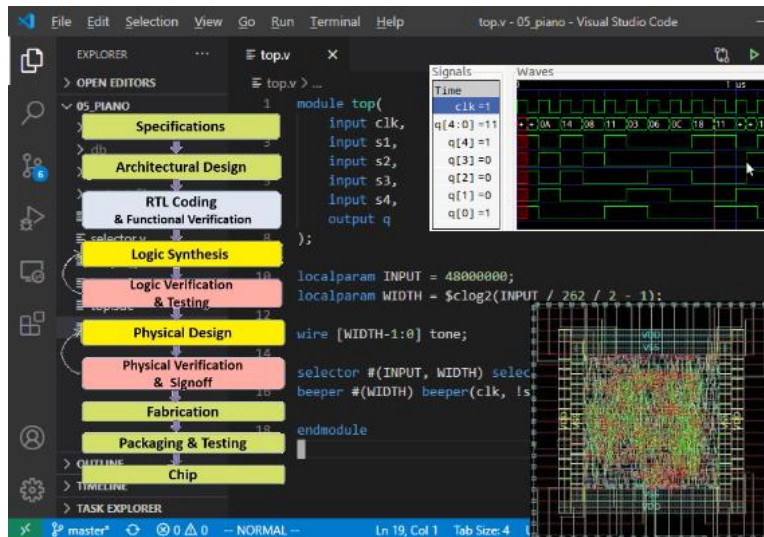




# Лингвистические средства проектирования

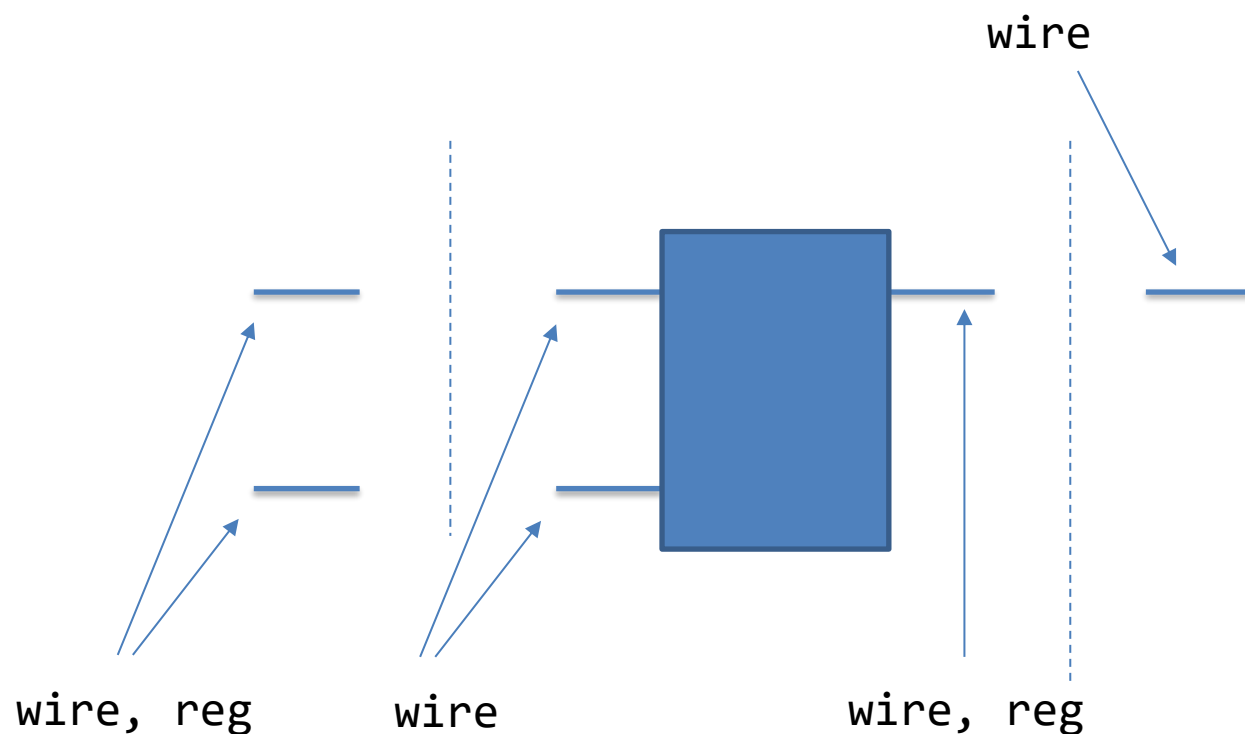
Лекция 2

**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
- 8'hFF
- 8'o326
- 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)

