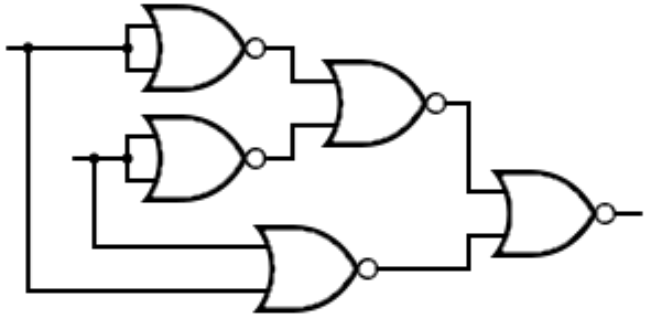


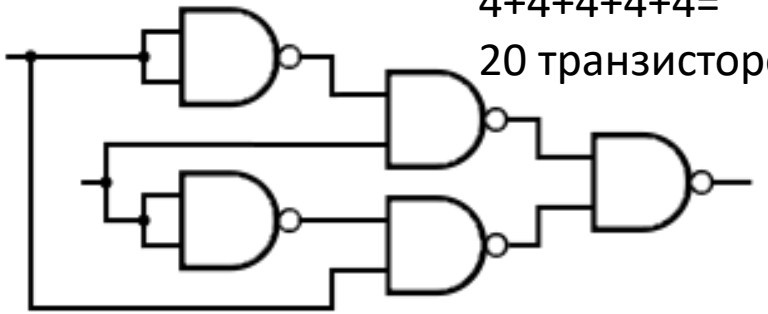
2+2+6+6+6=
22 транзистора



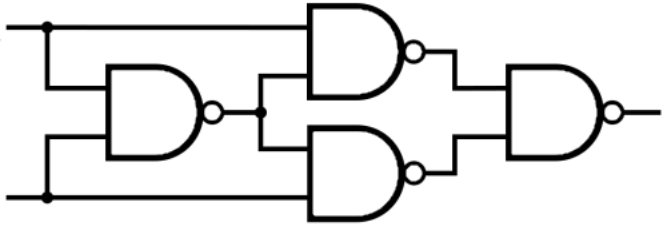
4+6+6=
16 транзисторов

4+4+4+4+4=
20 транзисторов





4+4+4+4+4=
20 транзисторов

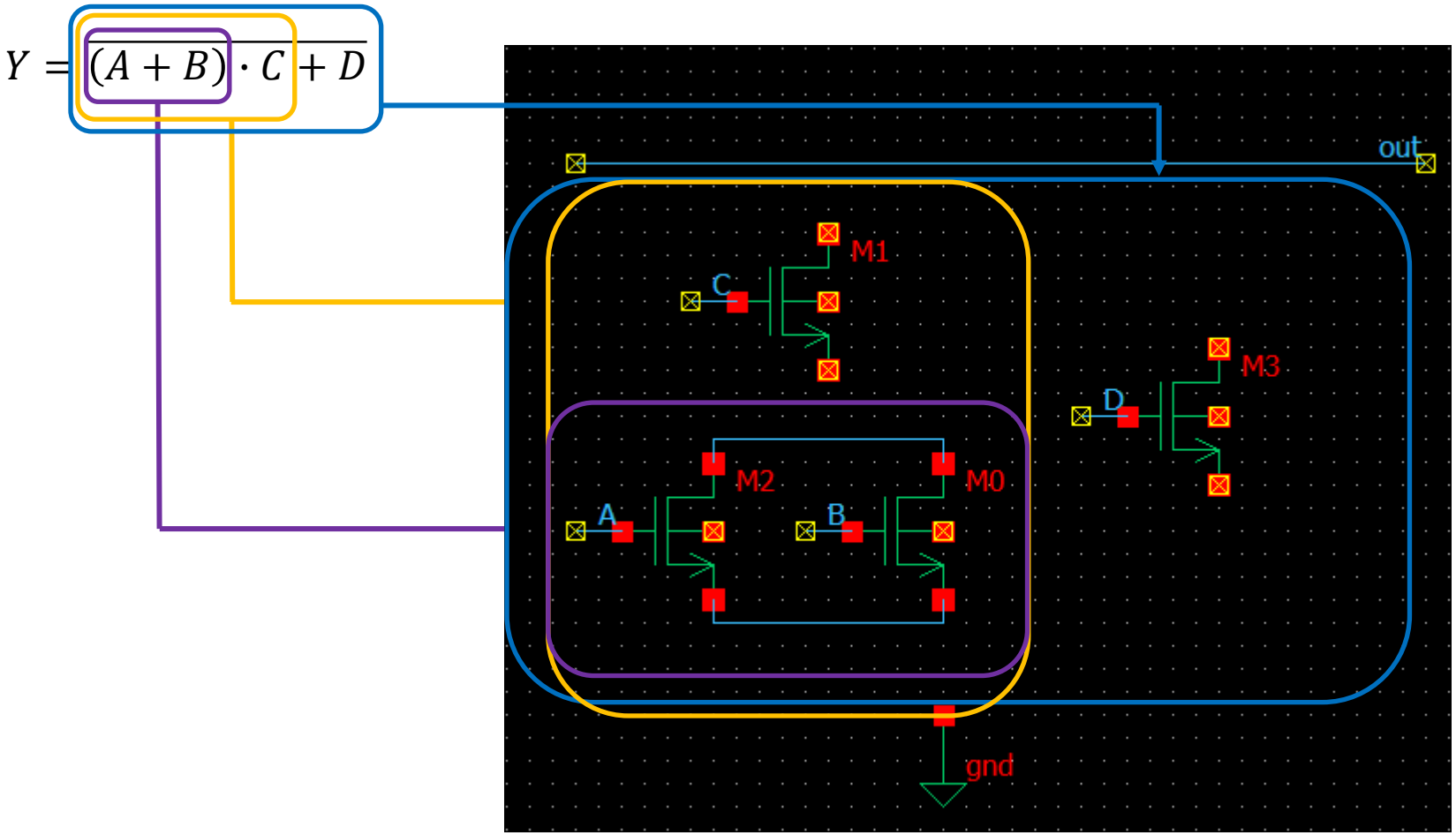


4+4+4+4=
16 транзисторов

Составление логических функций на транзисторном уровне (2)

Задача: составить схему для функции: $Y = \overline{(A + B) \cdot C + D}$

Шаг 1. Составляем часть схемы на nmos (нижняя часть схемы, под выходом)

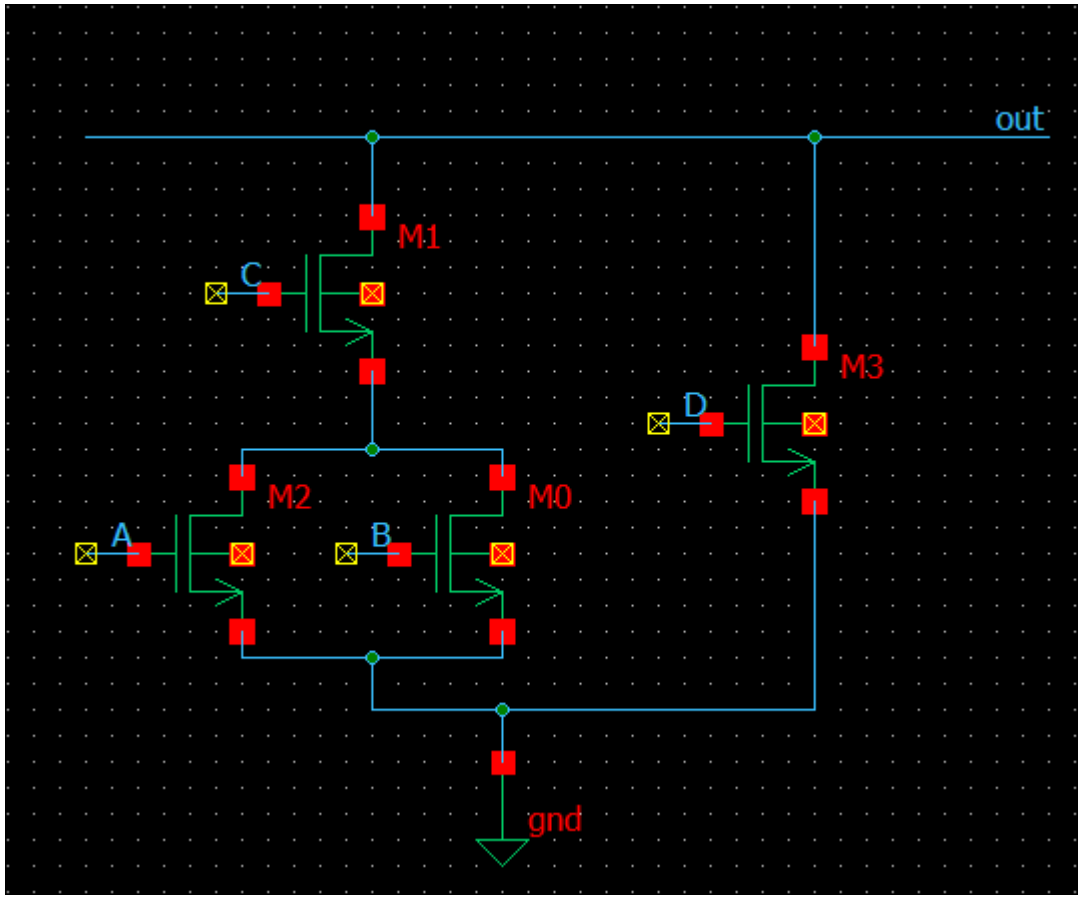


Составление логических функций на транзисторном уровне (3)

