

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Безопасности жизнедеятельности

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Языки проектирования аппаратуры»
Тема: Структура программы на VerilogHDL

Студенты гр. 1308

Мельник Д. А.

Лепов А. В.

Преподаватель

Гречухин М.Н.

Санкт-Петербург

2024

Цель работы.

Изучение вопросов спецификации объекта и анализ программы на алгоритмическом уровне с использованием моделирования.

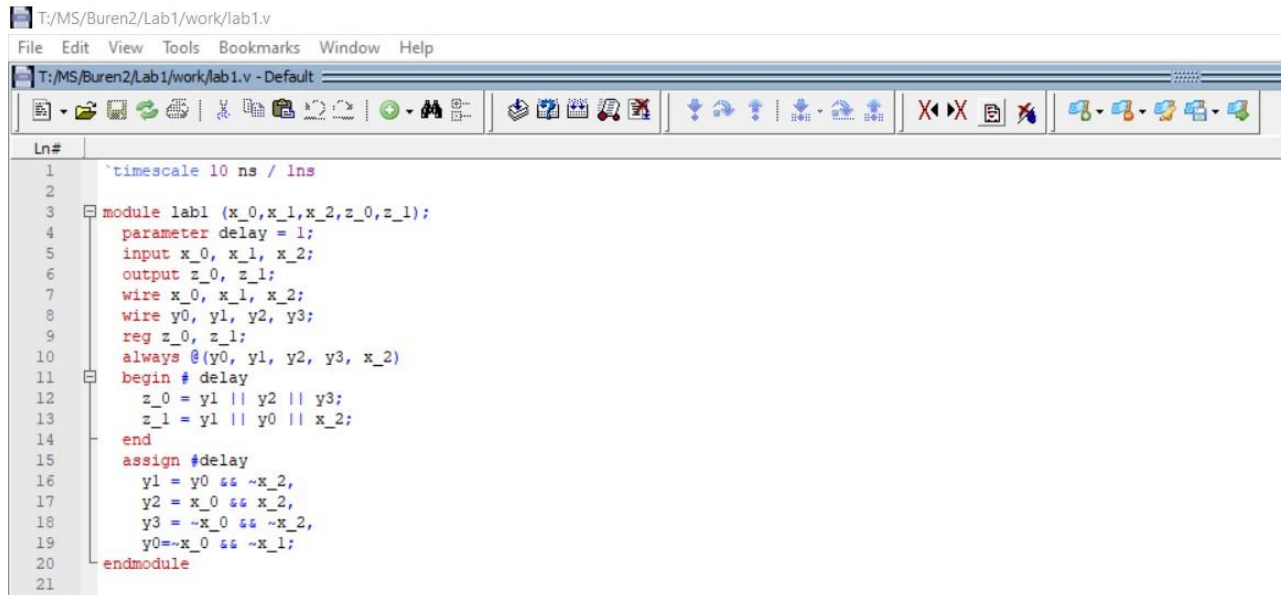
Задание на лабораторную работу

1. Запустить программу ModelSim-Altera, создать проект, включив в него файл Lab1.v. Просмотреть в редакторе текст файла Lab1.v.
2. Выполнить компиляцию проекта.
3. Запустить процедуру моделирования, вызвав команду Simulate. Открыть окна наблюдения Source, Process, Signals, Wave, List.
4. Сформировать тестовую последовательность, используя команду системы моделирования Force (в пределах необходимого для полноты проверки числа комбинаций входных данных).
5. Выполнить моделирование в пошаговом режиме. Обратит внимание на состояния процессов и «скачки» курсора между операторами программы в процессе интерпретации, объяснить, почему это происходит? Наблюдать моменты активизации процессов.
6. Выполнить моделирование в автоматическом режиме. Просмотреть временную диаграмму в окне Wave и убедиться в правильности вычисления логических функций.
7. Настроить и просмотреть окно List. Обратите внимание на то, что различным временным отметкам соответствует разное число строк в списке (календаре событий). Почему?
8. Добавить в проект файл Lab1_m, выполнить компиляцию этого файла и пункты 6 и 7, объявив при этом модуль Lab1_m вершиной проекта.
9. Заменить в операторе always блокирующие присваивания на неблокирующие. Выполнить моделирование в автоматическом режиме и оценить разницу диаграмм по пп. 9 и 10. Почему возникает различие? Какой вариант является ошибочным.