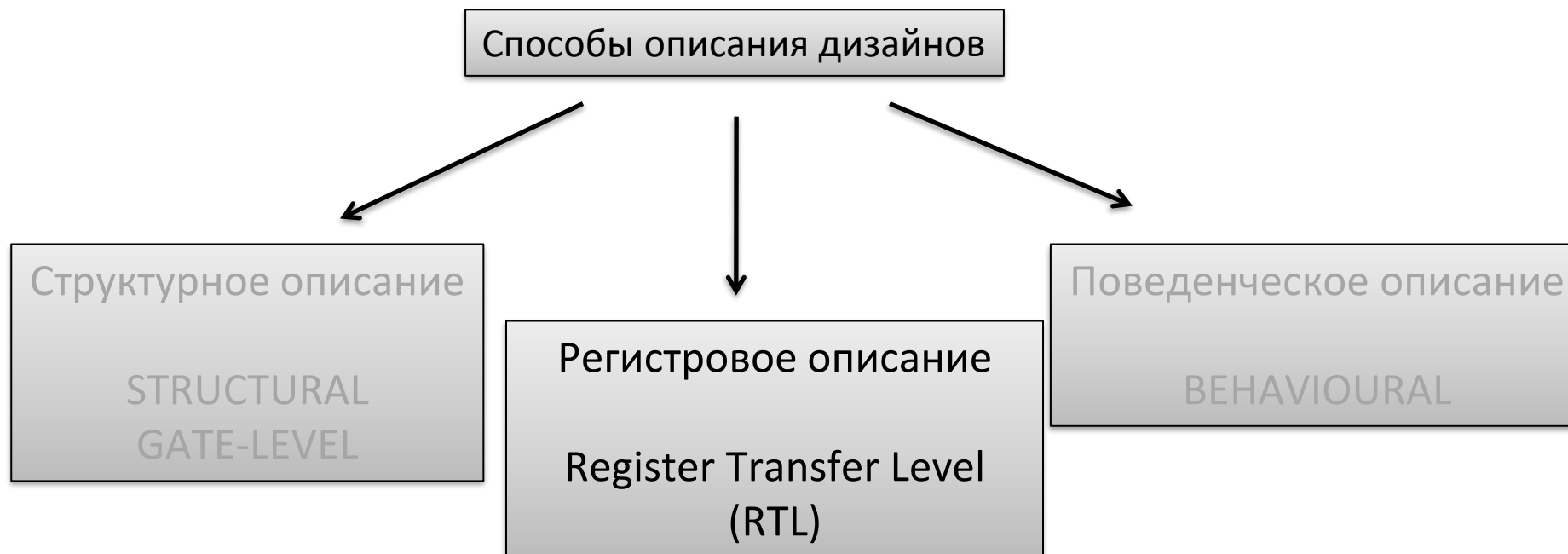
### Про wire:

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.

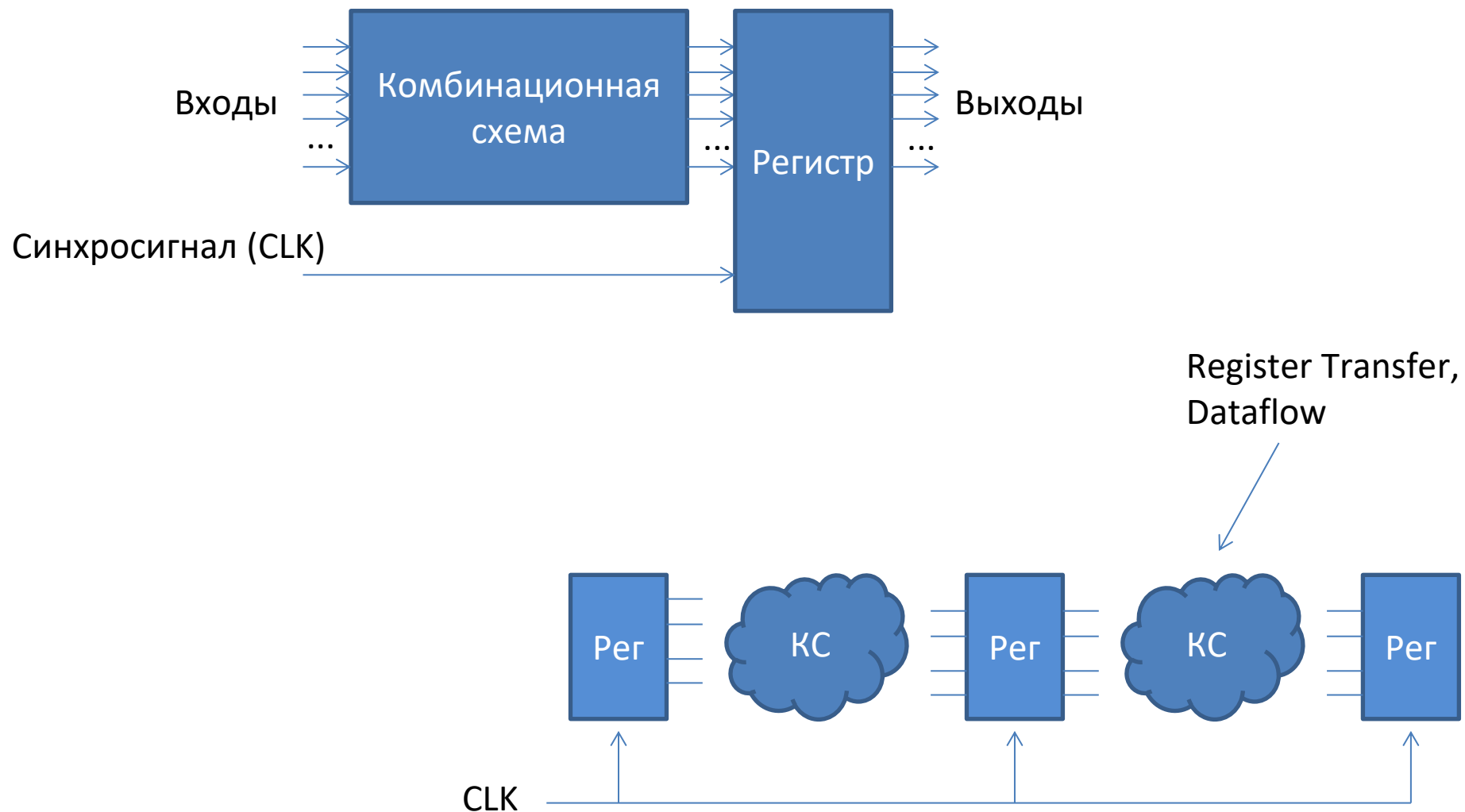
### Про reg:

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



# RTL: Register Transfer Level



## 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 `triereg`.

## 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;

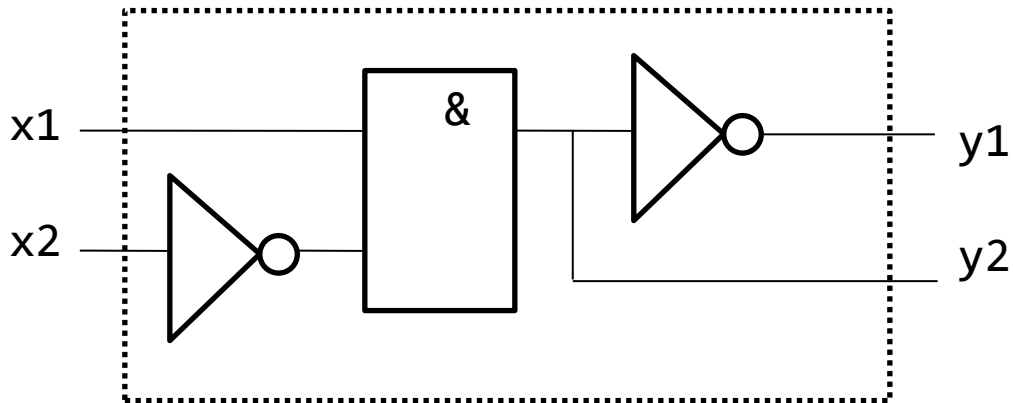
```
module device(x1, x2, y);  
    input      x1, x2;  
    output reg y;  
  
    always @ (x1 or x2)  
    begin  
        if(x1 == 1 and x2 == 0)  
            y <= 1'b0;  
        else  
            y <= 1'b1;  
        end  
  
endmodule
```

## Операции, применимые в блоке assign

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
```

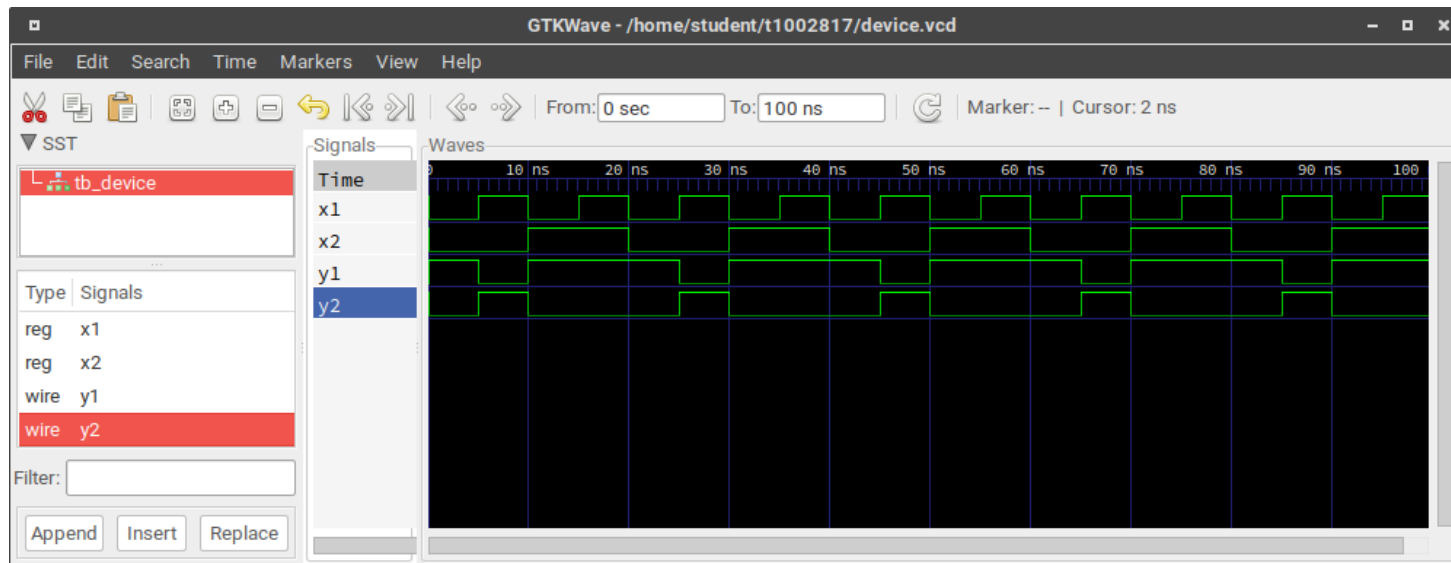
## Несколько блоков assign



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

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