Задача: составить схему для функции: $Y = \overline{(A + B) \cdot C + D}$

Шаг 1. Составляем часть схемы на nmos (нижняя часть схемы, под выходом)

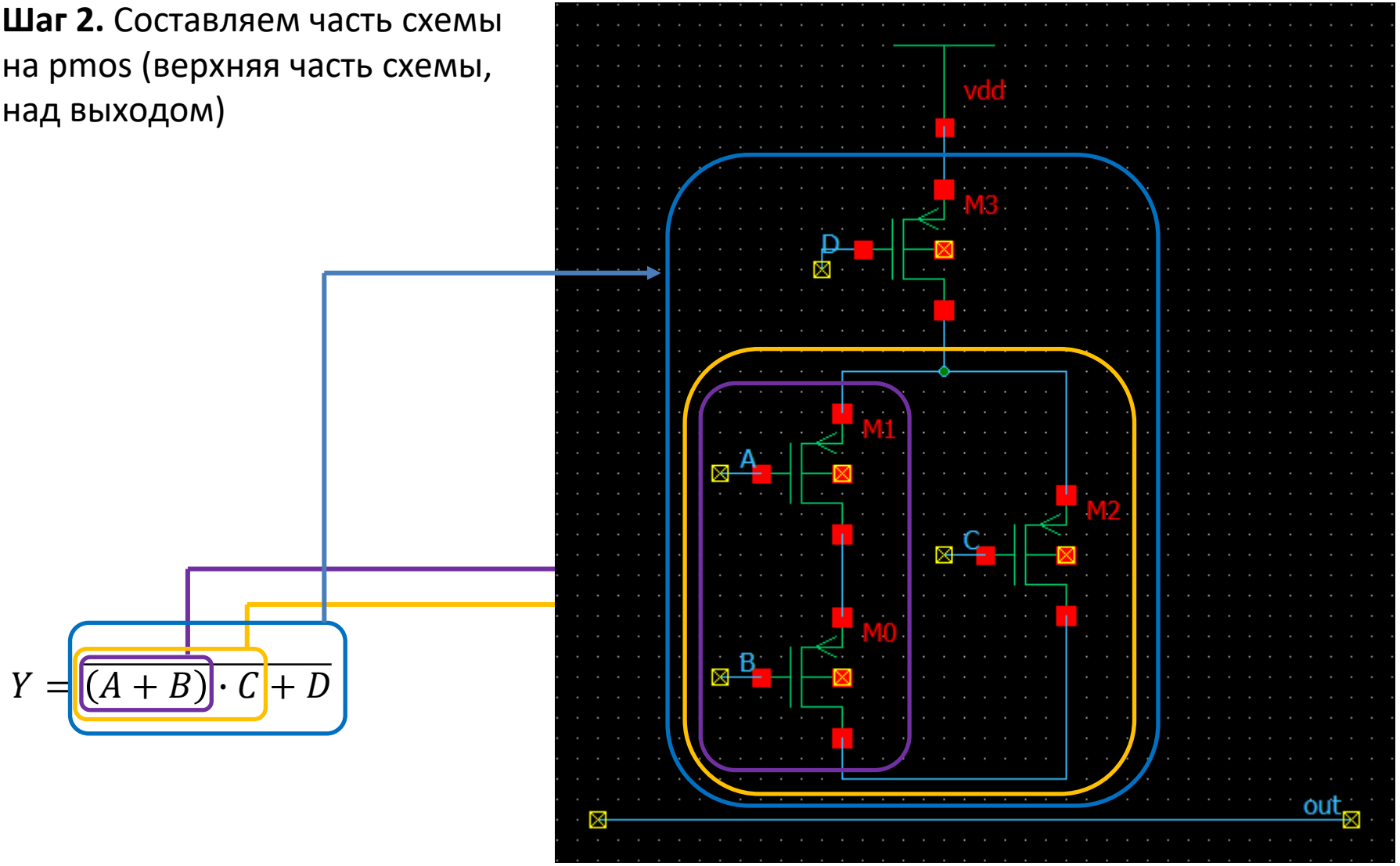
$Y = \overline{(A + B) \cdot C + D}$



Составление логических функций на транзисторном уровне (4)

Задача: составить схему для функции: $Y = \overline{(A + B) \cdot C + D}$

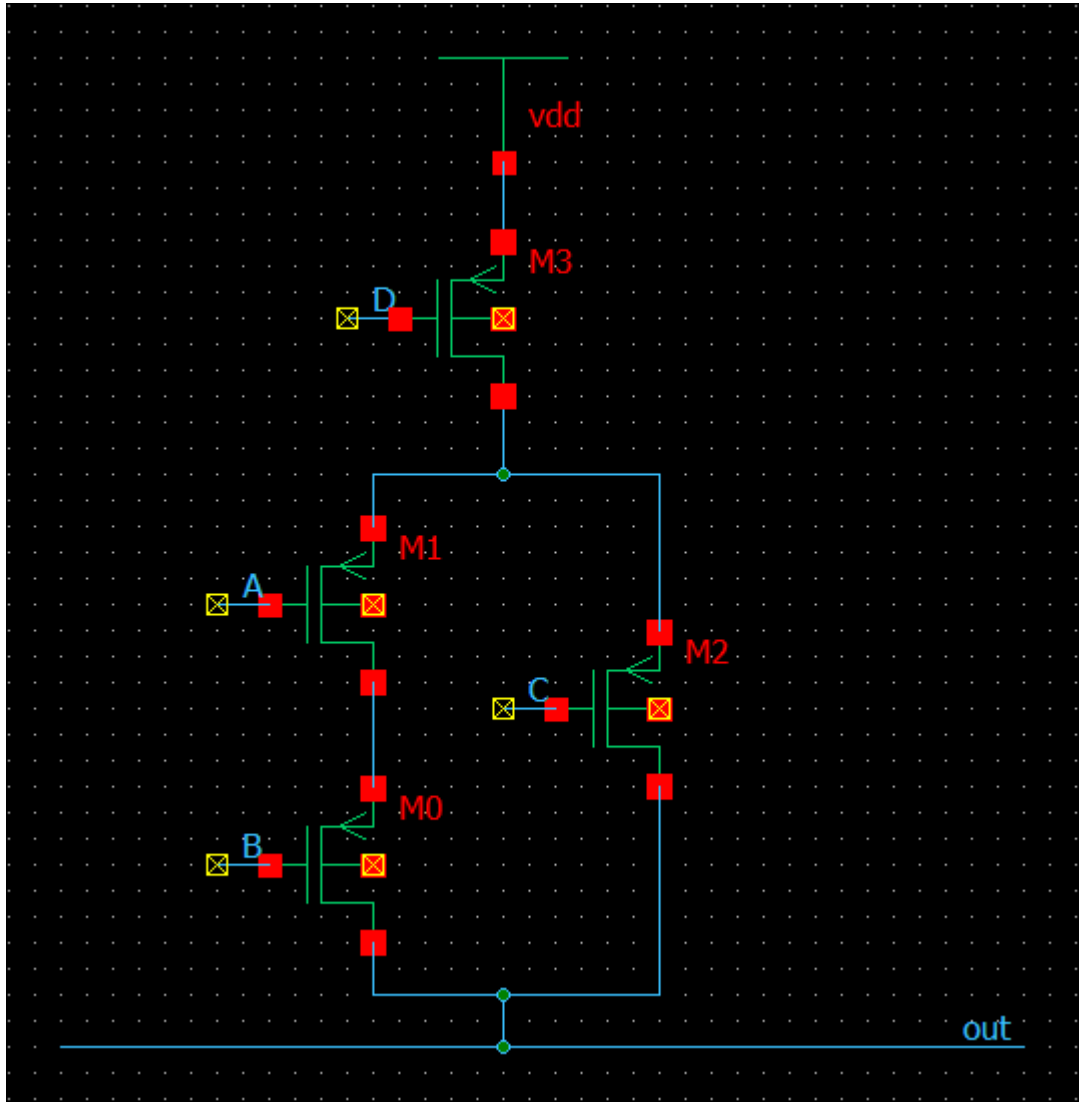
Шаг 2. Составляем часть схемы на pmos (верхняя часть схемы, над выходом)



Составление логических функций на транзисторном уровне (5)

