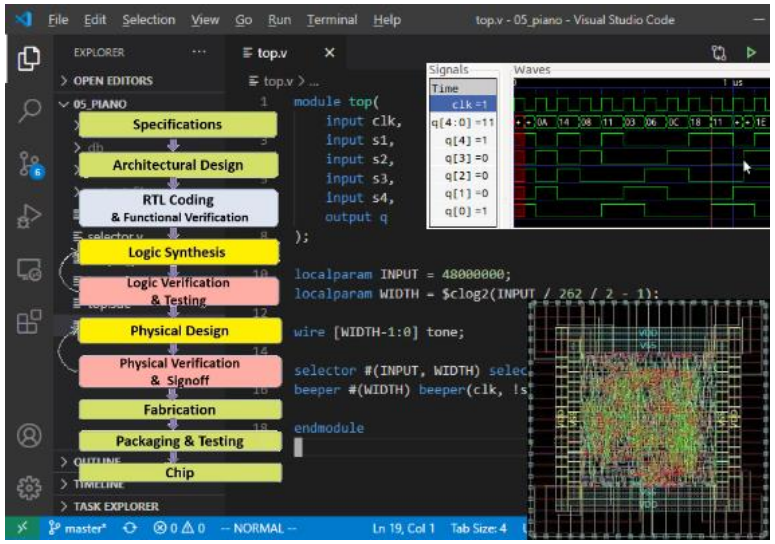
МИЭТ
Национальный исследовательский университет «МИЭТ»
Институт интегральной электроники