```
endmodule
```

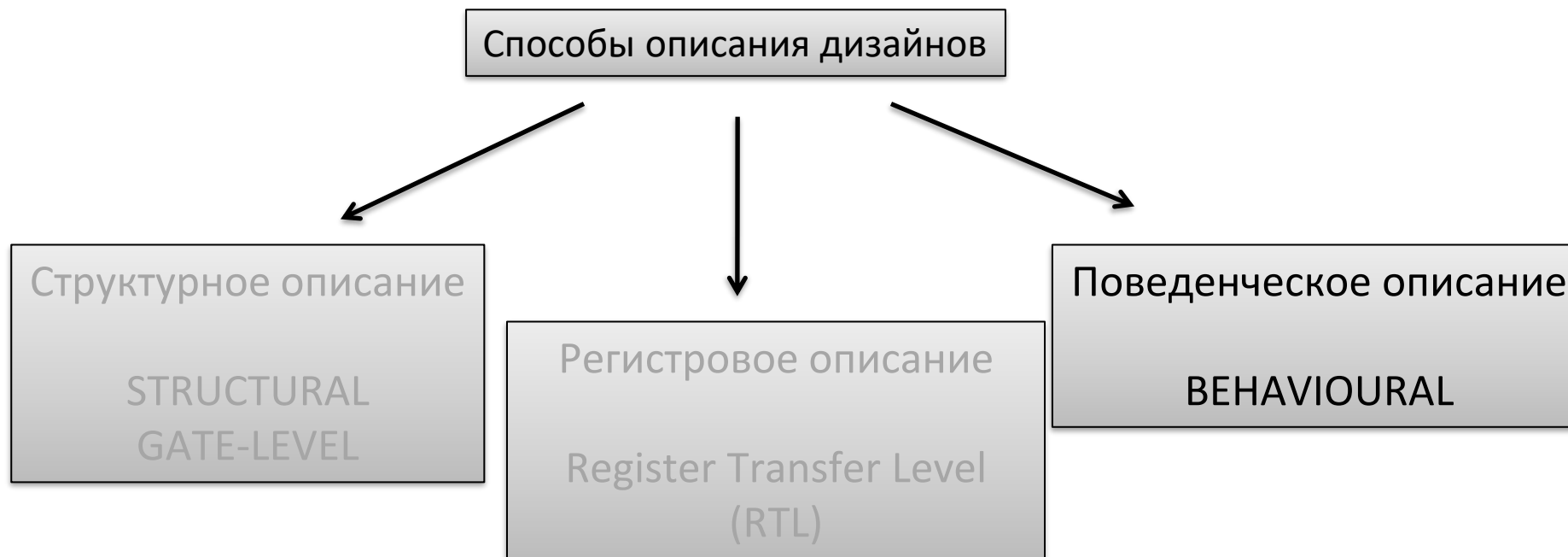


```
module device(x1, x2, y1, y2);  
  input  x1, x2;  
  inout y1;  
  output y2;
```

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

```
endmodule
```

## Способы описания дизайнов



## 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
```