Задача: составить схему для функции: $Y = \overline{(A + B) \cdot C + D}$

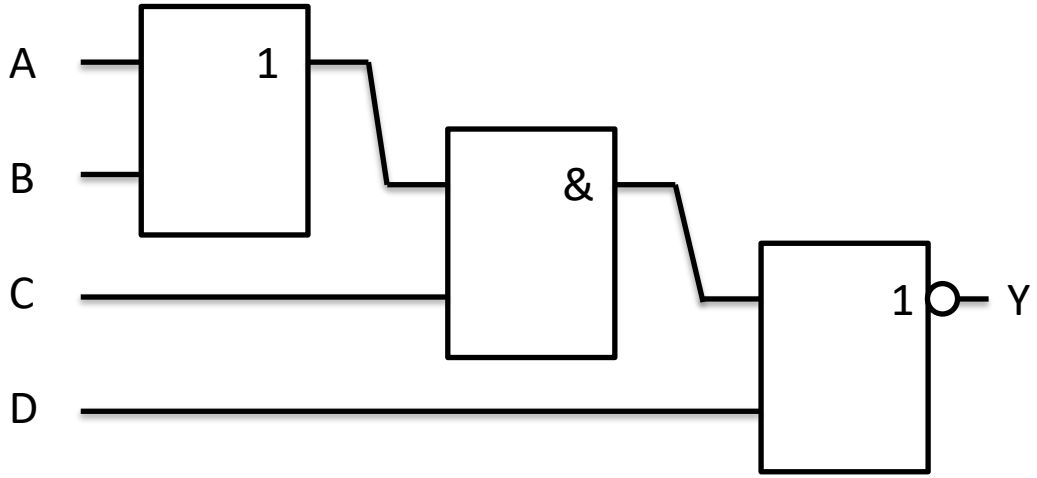
Шаг 2. Составляем часть схемы на pmos (верхняя часть схемы, над выходом)

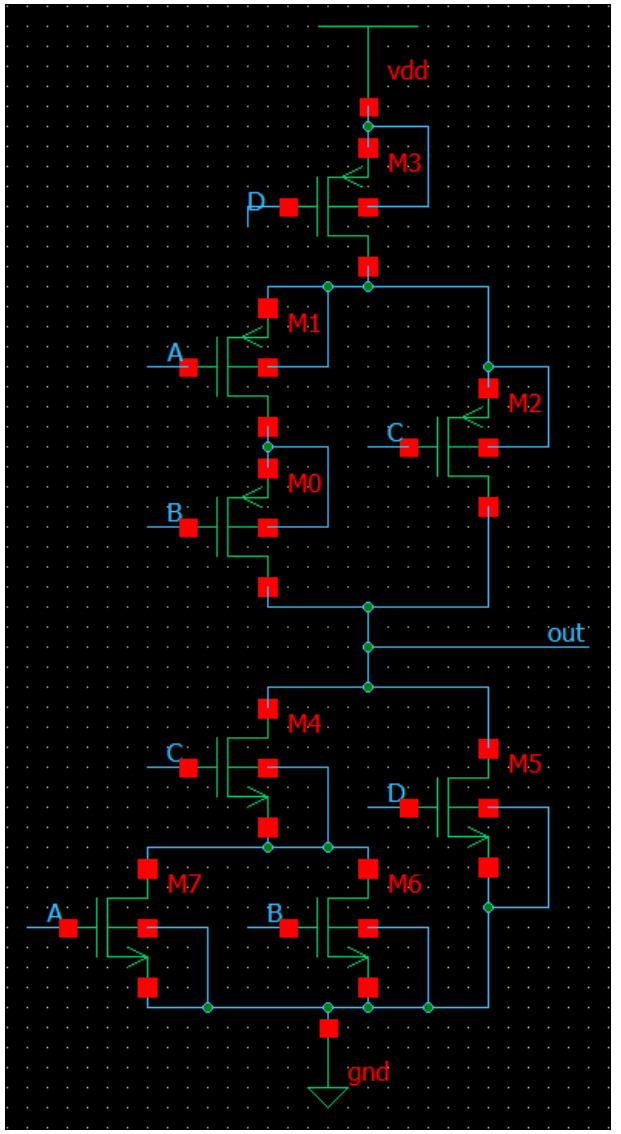


$Y = \overline{(A + B) \cdot C + D}$

Составление логических функций на транзисторном уровне (6)

Задача: составить схему для функции: $Y = \overline{(A + B) \cdot C + D}$





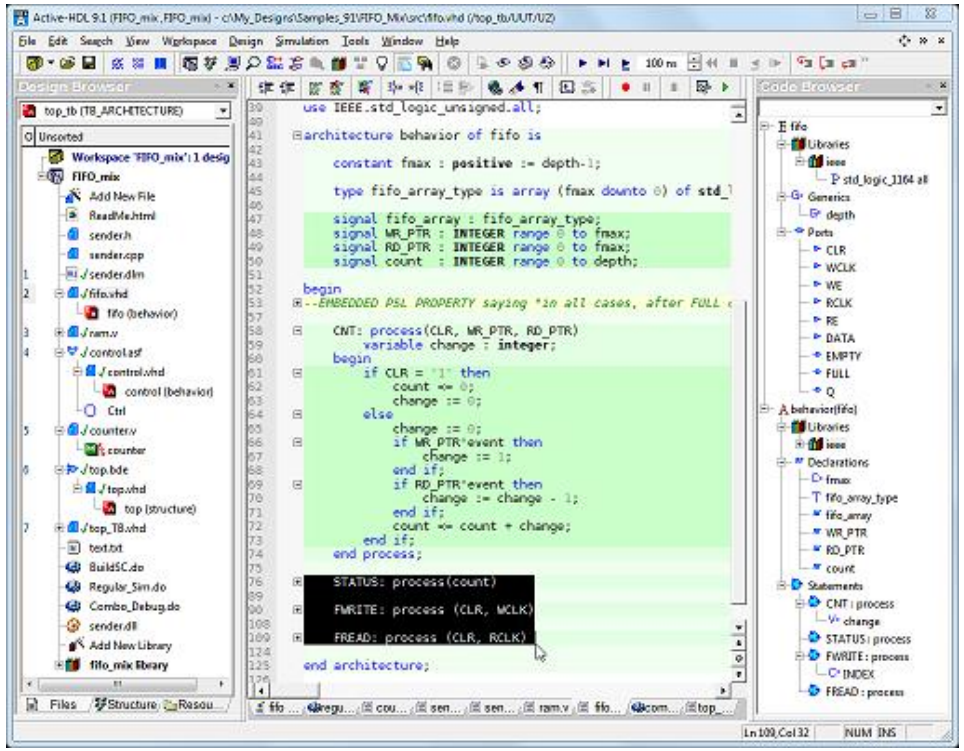
Основные языки описания цифровых схем

Verilog HDL

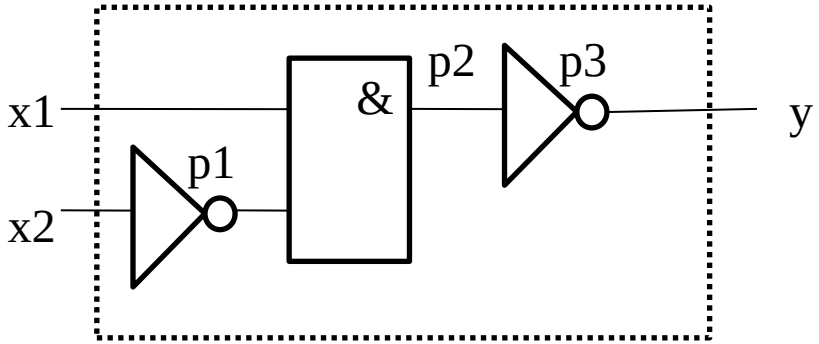
```
module inv(y, x);
  output y;
  input  x;

  assign y = ~x;
endmodule
```

VHDL

```
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;
```



Уровни абстракции при описании схем



Уровни абстракции (способы представления) цифровых устройств

Структурное описание

Structural

STR

Регистровое описание

Register Transfer Level

RTL

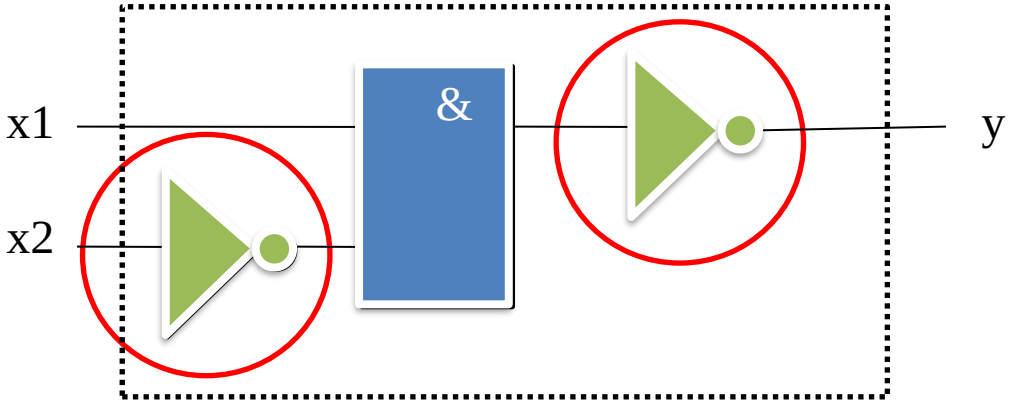
Поведенческое описание

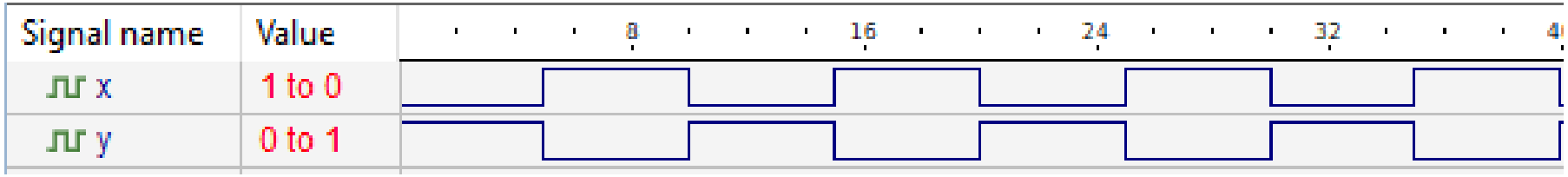
Behavioral

BEH

Регистровое описание на Verilog HDL (1)