Лингвистические средства проектирования

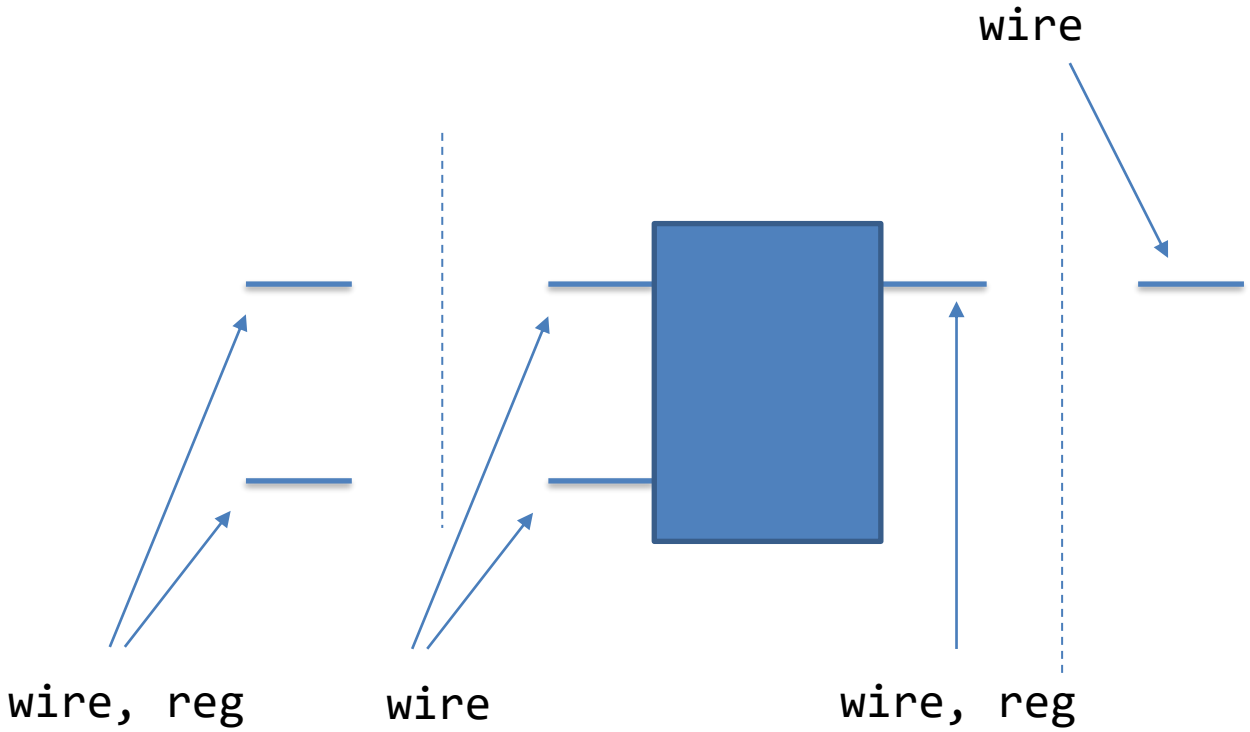


Лекция 2

Основы структурного, регистрового и поведенческого описаний

Типы данных (1)

В Verilog HDL данные - либо цепи, либо переменные.



Способы задания константных значений:
<размер>'<основание><значение>

- `4'b1011`
- `2'hFF`
- `3'o671`
- `4'b1x0z`

Данные могут быть организованы в векторы и массивы. Массивами могут быть объявлены только reg, integer, time

```
wire [3:0] data;
reg bit [1:8];
reg [3:0] mem [1:8];
```

Типы данных (2)

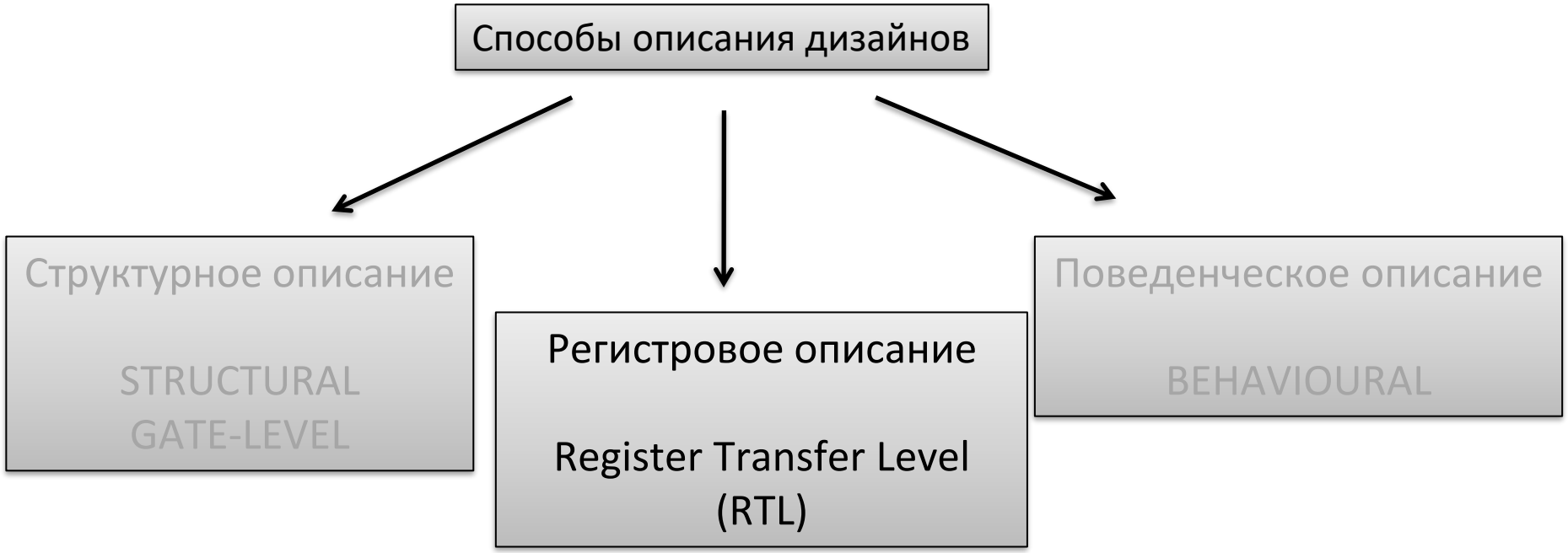
Про wire:

1. wire используется для соединения экземпляров компонентов
2. wire могут использоваться в качестве входов и выходов.
3. у wire должен быть источник сигнала.
4. wire не может быть использован в блоке always@ .
5. wire - единственный вариант при использовании конструкции assign.
6. wire используется в основном только в комбинационной логике.