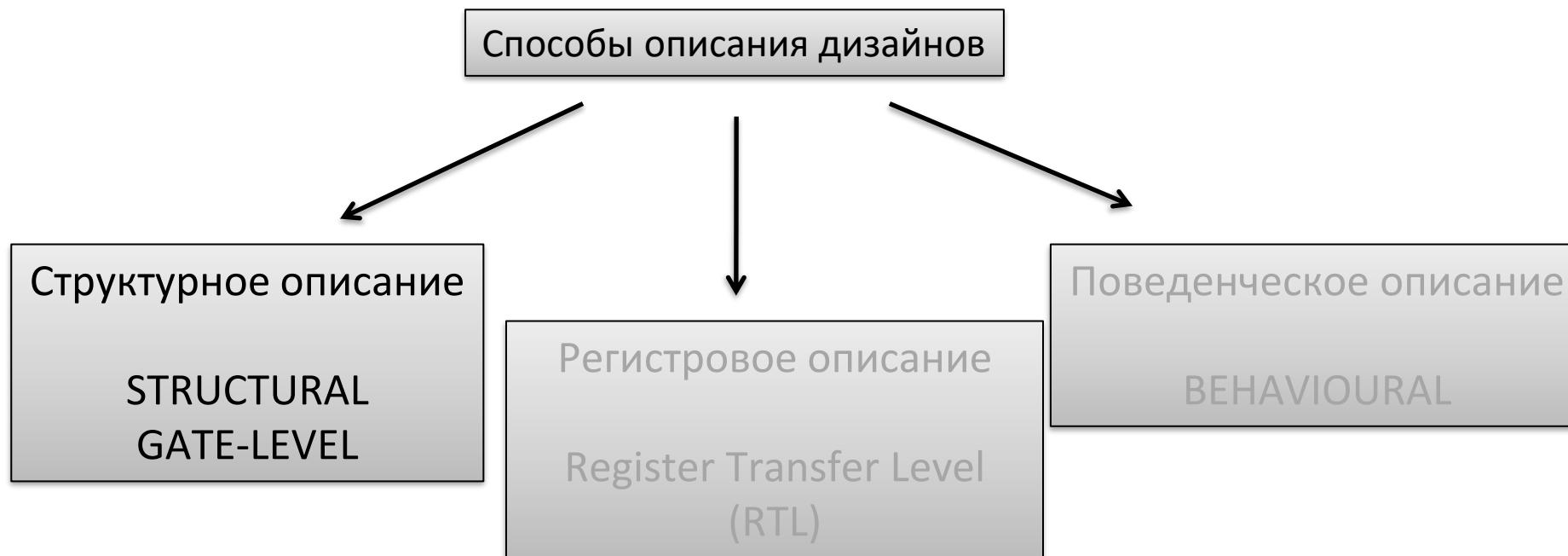
```
module tb_inverter;
  reg x;
  wire y;

  inverter dut(x, y);

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

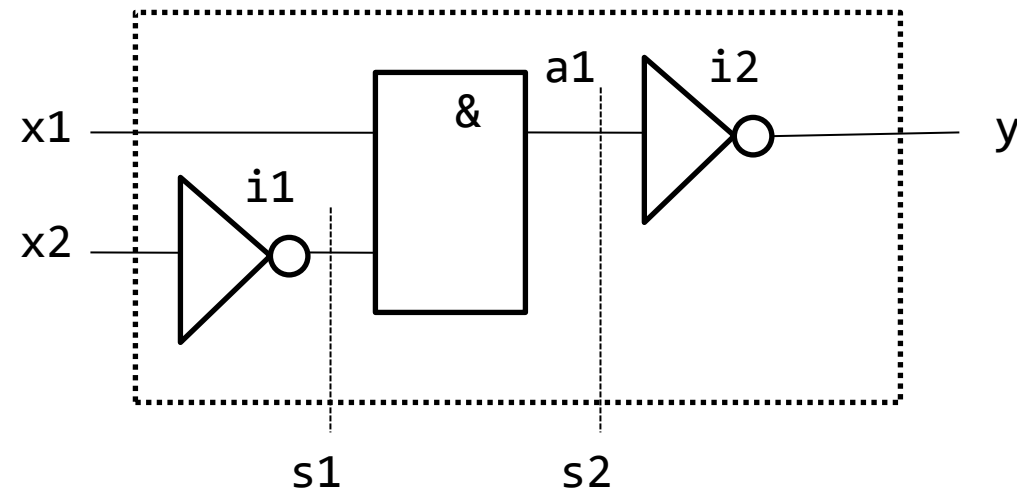
  always #10 x = ~x;
endmodule
```

## Способы описания дизайнов



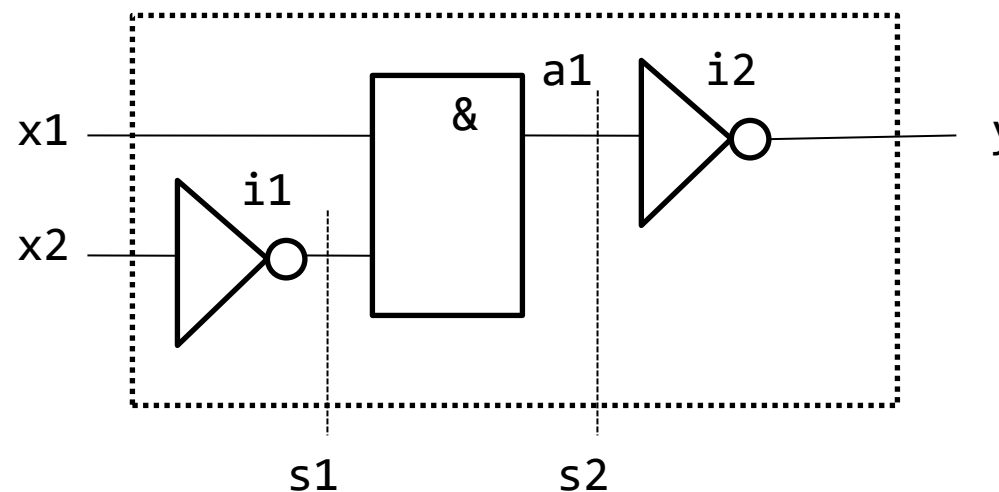


## Структурное описание на библиотечных элементах



```
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
```

## Структурное описание на вентиляном уровне



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

```
  wire s1, s2;
```

```
  not i1(s1, x2);
```

```
  and a1(s2, x1, s1);
```

```
  not i2(y, s2);
```



```
  not (s1, x2);
```

```
  and (s2, x1, s1);
```

```
  not (y, s2);
```

```
endmodule
```

## Структурное описание - финальный этап для логического проектирования

```
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
```

```
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
```

