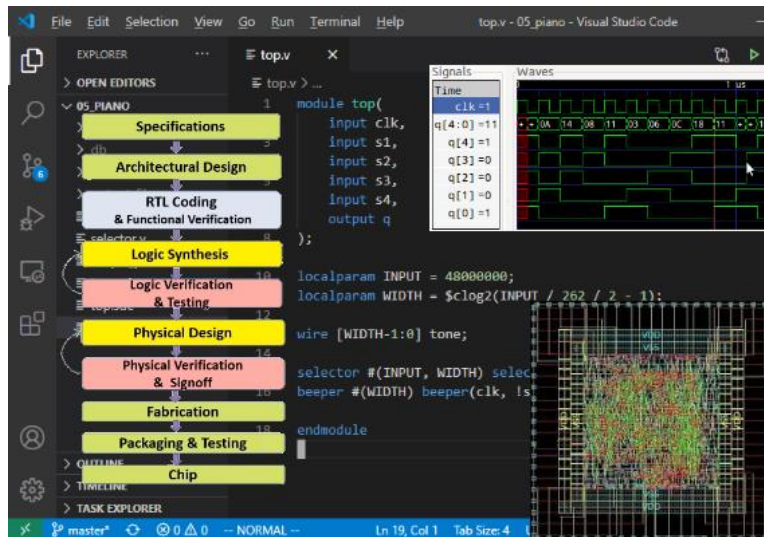




Лингвистические средства проектирования

Лекция 2

Последовательные операторы в поведенческом описании



Последовательные инструкции в языке Verilog

Последовательные инструкции встречаются в:

- процедурных блоках always

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

- блоках initial

```
initial begin
  x = 0;
  #10
  x1 = 1;
  #10 $finish;
end
```

- функциях

```
function do_job(input x);
begin
  if(x == 1)
    do_job = 1'b0;
  else
    do_job = 1'b1;
end
endfunction
```