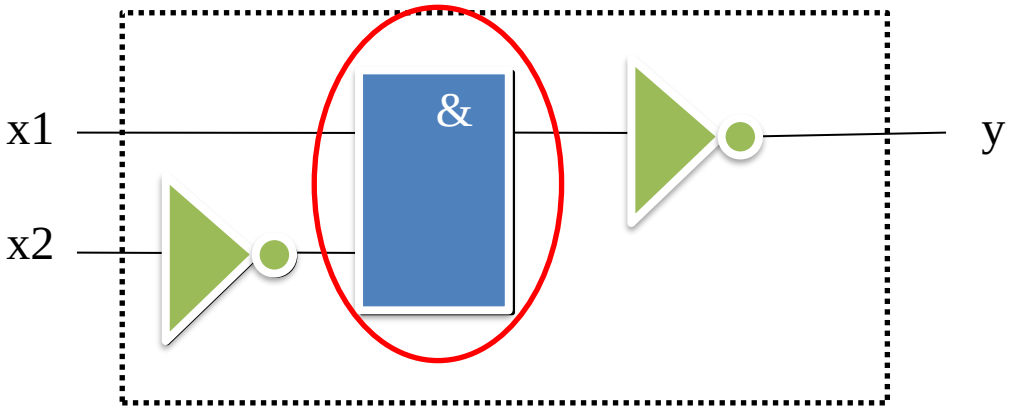
```
module inverter(x, y);
  input  x;
  output y;

  assign y = ~x;
endmodule
```

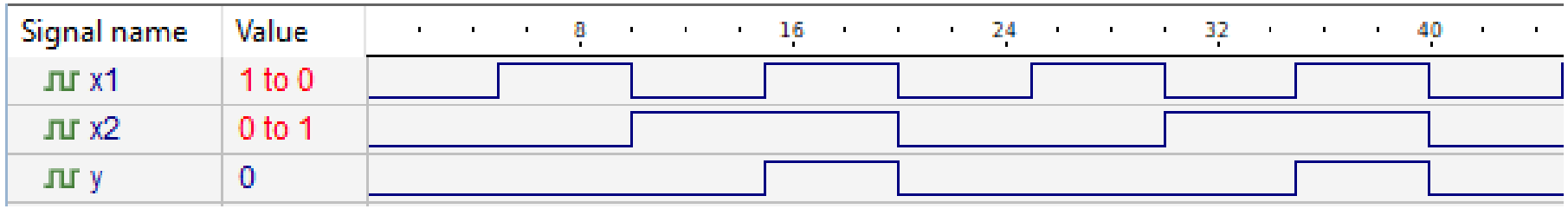




Регистровое описание на Verilog HDL (1)

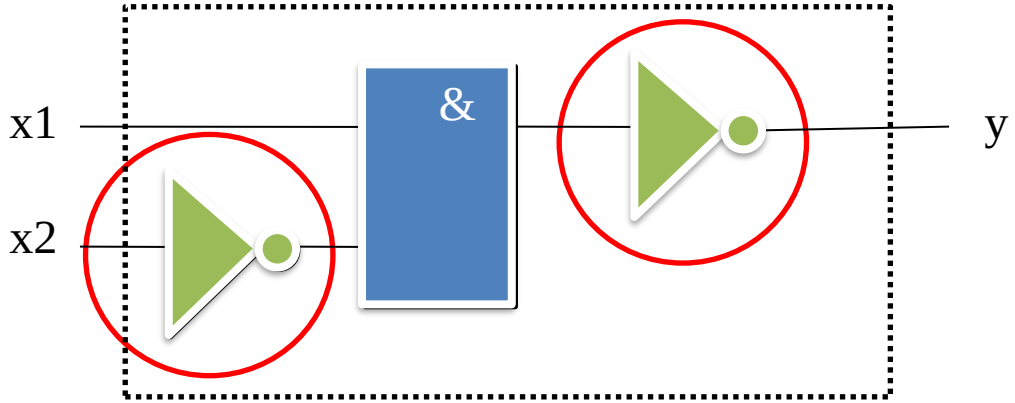
```
module and2(x1, x2, y);
  input  x1, x2;
  output y;

  assign y = x1 & x2;
endmodule
```

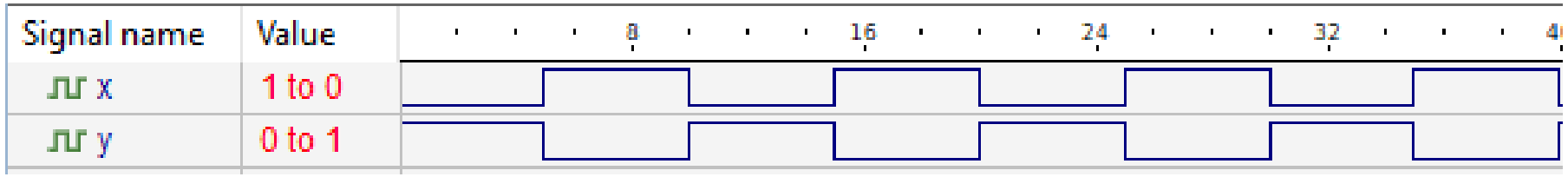




Регистровое описание на VHDL (1)

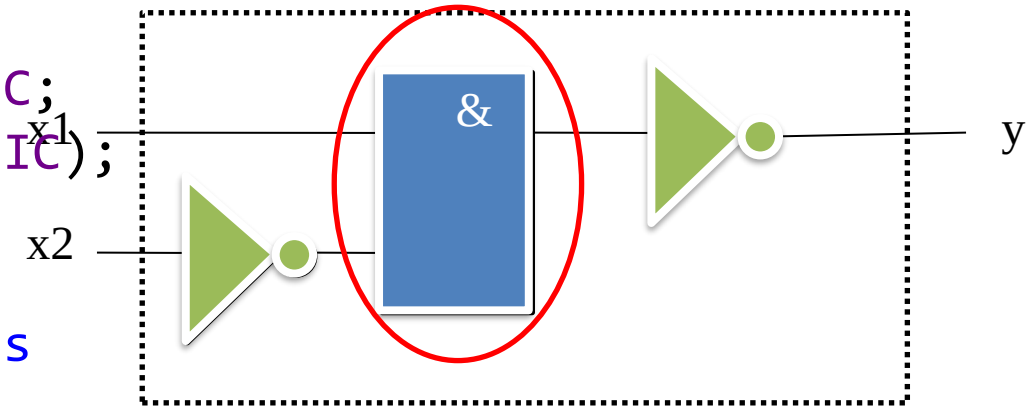
```
entity inv is
  port(x: in STD_LOGIC;
       y: out STD_LOGIC);
end inv;

architecture RTL of inv is
begin
  y <= not x;
end RTL;
```

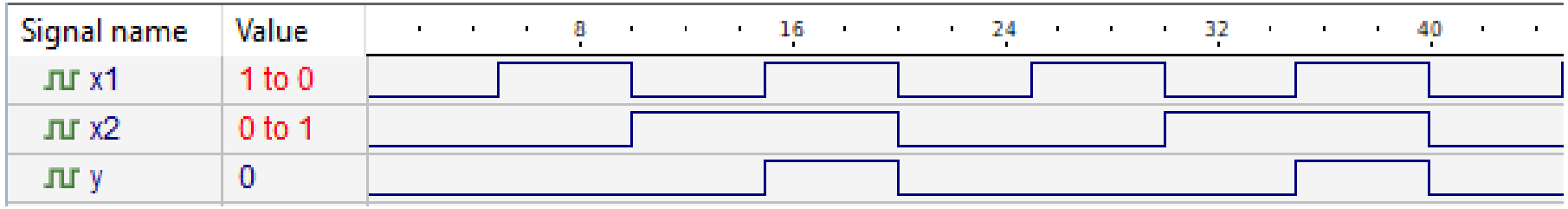




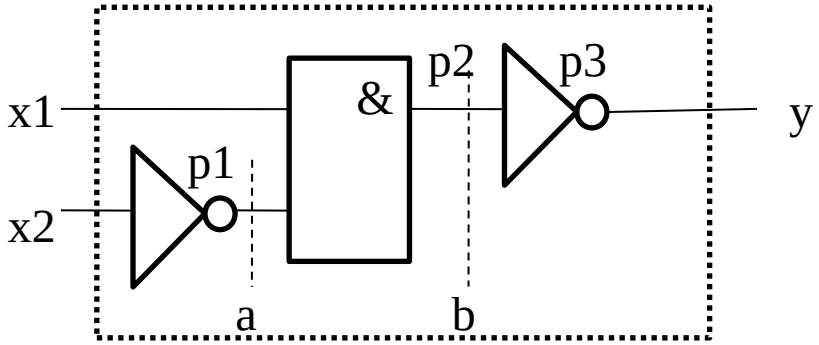
Регистровое описание на VHDL (2)

```
entity and2 is
  port(x1, x2: in STD_LOGIC;
               y: out STD_LOGIC);
end and2;

architecture RTL of and2 is
begin
  y <= x1 and x2;
end RTL;
```