Про reg:

1. reg может подсоединяться ко входу схемы.
2. reg не может подсоединяться к выходу схемы.
3. reg может использоваться в качестве выходов модуля.
4. reg не могут использоваться в качестве входов.
5. только reg используется в блоке always@.
6. только reg используется в тестбенчах для задания сигналов.
7. reg не может использоваться в конструкции assign.
8. reg может использоваться и при описании комбинационной логики, и последовательностной.

Способы описания дизайнов



Описание блока assign (1)

Синтаксис описания блока `assign`:

```
assign <net_expr> = [drive strength] [delay] <logical expressions with signals or constants>
```

Strength level	Name	Keyword
7	Supply drive	`supply0, supply1`
6	Strong drive	`strong0, strong1`
5	Pull drive	`pull0, pull1`
4	Large capacitive	`large`
3	Weak drive	`weak0, weak1`
2	Medium capacitive	`medium`
1	Small capacitive	`small`
0	High impedance	`highz0, highz1`

Verilog имеет:

- 4 силы сигнала;
- 3 емкостные силы;
- высокий импеданс.

Сила сигнала в цепи определяется динамически на основе силы самого сильного драйвера.

Емкостные силы задаются только для цепей, объявленных как trireg.

*Standard for Verilog® Hardware Description Language: IEEE Std 1364™-2005, 7.9
https://www.eg.bucknell.edu/~csci320/2016-fall/wp-content/uploads/2015/08/verilog-std-1364-2005.pdf

Описание блока assign (2)

Синтаксис описания блока `assign`:

```
assign <net_expr> = [drive strength] [delay] <logical expressions with signals or constants>
```

Правила использования блока `assign`:

- LHS - всегда wire скаляр, вектор или их конкатенация, и никогда - reg;

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

  assign y = ~x;

endmodule
```

```
module test;
  wire [3:0] y;
  wire [2:0] x1;
  wire       x2;

  assign y = {x1, x2};

endmodule
```

```
module test;
  reg  [3:0] y;
  wire [2:0] x1;
  wire       x2;

  assign y = {x1, x2};

endmodule
```

```
student@localhost:~/t1002817> iverilog ./test.v
./test.v:7: error: reg y; cannot be driven by primitives or continuous assignment.
1 error(s) during elaboration.
student@localhost:~/t1002817>
```

Описание блока assign (3)

Синтаксис описания блока `assign`:

```
assign <net_expr> = [drive strength] [delay] <logical expressions with signals or constants>
```

Правила использования блока `assign`:

- LHS - всегда wire скаляр, вектор или их конкатенация, и никогда - reg;
- RHS может содержать скалярные и векторные выражения, а также вызовы функций;

```
module test;
  wire [3:0] y;
  wire [2:0] x1;
  wire       x2;

  assign y = {x1, x2};

endmodule
```

```
module test;
  wire [2:0] y;
  wire [1:0] x1;
  wire x2;

  function concat;
    input a, b, c;
    begin
      concat = {a, b, c};
    end
  endfunction

  assign y = concat(x1[0], x1[1], x2);

endmodule
```

Описание блока assign (4)

Синтаксис описания блока `assign`:

```
assign <net_expr> = [drive strength] [delay] <logical expressions with signals or constants>
```

Правила использования блока `assign`:

- LHS - всегда wire скаляр, вектор или их конкатенация, и никогда - reg;
- RHS может содержать скалярные и векторные выражения, а также вызовы функций;
- как только любой элемент из RHS меняет значение, выражение LHS будет пересчитано;
- конструкция `assign` всегда активна;

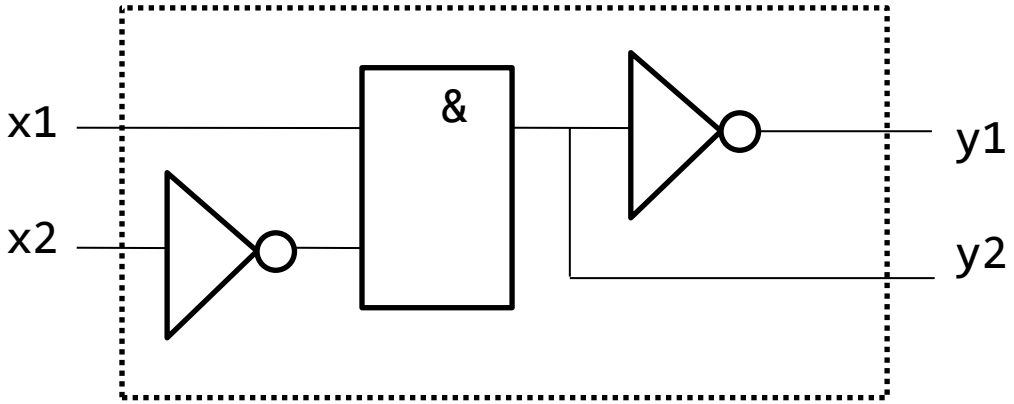
```
module device(x1, x2, y);
  input  x1, x2;
  output y;

  always @ (x1 or x2)
  begin
    if(x1 == 1 and x2 == 0)
      y <= 1'b0;
    else
      y <= 1'b1;
  end

endmodule
```

Операции, применимые в блоке always

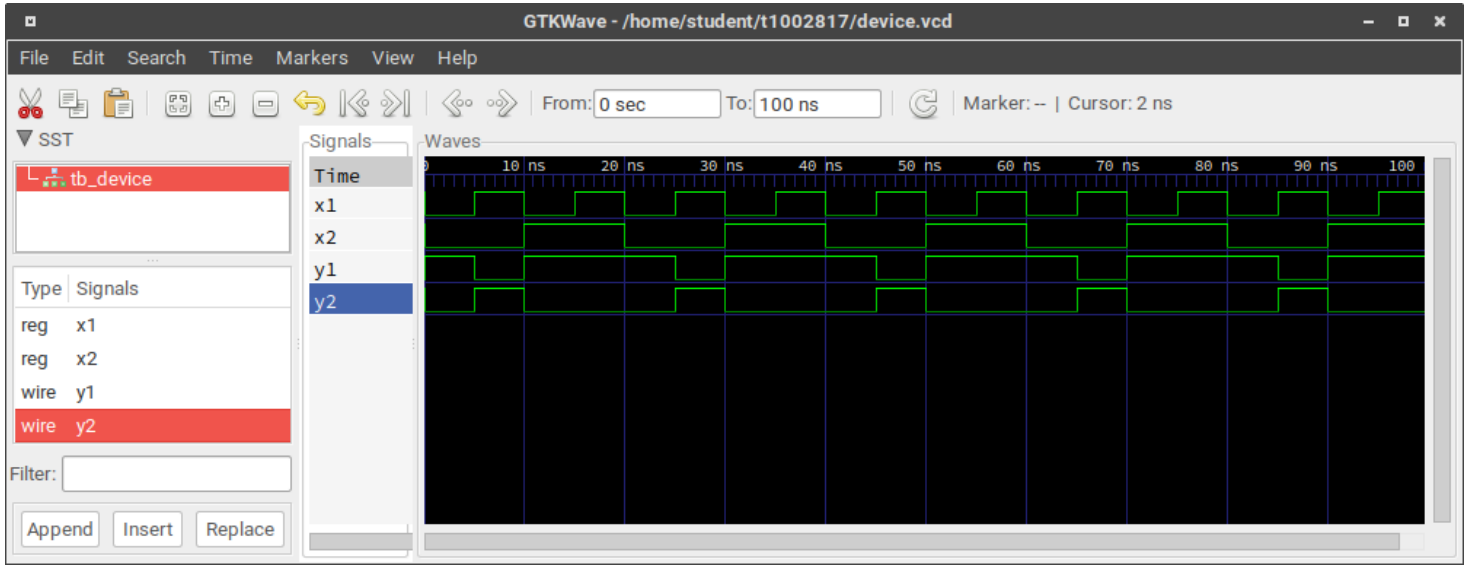
Оператор	Функция, назначение
( )	Группировка
~	NOT, НЕ, инверсия бит
&	AND, И, побитовое умножение
~&	NAND, И-НЕ, побитовое умножение с отрицанием
|	OR, ИЛИ, побитовое сложение
~|	NOR, ИЛИ-НЕ, побитовое сложение с отрицанием
^	XOR, ИСКЛЮЧАЮЩЕЕ ИЛИ
~^	XNOR, ИСКЛЮЧАЮЩЕЕ ИЛИ с отрицанием
+, -	сложение и вычитание
{}	оператор конкатенации
?:	тернарный оператор
[ ]	bit-select, оператор выбора битов
[:]	part-select, оператор выбора части вектора (срез)