- задачах

```
task init;
begin
  clk = 0;
  r = 0;
  d = 0;
end
endtask
```

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;
```

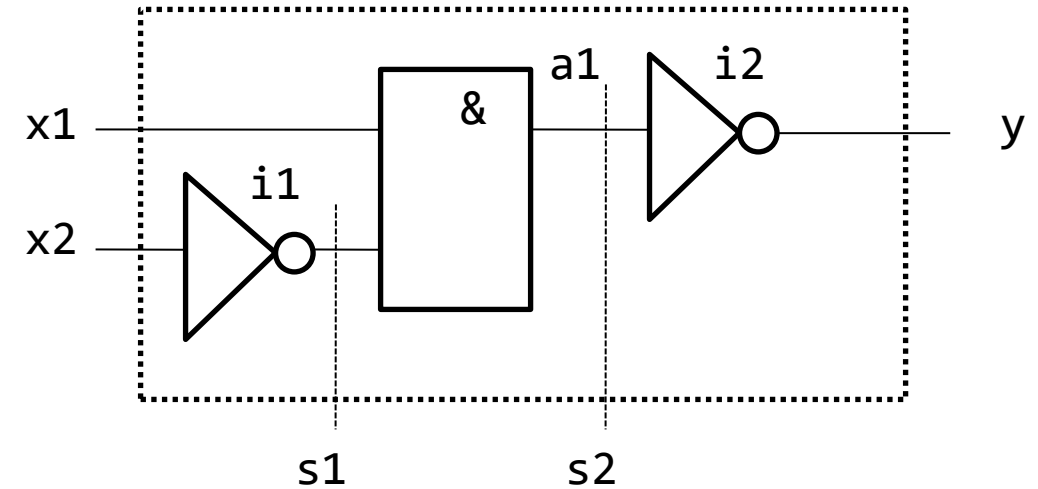
```
always @(negedge clk) begin  
  q <= d;  
  nq <= ~d;  
end
```

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

Варианты записи блока always

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

Verilog-1995



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

Verilog-2001

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

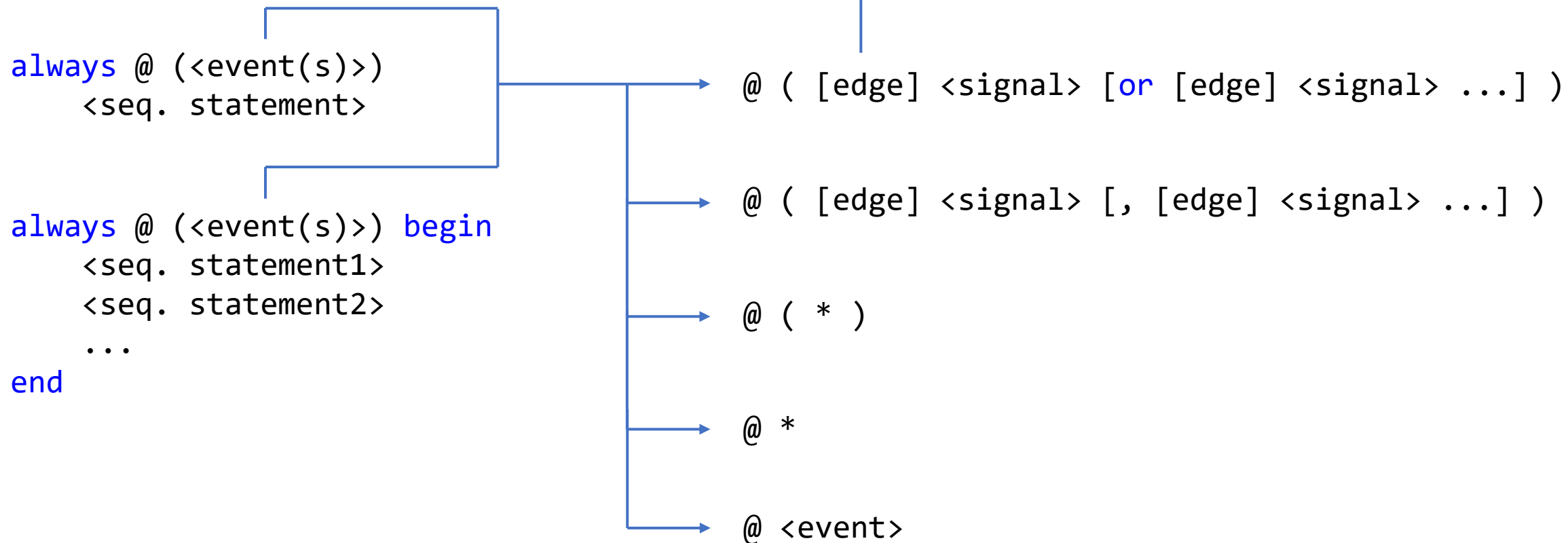
Verilog-2001

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

Verilog-2001

Формальный синтаксис списков чувствительности

Варианты синтаксиса описания блока `always`:



Блок начальной инициализации initial (2)

Варианты синтаксиса описания блока `initial`:

```
initial
  <seq. statement>
```

```
initial begin
  <seq. statement1>
  <seq. statement2>
  ...
end
```

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

  inv dut(x, y);

  initial begin
    $dumpfile("inv.vcd");
    $dumpvars(1, tb_inv);
  end

  initial
    x = 0;

  initial
    #100 $finish;

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

Функции в языке Verilog

Синтаксис описания функций:

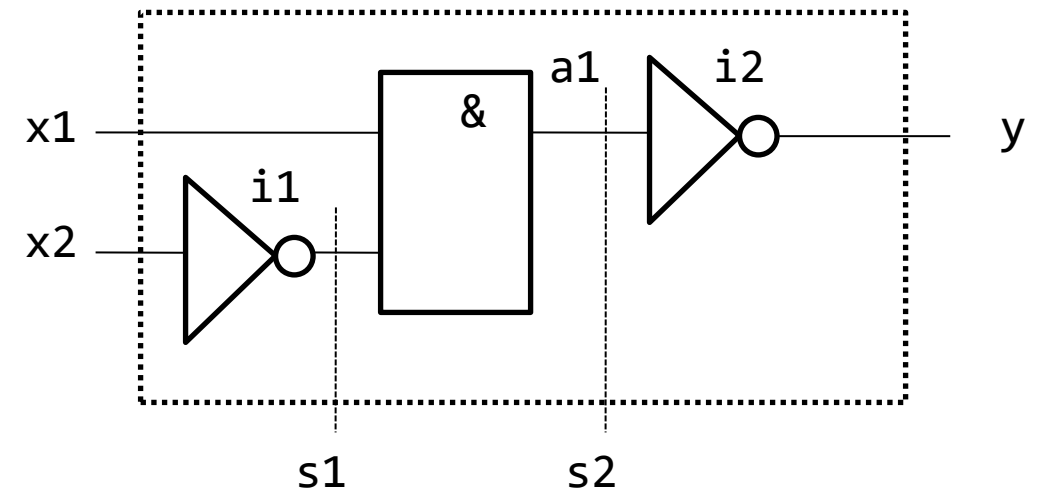
```
function [automatic] [<return type>] <function name> ([<port> [, <port> ...]]);  
  [<declarations>]  
begin  
  <seq. statement>  
  [<seq. statement>]  
  ...  
end  
endfunction
```