Структурное (вентильное) описание всей схемы на Verilog HDL (gate-level)

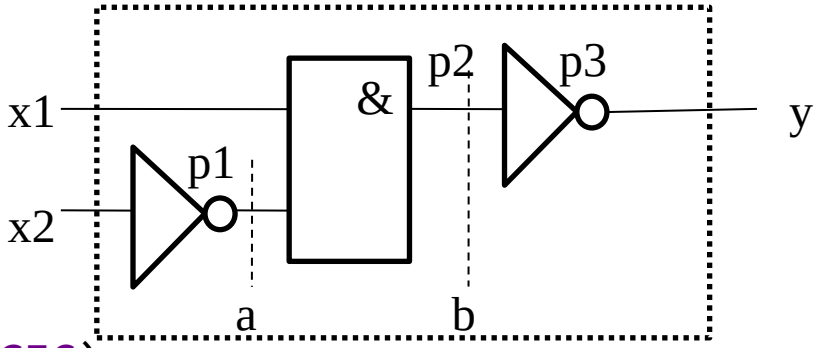



```
module device(x1, x2, y);
  input  x1, x2;
  output y;

  wire a, b;

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

endmodule
```

Перечень недостающих сигналов

Связь элементов посредством сигналов и портов

Структурное (вентильное) описание всей схемы на VHDL (gate-level)




```
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;
```

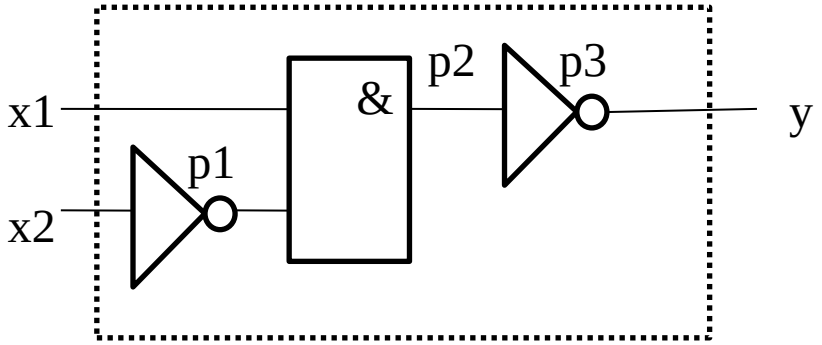
Какие типы элементов будут использованы?

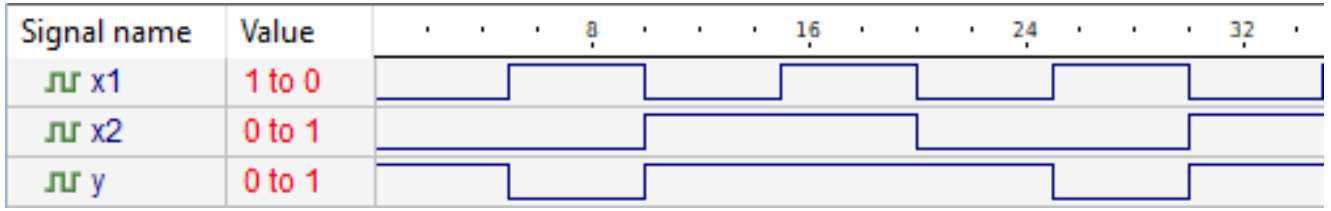
Перечень недостающих сигналов

Связь элементов посредством сигналов и портов

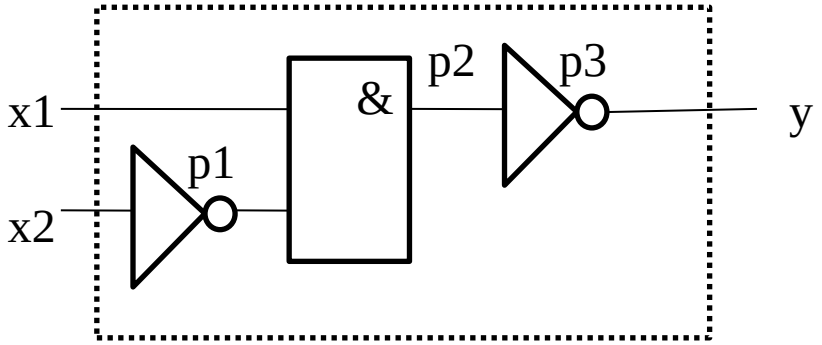
Написание тестового окружения на Verilog HDL

```
module tb;
  reg  x1, x2;
  wire y;

  device u(x1, x2, y);

  initial
  begin
    #0 x1 = 0;
       x2 = 0;
  end

  always #5 x1 = ~x1;

  always #10 x2 = ~x2;

endmodule
```

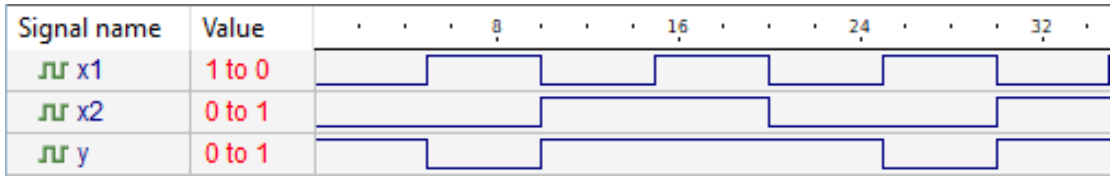




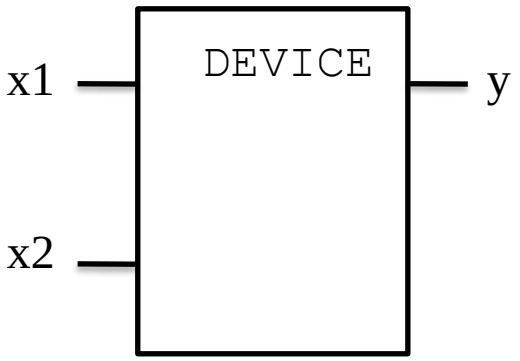
Написание тестового окружения на VHDL

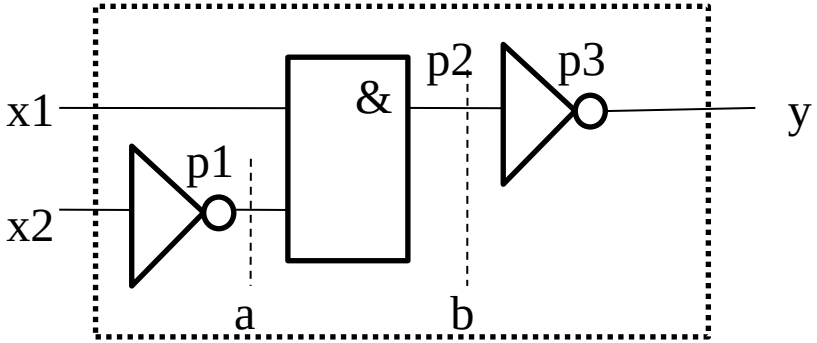



```
entity tb is
end tb;

architecture TEST of tb 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
  p1 : device port map(x1, x2, y);

  p2 : process
       begin
         x1 <= not x1;
         wait for 5 ns;
       end process;
```



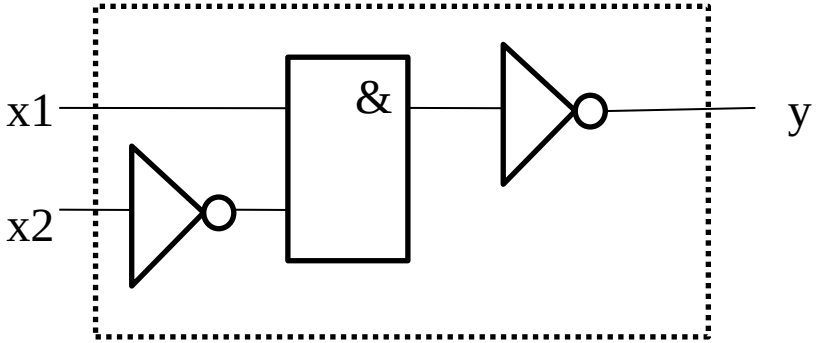
Поведенческое (функциональное) описание





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

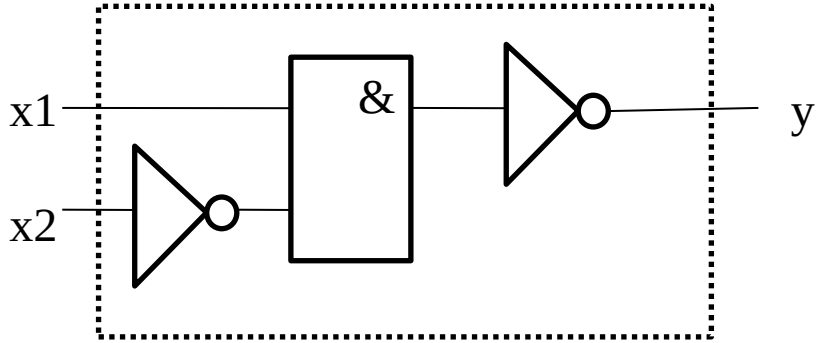
Поведенческое (функциональное) описание на языке Verilog HDL




```
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
```