```
module fulladder(input [3:0] a,
                 input [3:0] b,
                 input c_in,
                 output c_out,
                 output [3:0] sum);

  assign {c_out, sum} = a + b + c_in;

endmodule
```

Несколько блоков assign

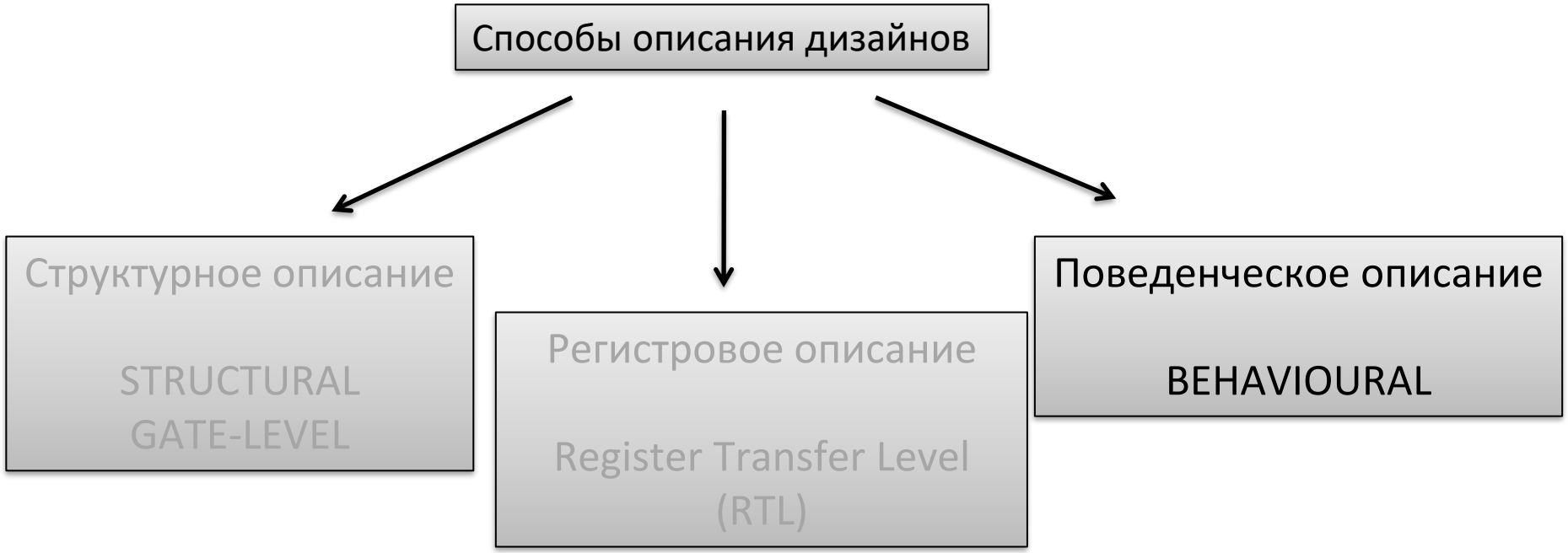



```
module device(x1, x2, y1, y2);
  input  x1, x2;
  output y1, y2;