```
function [automatic] [<return type>] <function name>;  
  [<port> [, <port> ...]]  
  [<declarations>]  
begin  
  <seq. statement>  
  [<seq. statement>]  
  ...  
end  
endfunction
```

Правила использования функций

- Функция не может содержать конструкции, явно или неявно работающие со временем (time-controlled statements): #, @, wait, posedge, negedge.
- Из функции нельзя запустить задачу (task), поскольку задачи могут работать с конструкциями, влияющими на модельное время.
- Функция должна иметь как минимум один аргумент (одно входное значение).
- Значение аргументов – всегда input, функция не может принимать аргумент с направлением output или inout.
- Функция не может иметь неблокирующие конструкции присваивания.

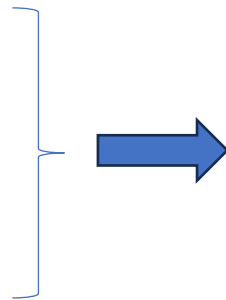
Пример описания функции в языке Verilog



```
`timescale 1ns/1ns
module device(x1, x2, y);
  input x1, x2;
  output reg y;

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

endmodule
```



```
function do_job(input x1, x2);
  begin
    if(x1 == 1 && x2 == 0)
      do_job = 1'b0;
    else
      do_job = 1'b1;
  end
endfunction

always @ (x1 or x2)
  y<= do_job(x1, x2);
```

Пример описания статической функции в языке Verilog

```
function do_job(input x1, x2);
  reg temp;
  begin
    $display("Before: temp=%d (at time=%0t)", temp, $time);
    temp = x1;
    $display("After : temp=%d (at time=%0t)", temp, $time);

    if(x1 == 1 && x2 == 0)
      do_job = 1'b0;
    else
      do_job = 1'b1;
    end
  endfunction

always @*
  y<= do_job(x1, x2);
```

```
Before: temp=x (at time=0)
After : temp=0 (at time=0)
Before: temp=0 (at time=5)
After : temp=1 (at time=5)
Before: temp=1 (at time=10)
After : temp=0 (at time=10)
Before: temp=0 (at time=15)
After : temp=1 (at time=15)
Before: temp=1 (at time=20)
After : temp=0 (at time=20)
Before: temp=0 (at time=25)
After : temp=1 (at time=25)
Before: temp=1 (at time=30)
After : temp=0 (at time=30)
Before: temp=0 (at time=35)
After : temp=1 (at time=35)
B
```

Пример описания автоматической функции в языке Verilog

```
function automatic do_job(input x1, x2);
  reg temp;
  begin
    $display("Before: temp=%d (at time=%0t)", temp, $time);
    temp = x1;
    $display("After : temp=%d (at time=%0t)", temp, $time);

    if(x1 == 1 && x2 == 0)
      do_job = 1'b0;
    else
      do_job = 1'b1;
    end
  endfunction

always @*
  y<= do_job(x1, x2);
```

```
Before: temp=x (at time=0)
After : temp=0 (at time=0)
Before: temp=x (at time=5)
After : temp=1 (at time=5)
Before: temp=x (at time=10)
After : temp=0 (at time=10)
Before: temp=x (at time=15)
After : temp=1 (at time=15)
Before: temp=x (at time=20)
After : temp=0 (at time=20)
Before: temp=x (at time=25)
After : temp=1 (at time=25)
Before: temp=x (at time=30)
After : temp=0 (at time=30)
Before: temp=x (at time=35)
After : temp=1 (at time=35)
```

Задачи в языке Verilog

Синтаксис описания задач:

```
task [automatic] <task name> [(input <ports>), [output <ports>] [inout <ports>]);  
  [<declarations>]  
  begin  
    <seq. statement>  
    [<seq. statement>]  
    ...  
  end  
endtask
```