10. Восстановить Lab1_m.v в исходном виде, а затем добавить опции задержки компонентов 5 nS, выполнить компиляцию и моделирование, сравнить результаты с ранее полученными

Ход выполнения работы

1. Создание проекта в ModelSim



The screenshot shows the ModelSim source code editor with the file path T:/MS/Buren2/Lab1/work/lab1.v. The code is as follows:

```
1 `timescale 10 ns / 1ns
2
3 module lab1 (x_0,x_1,x_2,z_0,z_1);
4     parameter delay = 1;
5     input x_0, x_1, x_2;
6     output z_0, z_1;
7     wire x_0, x_1, x_2;
8     wire y0, y1, y2, y3;
9     reg z_0, z_1;
10    always @(y0, y1, y2, y3, x_2)
11    begin # delay
12        z_0 = y1 || y2 || y3;
13        z_1 = y1 || y0 || x_2;
14    end
15    assign #delay
16        y1 = y0 && ~x_2,
17        y2 = x_0 && x_2,
18        y3 = ~x_0 && ~x_2,
19        y0 = ~x_0 && ~x_1;
20 endmodule
21
```

Рисунок 1. Код lab1.v в окне source

2. Компиляция проекта

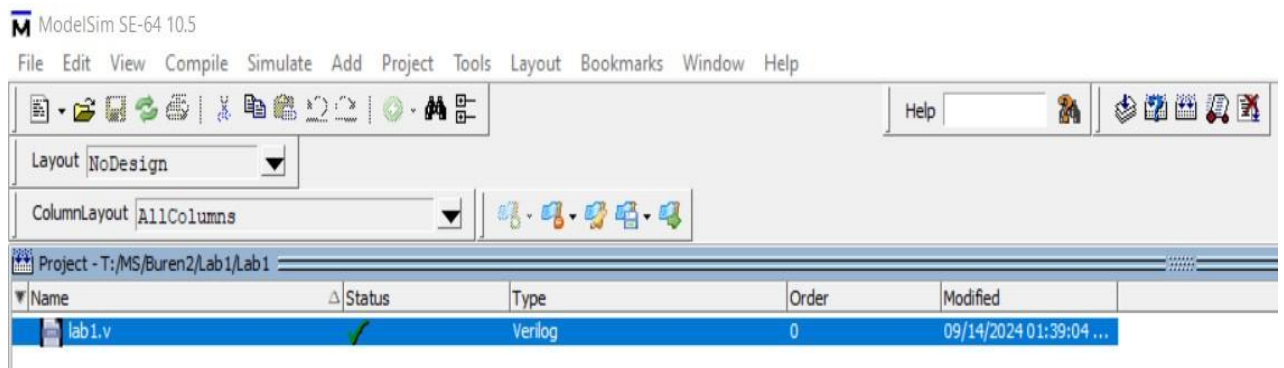


Рисунок 2. Удачный результат компиляции проекта

3. Первое моделирование проекта

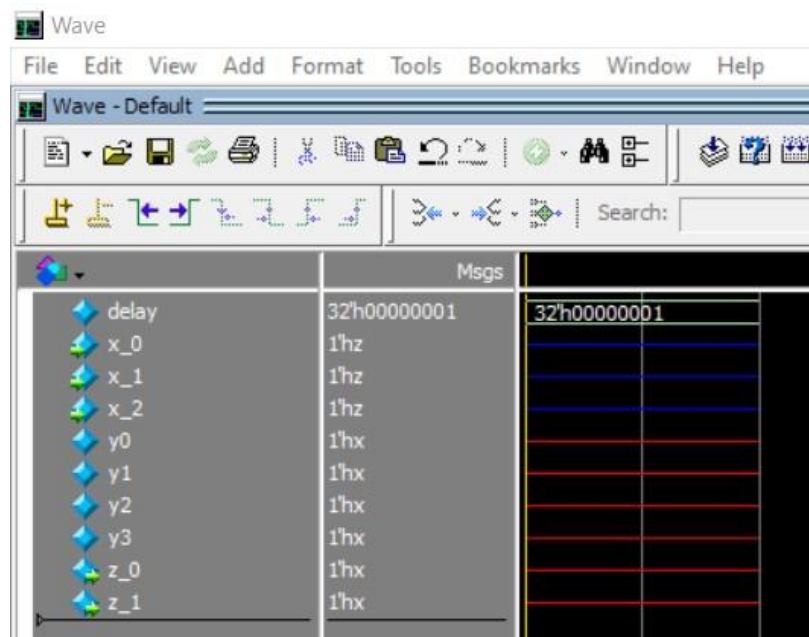


Рисунок 3. Окно "Wave" моделирования проекта

На диаграмме видны красные линии – неопределённые сигналы. Эти сигналы являются неопределёнными, так как перед моделированием не были заданы тестовые воздействия (x_0 , x_1 , x_2).