## Написание тестового окружения

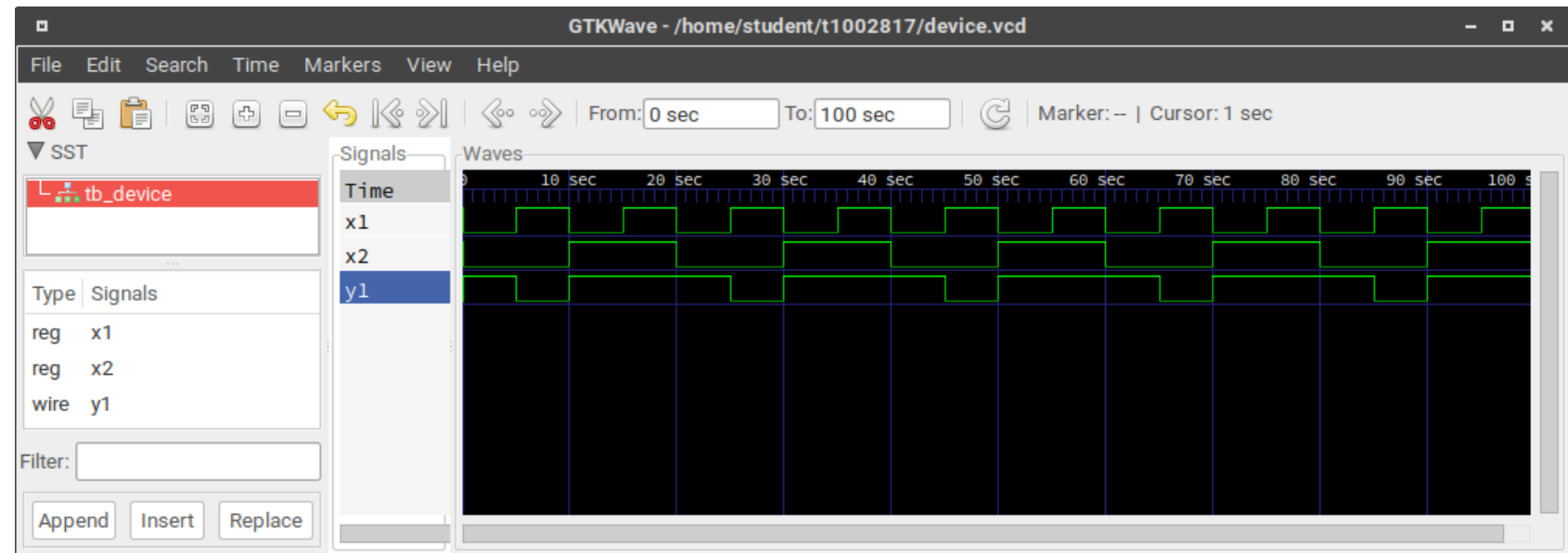
```
module tb_device;
    reg x1, x2;
    wire y1;

    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
```

## Задание единиц измерения времени (2)

```
`timescale 1ns/1ns
```

```
module test;  
  initial begin  
    #0  
    $display("Time now : %0t", $time);  
    #1  
    $display("Time now : %0t", $time);  
    #0.46  
    $display("Time now : %0t", $time);  
    #0.18  
    $display("Time now : %0t", $time);  
    #0.5  
    $display("Time now : %0t", $time);  
  end  
endmodule
```

```
● student@localhost:~/t1002817> vvp a.out  
Time now : 0  
Time now : 1  
Time now : 1  
Time now : 1  
Time now : 2
```

## Задание единиц измерения времени (3)

```
`timescale 1ns/1ps
```

```
module test;  
  initial begin  
    #0  
    $display("Time now : %0t", $time);  
    #1  
    $display("Time now : %0t", $time);  
    #0.46  
    $display("Time now : %0t", $time);  
    #0.18  
    $display("Time now : %0t", $time);  
    #0.5  
    $display("Time now : %0t", $time);  
  end  
endmodule
```

```
● student@localhost:~/t1002817> vvp a.out  
Time now : 0  
Time now : 1000  
Time now : 1000  
Time now : 2000  
Time now : 2000  
○ student@localhost:~/t1002817> █
```

## Задание единиц измерения времени (4)

```
`timescale 1ns/1ps
```

```
module test;  
  initial begin  
    #0  
    $display("Time now : %0t", $realtime);  
    #1  
    $display("Time now : %0t", $realtime);  
    #0.46  
    $display("Time now : %0t", $realtime);  
    #0.18  
    $display("Time now : %0t", $realtime);  
    #0.5  
    $display("Time now : %0t", $realtime);  
  end  
endmodule
```

```
● student@localhost:~/t1002817> vvp a.out  
Time now : 0  
Time now : 1000  
Time now : 1460  
Time now : 1640  
Time now : 2140  
○ student@localhost:~/t1002817> █
```