```
task [automatic] <task name>;  
  [input <ports>;]  
  [output <ports>;]  
  [inout <ports>;]  
  [<declarations>]  
  begin  
    <seq. statement>  
    [<seq. statement>]  
    ...  
  end  
endtask
```

Сравнение функций и задач

Функция	Задача
Не может иметь конструкции, связанные с временем	Может иметь конструкции, связанные с временем
Не может стартовать задачу	Может стартовать функцию или другую задачу
Должна иметь как минимум один входной аргумент	Может не иметь аргументов
Не может иметь выходных аргументов	Может иметь аргументы любого направления
Может вернуть единственное значение	Не возвращает значение напрямую (в синтаксисе функций), но может вернуть значения через выходные аргументы

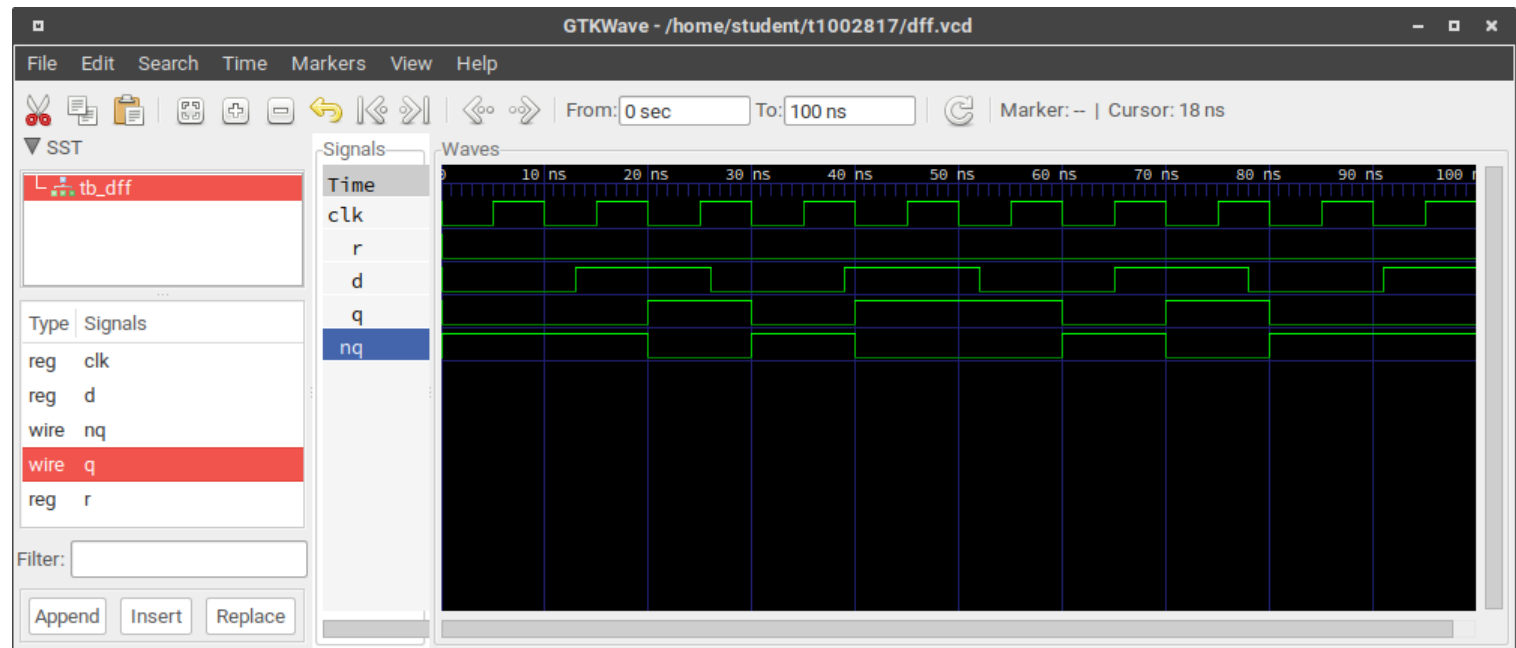
Пример использования задач (1)