4. Формирование тестовой последовательности

```
VSIM 14> force x_0 0 0ns, 1 40ns  
VSIM 15> force x_1 0 0ns, 1 20ns, 0 40ns, 1 60ns  
VSIM 16> force x_2 0 0ns, 1 10ns, 0 20ns, 1 30ns, 0 40ns, 1 50ns, 0 60ns, 1 70ns
```

Рисунок 4. Команды формирования тестовой последовательности

Таким образом была сформирована типовая тестовая последовательность, в которой перебираются трёхзначные бинарные числа от 0 до 7 (000 до 111)

5. Первое моделирование в пошаговом режиме

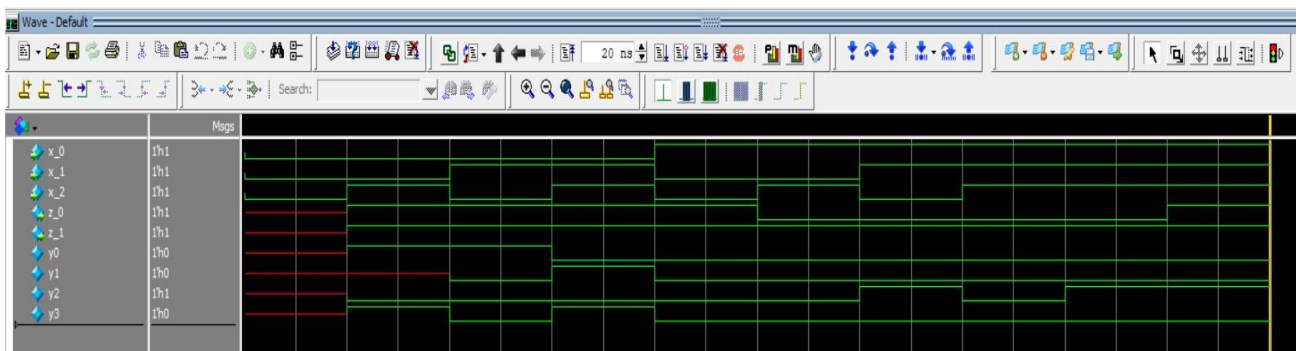


Рисунок 5. Диаграмма из окна "Wave" проекта

Так как в этот раз были заданы тестовые воздействия, на диаграмме видны переходы состояний сигналов во времени. Следует обратить внимание на неопределённость ряда сигналов первые 10 нс (красные линии на нижних графиках): это вызвано тем, что эти элементы работают с указанной в коде задержкой *delay*.

```
Ln# 1 `timescale 10 ns / 1 ns
2
3 module lab1 (x_0,x_1,x_2,z_0,z_1);
4     parameter delay = 1;
5     input x_0, x_1, x_2;
6     output z_0, z_1;
7     wire x_0, x_1, x_2;
8     wire y0, y1, y2, y3;
9     reg z_0, z_1;
10    always @(y0, y1, y2, y3, x_2)
11    begin # delay
12        z_0 = y1 || y2 || y3;
13        z_1 = y1 || y0 || x_2;
14    end
15    assign #delay
16        y1 = y0 && ~x_2,
17        y2 = x_0 && x_2,
18        y3 = ~x_0 && ~x_2,
19        y0=~x_0 && ~x_1;
20 endmodule
21
```

Рисунок 6. Курсор при пошаговом моделировании.

В ходе пошагового моделирования наблюдались скачки курсора между операторами внутри блока *always*. При каждом шаге курсор мог как стоять на месте, так и передвигаться между красными точками. Это вызвано тем, что код

в блоке `always` в данном случае выполняется последовательно при каждом изменении `y0`, `y1`, `y2`, `y3`, `x2`. Эти переменные изменяются не на каждом шаге.