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

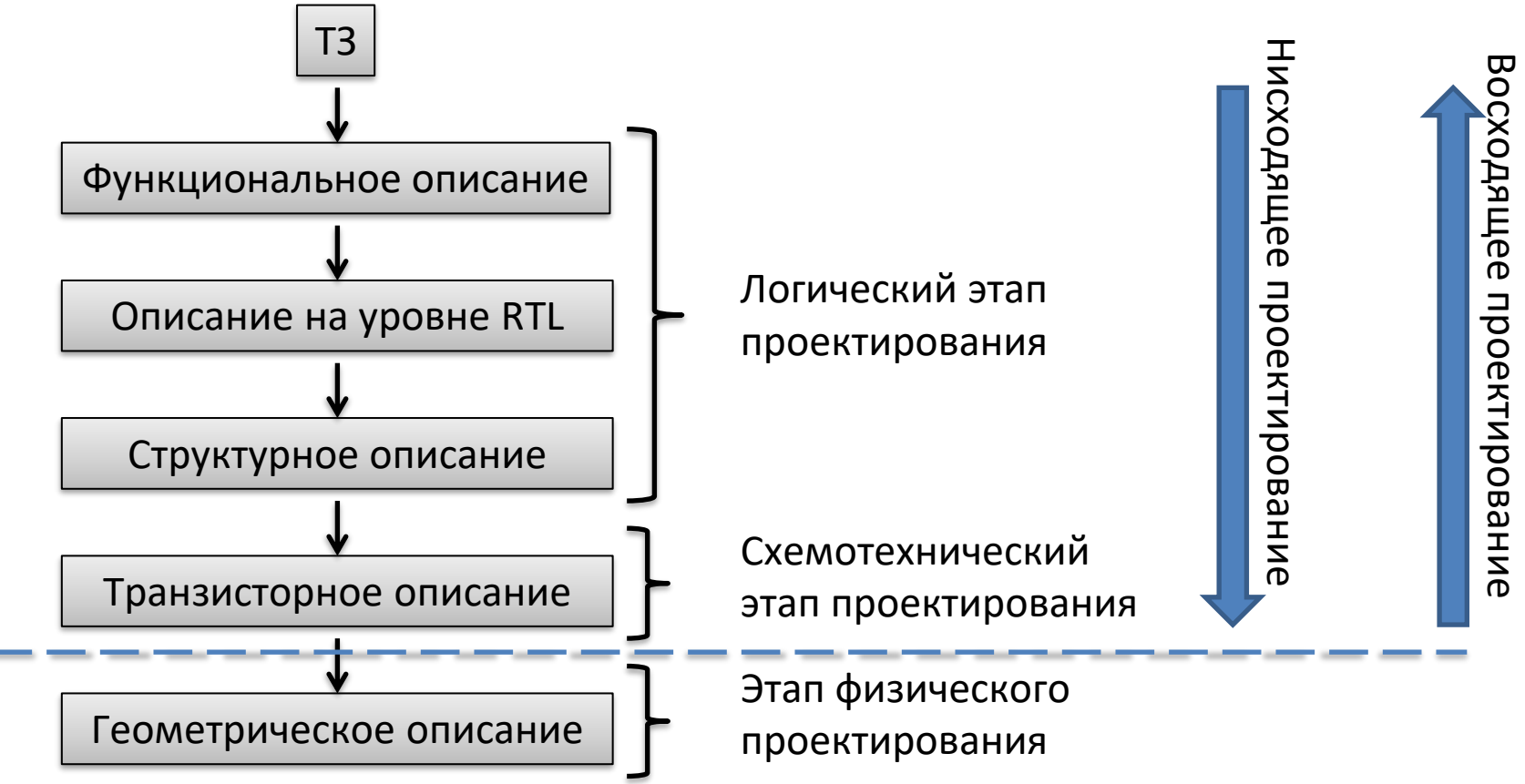
Поведенческое (функциональное) описание на языке VHDL



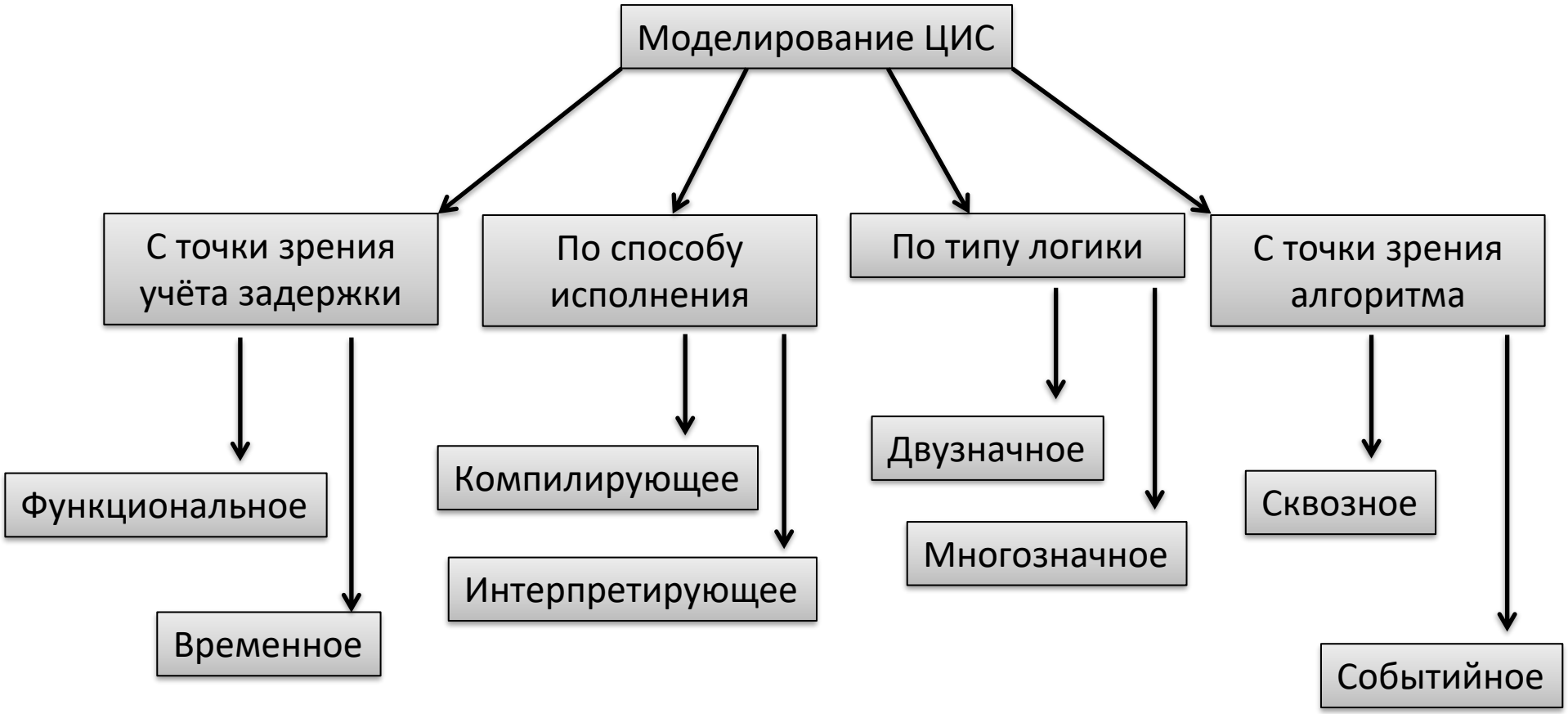

```
architecture BEH of DEVICE is
begin

  process(x1, x2)
  begin
    if(x1='1' and x2='0') then
      y <= '0';
    else
      y <= '1';
    end if;
  end process;

end BEH;
```