```
`timescale 1ns/1ns
module tb_dff;
    reg clk, d, r;
    wire q, nq;

    dff dut(clk, d, r, q, nq);

    initial begin
        $dumpfile("dff.vcd");
        $dumpvars(1, tb_dff);
        clk = 0;
        r = 0;
        d = 0;
        #100 $finish;
    end

    always #5 clk = ~clk;
    always #13 d = ~d;
endmodule
```



Пример использования задач (2)

```
`timescale 1ns/1ns
module tb_dff;
  reg clk, d, r;
  wire q, nq;

  dff dut(clk, d, r, q, nq);

  initial begin
    $dumpfile("dff.vcd");
    $dumpvars(1, tb_dff);
    clk = 0;
    r = 0;
    d = 0;
    #100 $finish;
  end

  always #5 clk = ~clk;
  always #13 d = ~d;
endmodule
```

```
task init;
begin
  clk = 0;
  r = 0;
  d = 0;
end
endtask
```

```
initial begin
  $dumpfile("dff.vcd");
  $dumpvars(1, tb_dff);
  init;
  #100 $finish;
end
```

Пример использования задач (3)

```
`timescale 1ns/1ns
module tb_dff;
  reg clk, d, r;
  wire q, nq;

  dff dut(clk, d, r, q, nq);

  initial begin
    $dumpfile("dff.vcd");
    $dumpvars(1, tb_dff);
    clk = 0;
    r = 0;
    d = 0;
    #100 $finish;
  end

  always #5 clk = ~clk;
  always #13 d = ~d;
endmodule
```

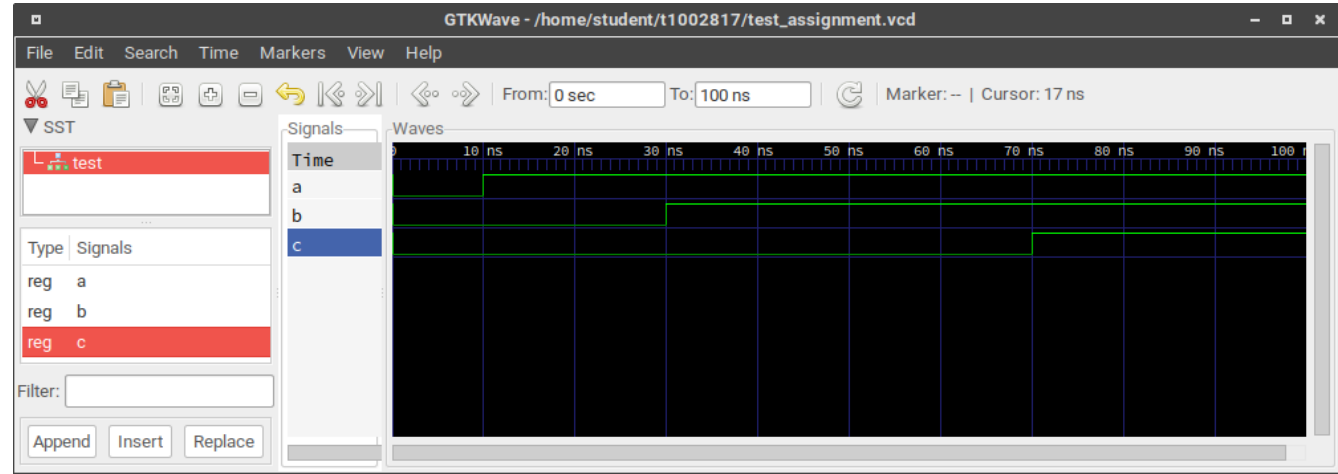