  assign y1 = ~(x1 & ~x2);
  assign y2 = x1 & ~x2;

endmodule
```

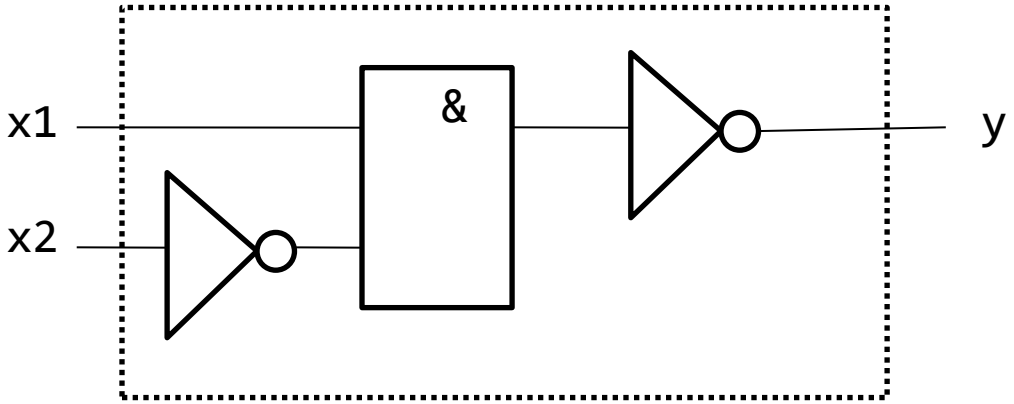



```
module device(x1, x2, y1, y2);
  input x1, x2;
  inout y1;
  output y2;

  assign y1 = ~(x1 & ~x2);
  assign y2 = ~y1;

endmodule
```

Способы описания дизайнов



Поведенческое описание



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

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

  always @ (x1 or x2)
  begin
    if(x1 == 1 && x2 == 0)
      y <= 1'b0;
    else
      y <= 1'b1;
  end

endmodule
```

Процедурный блок always (1)

Варианты синтаксиса описания блока `always`:

```
always @ (<event(s)>)
    <seq. statement>
```

```
always @ (<event(s)>) begin
    <seq. statement1>
    <seq. statement2>
    ...
end
```

```
always <time delay> <net expression>
```

```
always @ (x)
    if(x == 1'b1)
      y <= 1'b0;
    else
      y <= 1'b1;
```

```
always @ (x1 or x2)
  if(x1 == 1 && x2 == 0)
    y <= 1'b0;
  else
    y <= 1'b1;
```

```
always @(negedge clk) begin
  q <= d;
  nq <= ~d;
end
```

Процедурный блок always (2)

Варианты синтаксиса описания блока `always`:

```
always @ (<event(s)>)
    <seq. statement>
```

```
always @ (<event(s)>) begin
    <seq. statement1>
    <seq. statement2>
    ...
end
```

```
always <time delay> <net expression>
```