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

Нисходящее и восходящее проектирования



Классификация алгоритмов логического моделировнаия

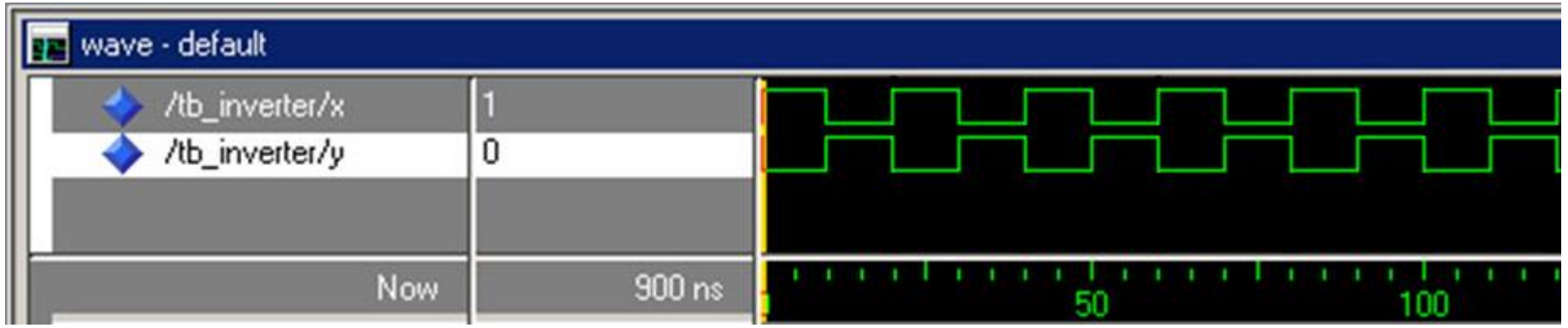


Функциональное моделирование

```
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.

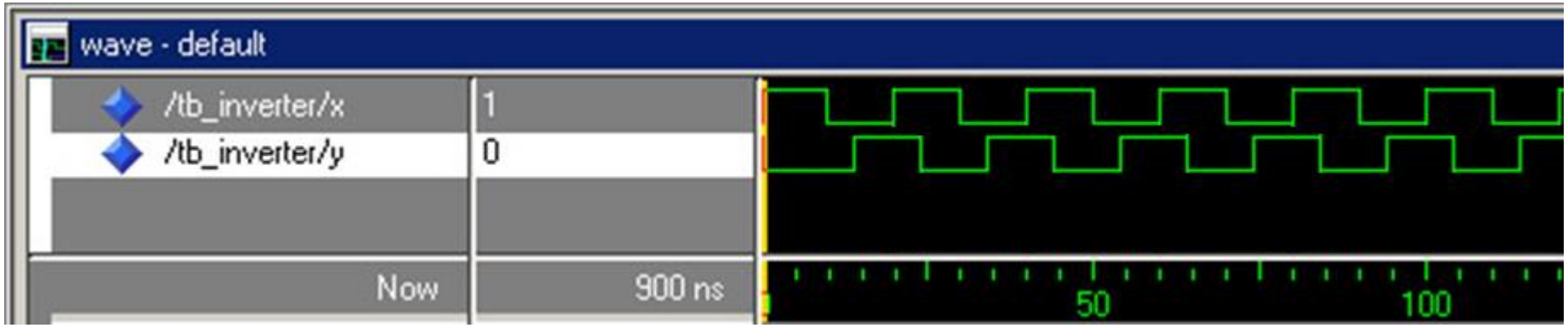


Временное моделирование

```
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;
```

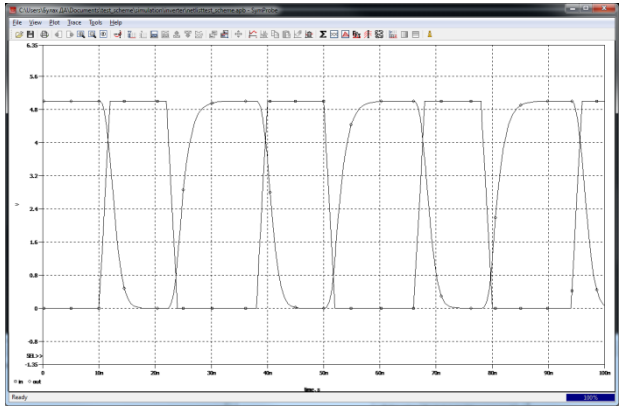
Характеристики:

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

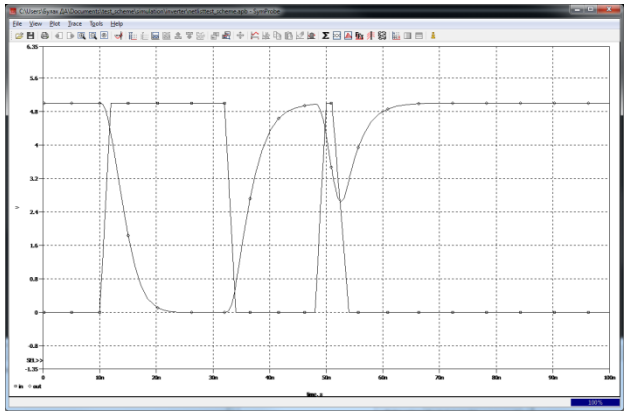


Типы задержек временного моделирования (3)

Транспортная задержка

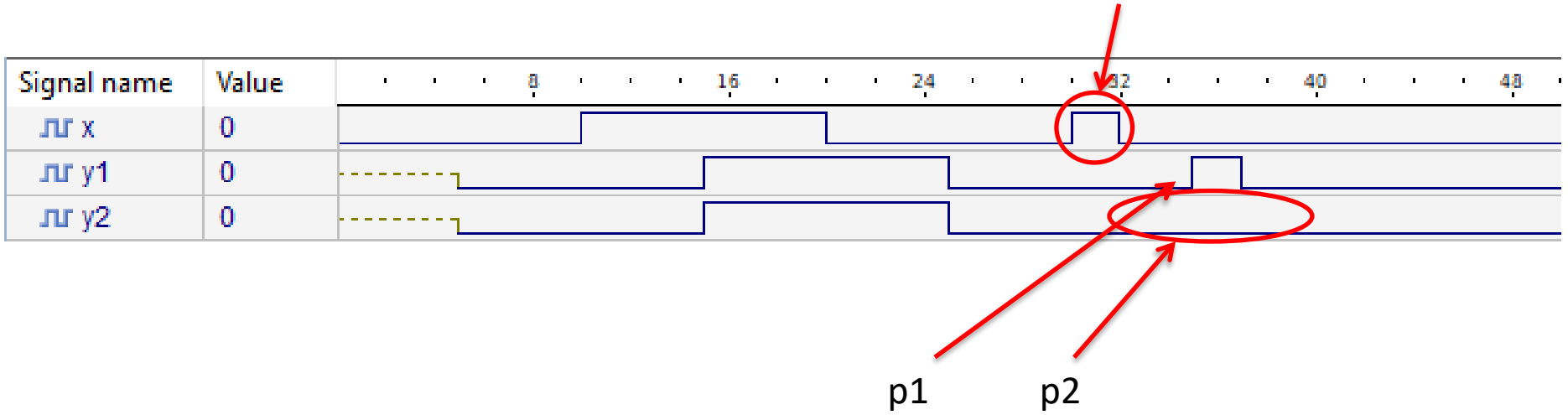


Инерциальная задержка



```
y1 <= transport x after 5 ns;
y2 <= inertial  x after 5 ns;
```

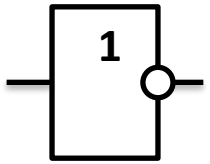
Входное воздействие длительностью 2 ns

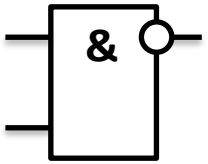


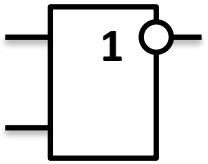
Типы логик: двузначная логика

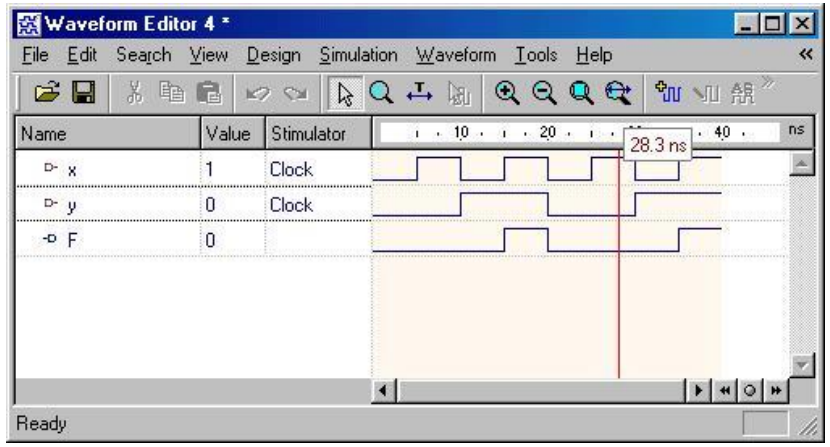
Двузначная логика