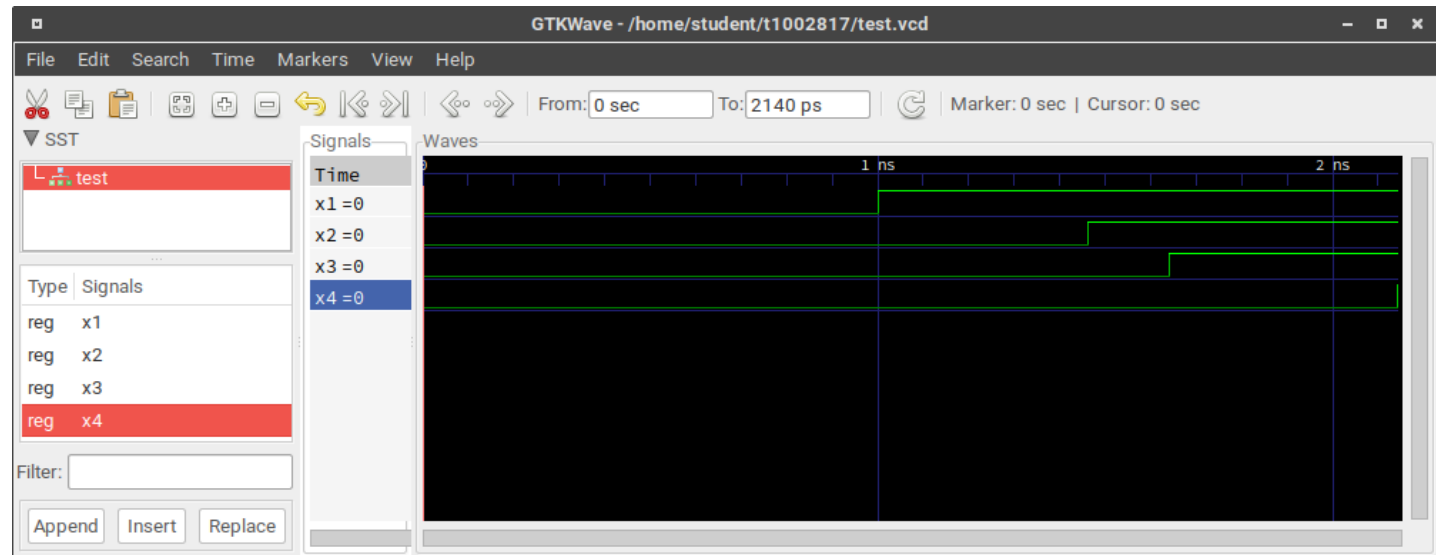
## Задание единиц измерения времени (5)

```
`timescale 1ns/1ps
module test;
  reg x1, x2, x3, x4;

  initial begin
    x1 = 0; x2 = 0; x3 = 0; x4 = 0;
    $dumpfile("test.vcd");
    $dumpvars(1, test);
    #0
    $display("Time now : %0t", $realtime);
    #1
    $display("Time now : %0t", $realtime);
    x1 = 1;
    #0.46
    $display("Time now : %0t", $realtime);
    x2 = 1;
    #0.18
    $display("Time now : %0t", $realtime);
    x3 = 1;
    #0.5
    $display("Time now : %0t", $realtime);
    x4 = 1;

  end
endmodule
```

```
● student@localhost:~/t1002817> vvp a.out
VCD info: dumpfile test.vcd opened for output.
Time now : 0
Time now : 1000
Time now : 1460
Time now : 1640
Time now : 2140
○ student@localhost:~/t1002817>
```



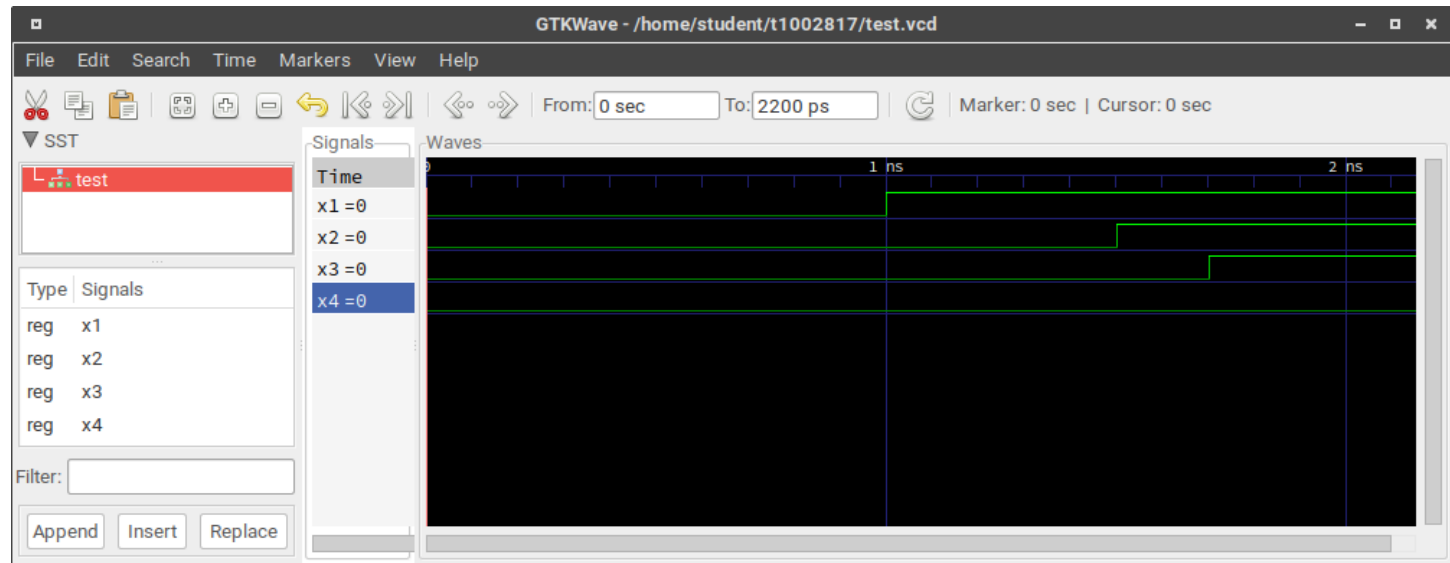
## Задание единиц измерения времени (6)

```
`timescale 1ns/100ps
module test;
  reg x1, x2, x3, x4;

  initial begin
    x1 = 0; x2 = 0; x3 = 0; x4 = 0;
    $dumpfile("test.vcd");
    $dumpvars(1, test);
    #0
    $display("Time now : %0t", $realtime);
    #1
    $display("Time now : %0t", $realtime);
    x1 = 1;
    #0.46
    $display("Time now : %0t", $realtime);
    x2 = 1;
    #0.18
    $display("Time now : %0t", $realtime);
    x3 = 1;
    #0.5
    $display("Time now : %0t", $realtime);
    x4 = 1;

  end
endmodule
```