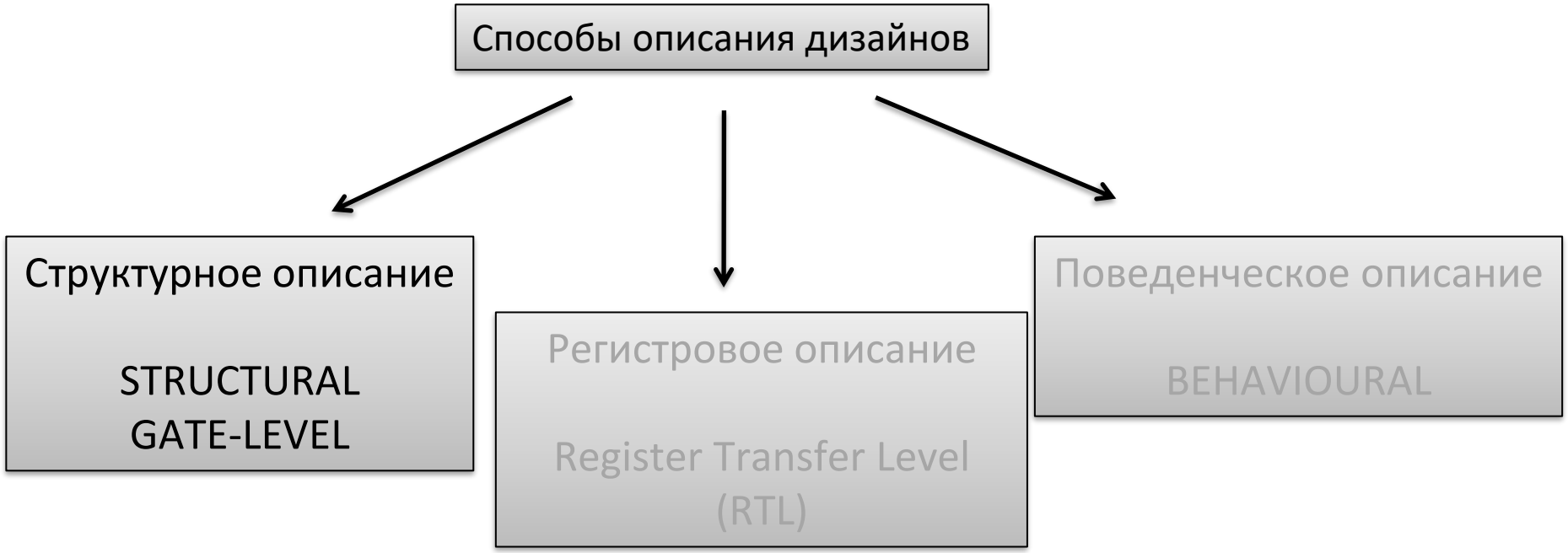
```
always #5 x = ~x;
```

Задание входных воздействий

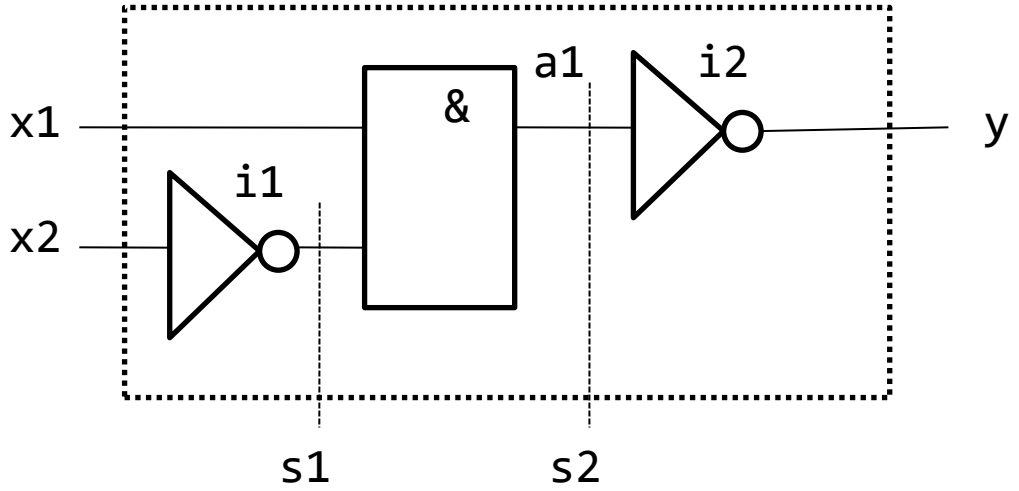
```
module tb_inverter;
    reg x;
    wire y;

    inverter dut(x, y);

    initial begin
            x = 0;
        #10 x = 1;
        #10 x = 0;
        #10 x = 1;
        #10 $finish;
    end

endmodule
```

```
module tb_inverter;
    reg x;
    wire y;

    inverter dut(x, y);

    initial begin
            x = 0;
        #40 $finish;
    end

    always #10 x = ~x;

endmodule
```