```
task handle_reset;
begin
  r = 0;
  #23
  r = 1;
  #1
  r = 0;
end
endtask
```

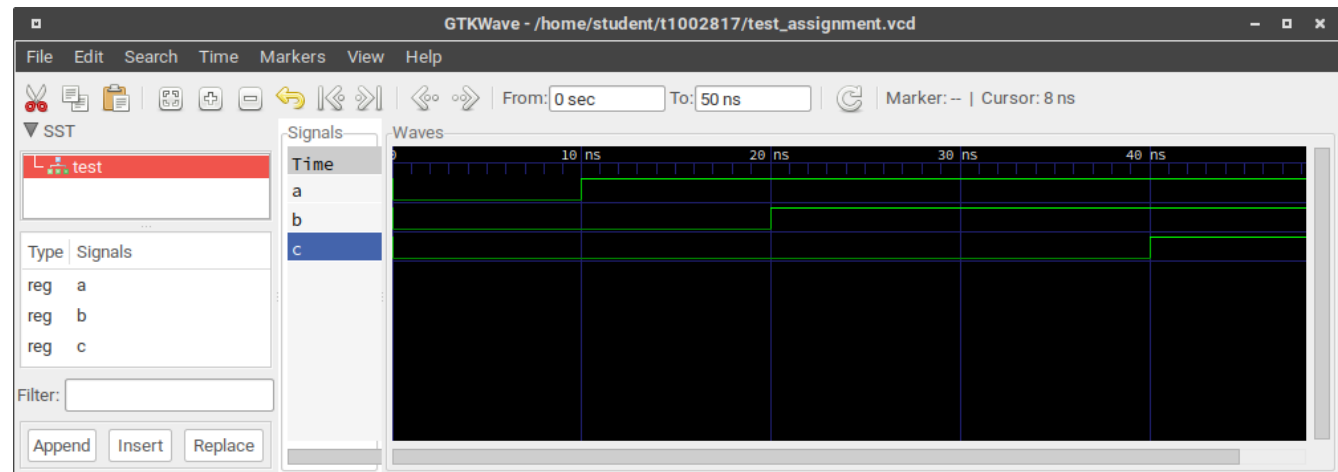
```
initial begin
  $dumpfile("dff.vcd");
  $dumpvars(1, tb_dff);
  init;
  handle_reset;
  #100 $finish;
end
```


Последовательные операторы: блокирующий и нет оператор присваивания

```
a = 0;  
b = 0;  
c = 0;  
a = #10 1'b1;  
b = #20 1'b1;  
c = #40 1'b1;  
#30 $finish;
```



```
a = 0;  
b = 0;  
c = 0;  
a <= #10 1'b1;  
b <= #20 1'b1;  
c <= #40 1'b1;  
#50 $finish;
```



Последовательные операторы: if-else

```
module dff (clk, d, r, q, nq);
  input clk, d, r;
  output reg q, nq;

  always @(negedge clk or posedge r) begin
    if (r == 1) begin
      q <= 1;
      nq <= 0;
    end
    else begin
      q <= d;
      nq <= ~d;
    end;
  end
endmodule
```

```
module mux(A1, A2, x1, x2, x3, x4, y);
  input A1, A2, x1, x2, x3, x4;
  output reg y;

  always @(*) begin
    if (A1 == 0 && A2 == 0)
      y <= x1;
    else if (A1 == 0 && A2 == 1)
      y <= x2;
    ...
  end
endmodule
```

Последовательные операторы: case, casex, casez

```
module mux4to1(a, b, c, d, sel, out);

    input      [3:0] a, b, c, d;
    input      [1:0] sel;
    output reg [3:0] out;

    always@(a or b or c or d or sel) begin
        case (sel)
            2'b00 : out <= a;
            2'b01 : out <= b;
            2'b10 : out <= c;
            2'b11 : out <= d;
        endcase
    end
endmodule
```

```
module mux(A1, A2, x1, x2, x3, x4, y);
    input A1, A2, x1, x2, x3, x4;
    output reg y;

    always @(*) begin
        case ({A1, A2})
            2'b00: y <= x1;
            2'b01: y <= x2;
            2'b10: y <= x3;
            2'b11: y <= x4;
        endcase
    end
endmodule