```
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

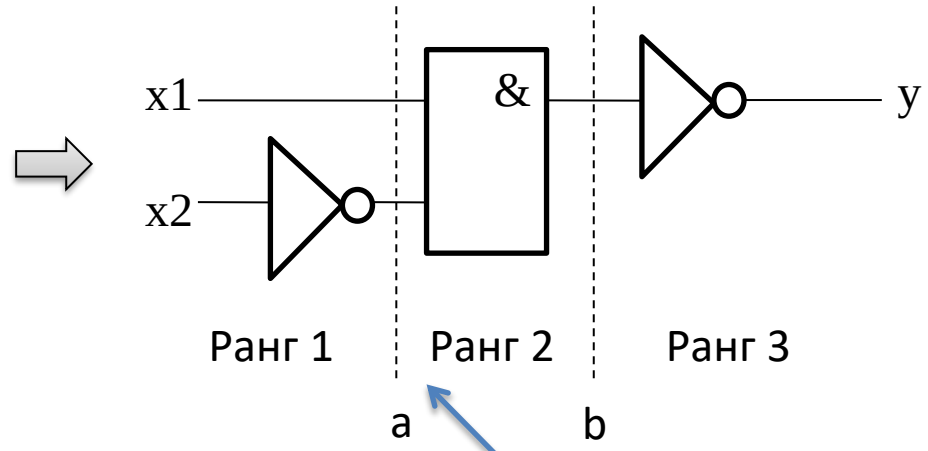
```
STD_LOGIC : {
  'U',   -- неинициализированное значение
  'X',   -- неизвестное значение
  '0',   -- логический 0
  '1',   -- логическая 1
  'Z',   -- высокий импеданс
  'W',   -- слабый сигнал, непонятно, он 0 или 1
  'L',   -- слабый сигнал, подтянутый к 0
  'H',   -- слабый сигнал, подтянутый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

Позволяет учитывать:

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

Алгоритмы моделирования: сквозное моделирование

Генератор входных воздействий

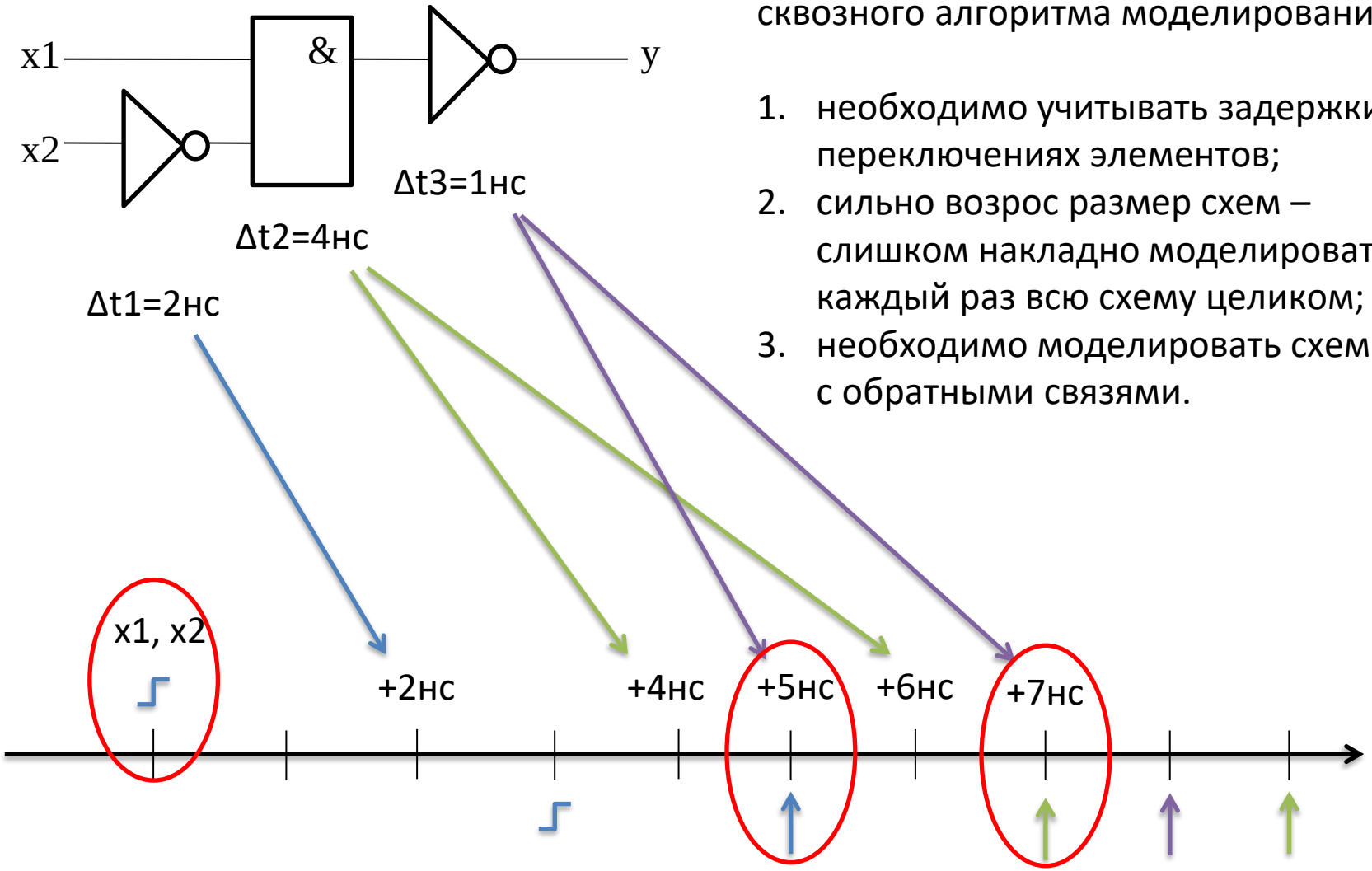



```
1. a = not x2
2. b = x1 and a
3. y = not b
```

Базовый сквозной алгоритм

- Провести ранжирование схемы
- Сформировать ММ
- Провести моделирование

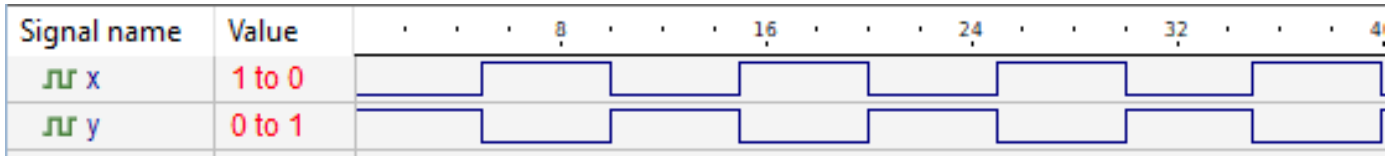
Алгоритм событийного моделирования



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.

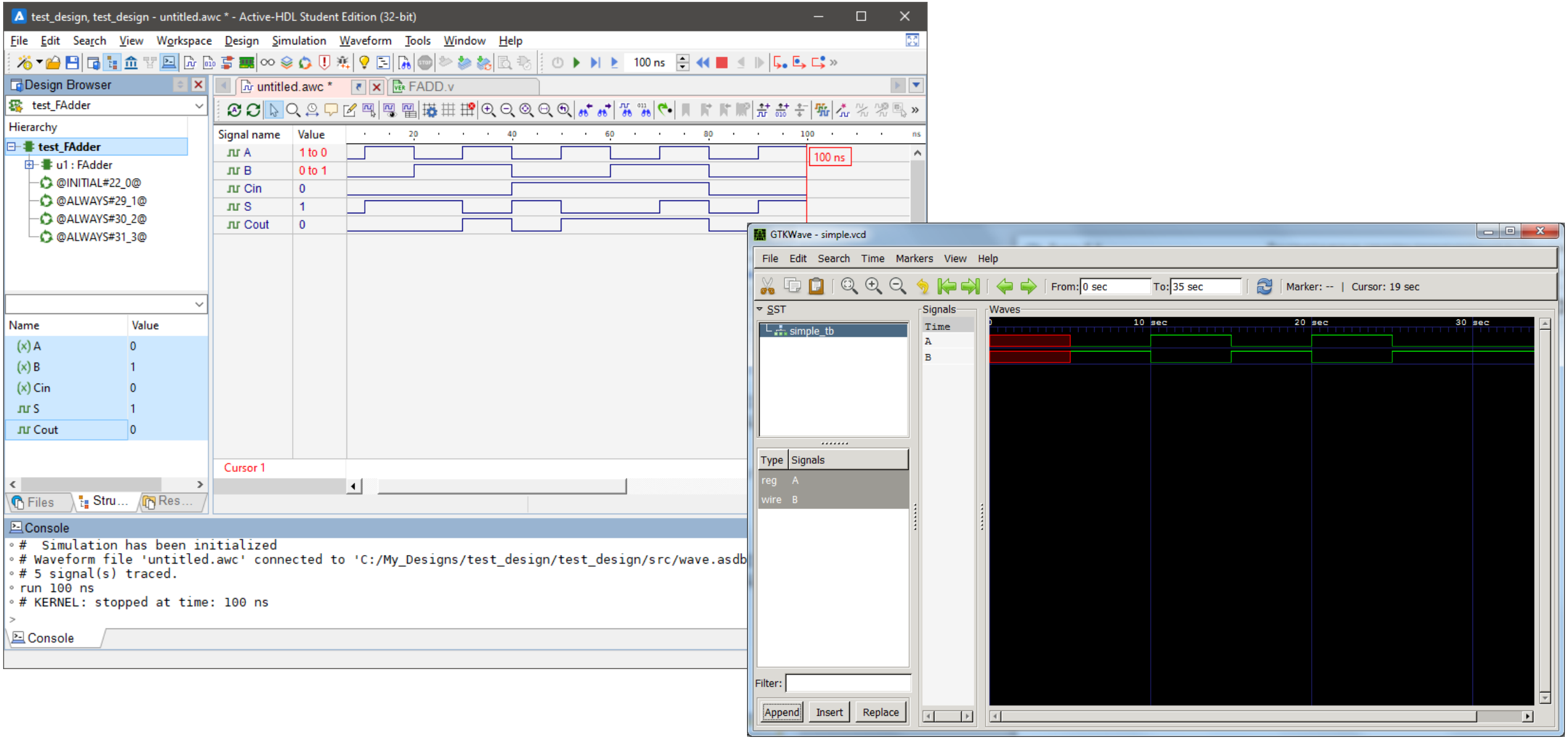
Представление результатов моделирования: формат VCD




```
$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"
```

Верификация результатов моделирования



Verilog PLI (Programming Language Interface)

```c
#include <stdio.h>
#include <vpi_user.h>

void hello() {
  printf("HELLO");
}

void register_hello() {
  s_vpi_systf_data data;
  data.type      = vpiSysTask;
  data.tfname    = "$hello";
  data.calltf    = hello;
  data.compiletf = 0;
  data.sizetf    = 0;
  data.user_data = 0;
  vpi_register_systf(&data);
}
```

```
gcc
  hello_vpi.c
  -fPIC
  -shared
  -o hello_vpi.so
```

```
module test();

  initial
  begin
    $hello;
    #10 $finish;
  end
endmodule
```

```
ncverilog
  test.v +access+r
  -loadvpi ./hello_vpi.so:register_hello
```

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;
  }
};
```

```
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;
}
```

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