Способы описания дизайнов



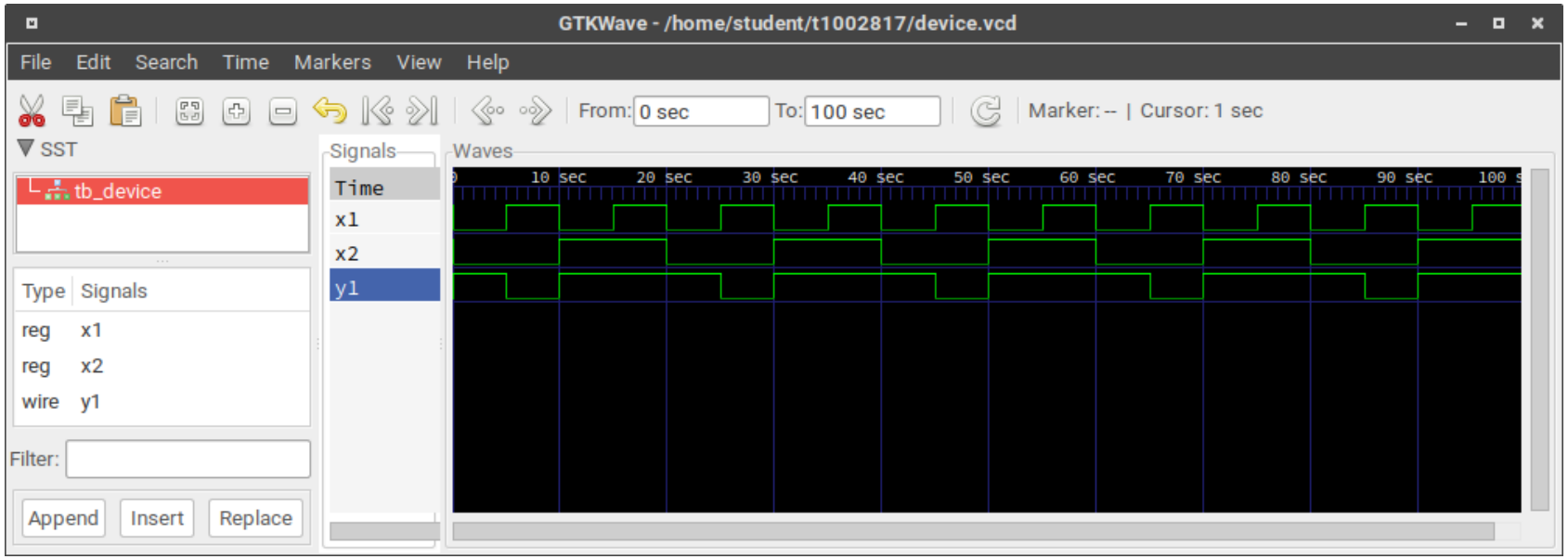
Структурное описание (1)




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

  wire s1, s2;

  inv  i1(x2, s1);
  and2 a1(x1, s1, s2);
  inv  i2(s2, y);

endmodule
```

Написание тестового окружения

```
module tb_device;
  reg x1, x2;
  wire y1;

  device dut(x1, x2, y1);

  initial begin
    $dumpfile("device.vcd");
    $dumpvars(1, x1, x2, y1);
    x1 = 0;
    x2 = 0;
    #100 $finish;
  end

  always #5 x1 = ~x1;
  always #10 x2 = ~x2;

endmodule
```



Задание единиц измерения времени (1)

Синтаксис конструкции `timescale для задания времени:

```
`timescale <time_unit>/<time_precision>
```

Примеры:

```
`timescale 1ns/1ns
`timescale 1ns/1ps
`timescale 10ns/1ns
`timescale 1us/10ns
```

Единицы измерения времени:

- s
- ms
- us
- ns
- ps
- fs

```
`timescale 1ns/1ns
module tb_device;
  reg x1, x2;
  wire y1;

  device dut(x1, x2, y1);

  initial begin
    $dumpfile("device.vcd");
    $dumpvars(1, x1, x2, y1);
    x1 = 0;
    x2 = 0;
    #100 $finish;
  end

  always #5 x1 = ~x1;
  always #10 x2 = ~x2;

endmodule
```