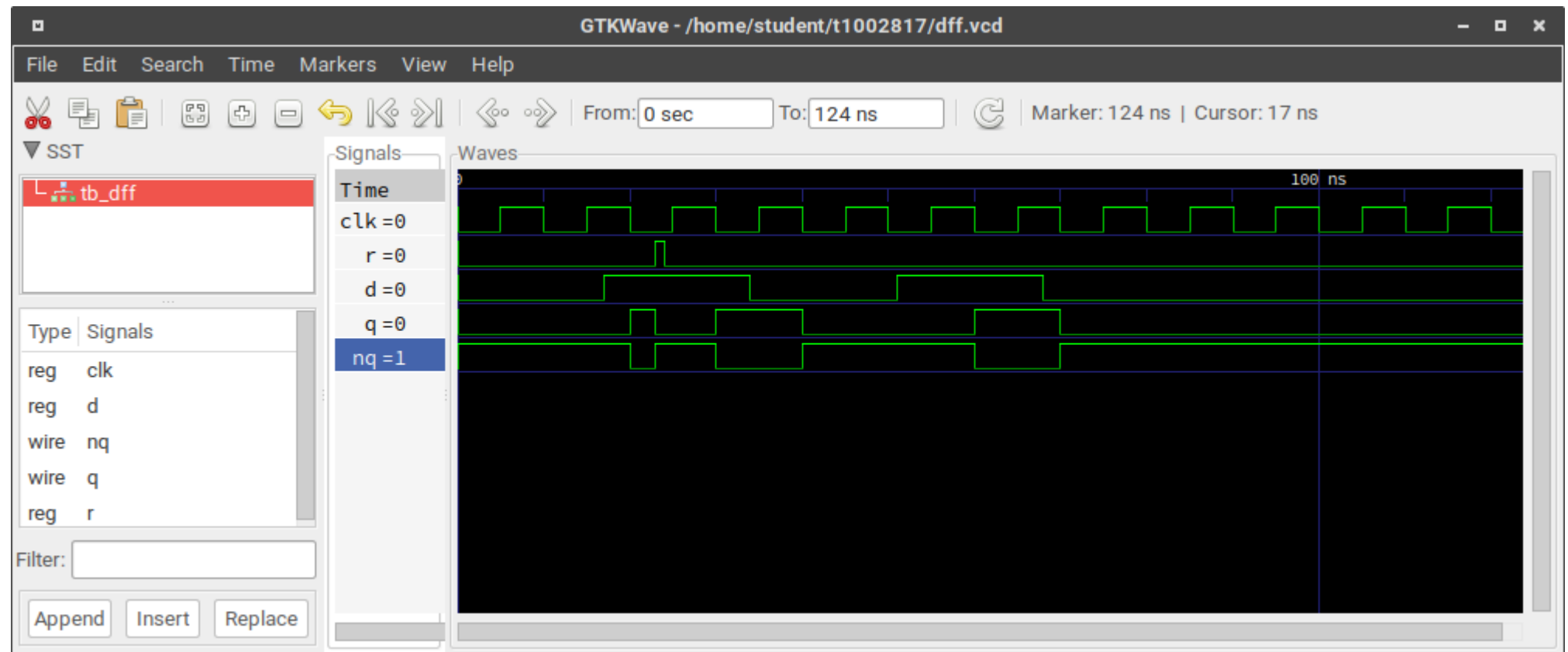
always @(*) begin
    case ({A1, A2})
        0: y <= x1;
        1: y <= x2;
        2: y <= x3;
        3: y <= x4;
    endcase
end

always @(*) begin
    case ({E, A1, A2})
        3'b0?? : y <= 0;
        3'b100 : y <= x1;
        3'b101 : y <= x2;
        3'b110 : y <= x3;
        3'b111 : y <= x4;
    endcase
end
```

Последовательные операторы: forever и repeat

```
initial
begin
  forever
    #5 clk = ~clk;
end
```

```
initial begin
  repeat (4)
    #17 d = ~d;
end
```



Системные функции для вывода сообщений (1)

- `$display`
Выполняется однократно, немедленно
- `$write`
Выполняется однократно, немедленно, по действию аналогичен `$display`, но не добавляет автоматически перевод на новую строку
- `$strobe`
Выполняется однократно в самом конце текущего шага моделирования
- `$monitor`
Выполняется постоянно до конца моделирования, один раз в самом конце каждого шага моделирования

```
initial begin
    $monitor("Monitor : x=%d, y=%d", x, y);
    #0
    x = 0;
    #10
    $display("Display1: x=%d, y=%d", x, y);
    $strobe ("Strobe : x=%d, y=%d", x, y);
    x = 1;
    $display("Display2: x=%d, y=%d", x, y);
    #10
    x = 0;
    #10
    x = 1;
    #10 $finish;
end
```

Системные функции для вывода сообщений (2)