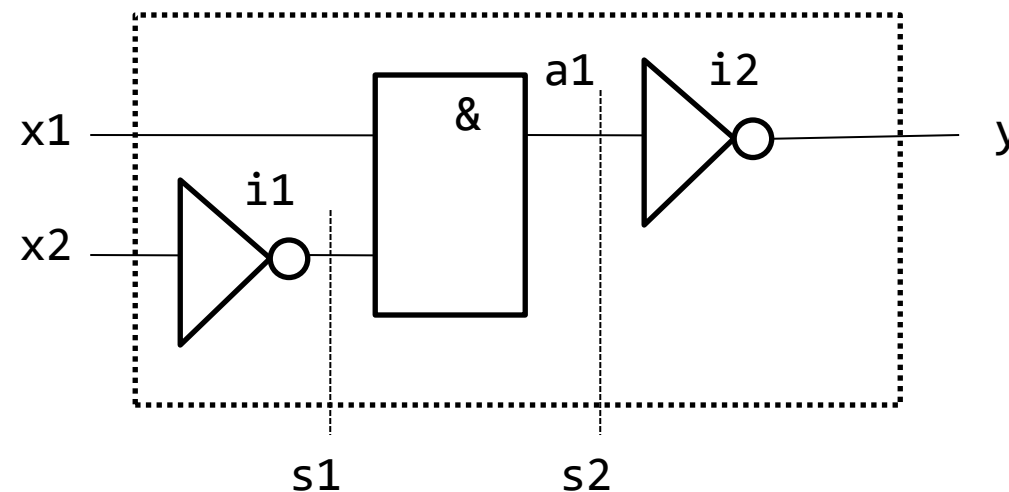
- student@localhost:~/t1002817> vvp a.out  
VCD info: dumpfile test.vcd opened for output.  
Time now : 0  
Time now : 10  
Time now : 15  
Time now : 17  
Time now : 22
- student@localhost:~/t1002817> □



## Задержка в переключении элементов

```
`timescale 1ns/1ns
module inverter (x, y);
  input x;
  output y;

  assign #2 y = ~x;
endmodule
```



```
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);
```



```
  inv  #(3) i1(x2, s1);
  and2 #(4) a1(x1, s1, s2);
  inv  #(2) i2(s2, y);
```

```
endmodule
```

## Формат описания задержек

```
(DELAYFILE
(SDFVERSION "3.0")
(DESIGN "BIGCHIP")
(DATE "Fri Jul 8 09:31:42 2022")
(TIMESCALE 100 ps)
(CELL
(CELLTYPE "AND2")
(INSTANCE top/b/d)
(DELAY
(ABSOLUTE
(IOPATH a y (1.5:2.5:3.4) (2.5:3.6:4.7))
(IOPATH b y (1.4:2.3:3.2) (2.3:3.4:4.3))
)
)
)
)
(CELL
(CELLTYPE "DFF")
(INSTANCE top/b/c)
(DELAY
(ABSOLUTE
(IOPATH (posedge clk) q (2:3:4) (5:6:7))
(PORT clr (2:3:4) (5:6:7))
)
)
)
)
```

```
(CELL
(CELLTYPE "sky130_fd_sc_hd__or2b_1")
(INSTANCE _104_)
(DELAY
(ABSOLUTE
(IOPATH A X (0.090:0.091:0.091) (0.222:0.222:0.222))
(IOPATH B_N X (0.167:0.167:0.167) (0.247:0.247:0.247))
)
)
)
(CELL
(CELLTYPE "sky130_fd_sc_hd__or2b_1")
(INSTANCE _110_)
(DELAY
(ABSOLUTE
(IOPATH A X (0.086:0.087:0.088) (0.218:0.218:0.218))
(IOPATH B_N X (0.165:0.165:0.165) (0.245:0.245:0.245))
)
)
)
```