```
initial begin
```

```
  $monitor("Monitor : x=%d, y=%d", x, y);
```

```
  #0
```

```
  x = 0;
```

```
  #10
```

```
  $display("Display1: x=%d, y=%d", x, y);
```

```
  $strobe ("Strobe : x=%d, y=%d", x, y);
```

```
  x = 1;
```

```
  $display("Display2: x=%d, y=%d", x, y);
```

```
  #10
```

```
  x = 0;
```

```
  #10
```

```
  x = 1;
```

```
  #10 $finish;
```

```
end
```

```
● student@localhost:~/t1002817> iverilog -o inv.o ./inv.v ./tb_inv.v
```

```
● student@localhost:~/t1002817> vvp ./inv.o
```

```
Monitor : x=0, y=1
```

```
Display1: x=0, y=1
```

```
Display2: x=1, y=1
```

```
Strobe : x=1, y=0
```

```
Monitor : x=1, y=0
```

```
Monitor : x=0, y=1
```

```
Monitor : x=1, y=0
```

```
○ student@localhost:~/t1002817> █
```

Сохранение результатов моделирования в формат VCD

- `$dumpfile(<имя файла>);`

Задаёт имя файла для сохранения результатов моделирования.

- `$dumpvars(<уровень>, <<сигнал>[, <сигнал>, ...]>);`

`$dumpvars(<уровень>, <<модуль>[, <модуль>, ...]>);` Уровень=0

`$dumpvars;`

Задаёт перечень сигналов, которые нужно сохранить.

Все сигналы указанного модуля и все сигналы всех модулей ниже по иерархии будут сохранены

- `$dumplimit(<размер>);`

Задаёт максимальный допустимый размер файла с результатами. Размер задаётся в байтах.

Уровень=1

Будут сохранены все сигналы только указанного модуля

Уровень=2

Будут сохранены все сигналы указанного модуля и все сигналы модулей, расположенных только на 1 уровень ниже

- `$dumpoff;`

Приостановить сохранение данных

- `$dumpon;`

Продолжить сохранение данных

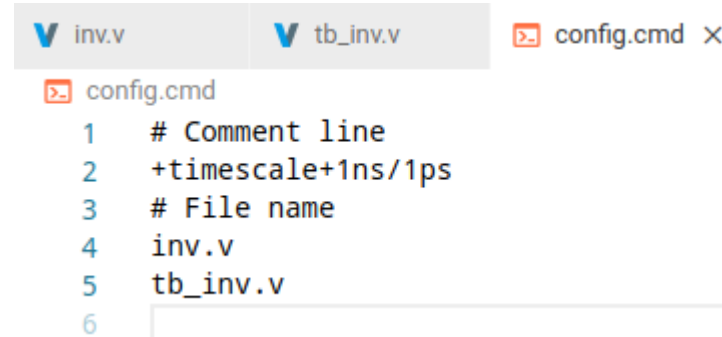
Использование командных файлов в Icarus Verilog

Файл inv.v

```
module inv(x, y);  
    input x;  
    output y;  
    assign y = ~x;  
endmodule
```

Файл tb_inv.v

```
module tb_inv;  
    reg x;  
    wire y;  
  
    inv dut(x, y);  
  
    initial begin  
        $dumpfile("inv.vcd");  
        $dumpvars(1, tb_inv);  
        x = 0;  
        #100 $finish;  
    end  
    always #5 x = ~x;  
endmodule
```



```
inv.v  tb_inv.v  config.cmd x  
config.cmd  
1 # Comment line  
2 +timescale+1ns/1ps  
3 # File name  
4 inv.v  
5 tb_inv.v  
6
```

```
student@localhost:~/t1002817> iverilog -c config.cmd  
student@localhost:~/t1002817> vvp a.out  
VCD info: dumpfile inv.vcd opened for output.  
student@localhost:~/t1002817> █
```


Использование скриптов на bash для автоматизации запуска

```
#!/usr/bin/bash
```

```
if [ -f ./a.out ]; then  
    rm ./a.out  
fi
```

```
iverilog -c ./config.cmd  
if [ -f ./a.out ]; then  
    vvp ./a.out  
fi
```

```
#!/usr/bin/bash
```

```
if [ -f ./a.out ]; then  
    rm ./a.out  
fi
```

```
iverilog -c ./config.cmd  
if [ -f ./a.out ]; then  
    ./vvp a.out  
    gtkwave ./inv.vcd  
else  
    echo -e "\e[31m Verilog code compilation error!\e[0m"  
fi
```

```
● student@localhost:~/t1002817> ./run.sh  
tb_inv.v:5: error: Unknown module type: in  
2 error(s) during elaboration.  
*** These modules were missing:  
        in referenced 1 times.  
***  
Verilog code compilation error!  
○ student@localhost:~/t1002817> █
```