6. Детали моделирования

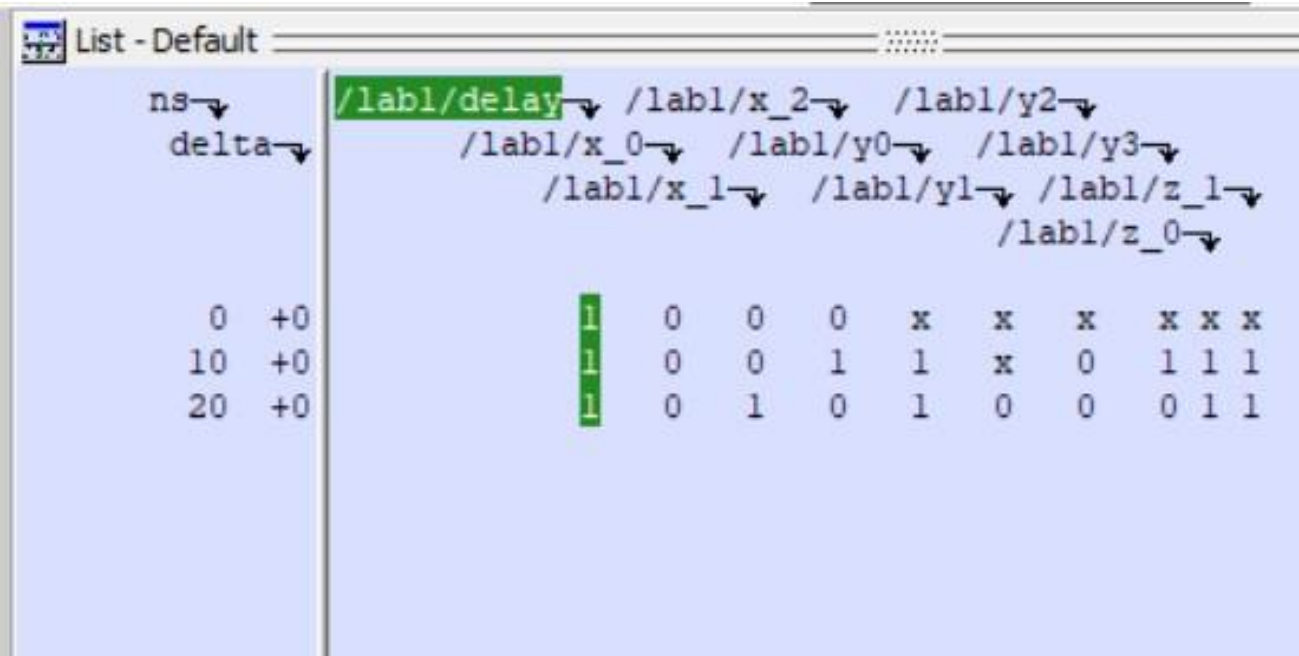


Рисунок 7. Окно `List` на начальном этапе моделирования

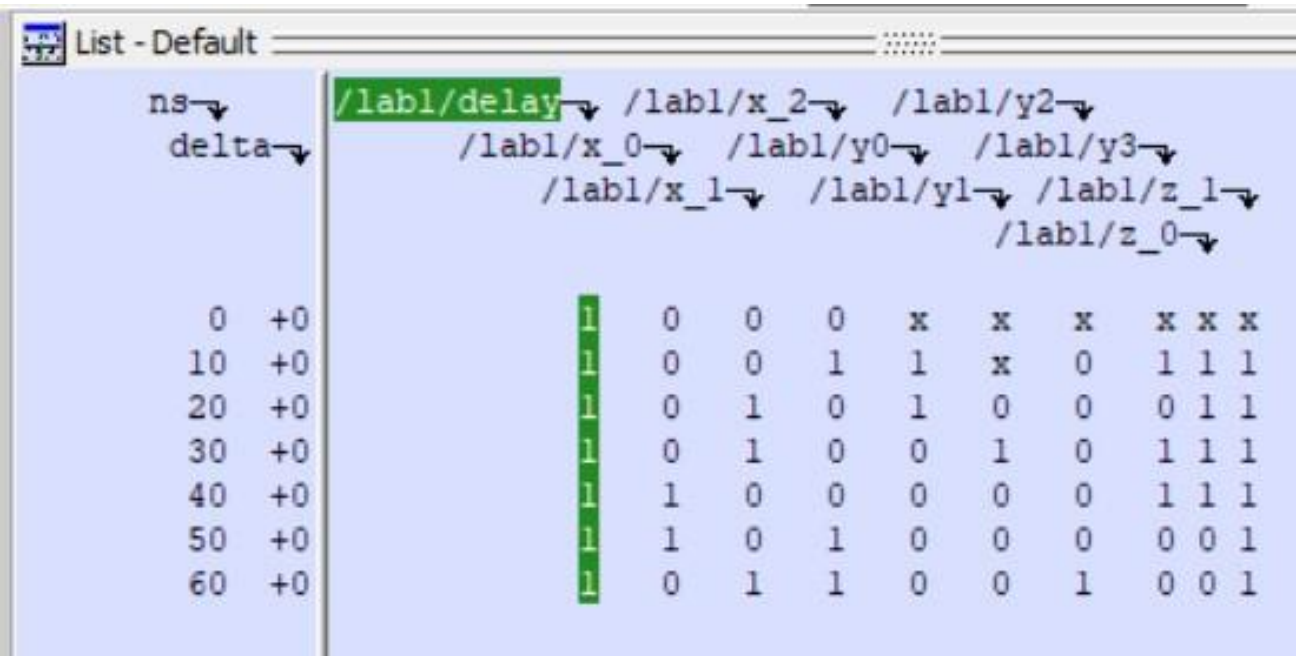
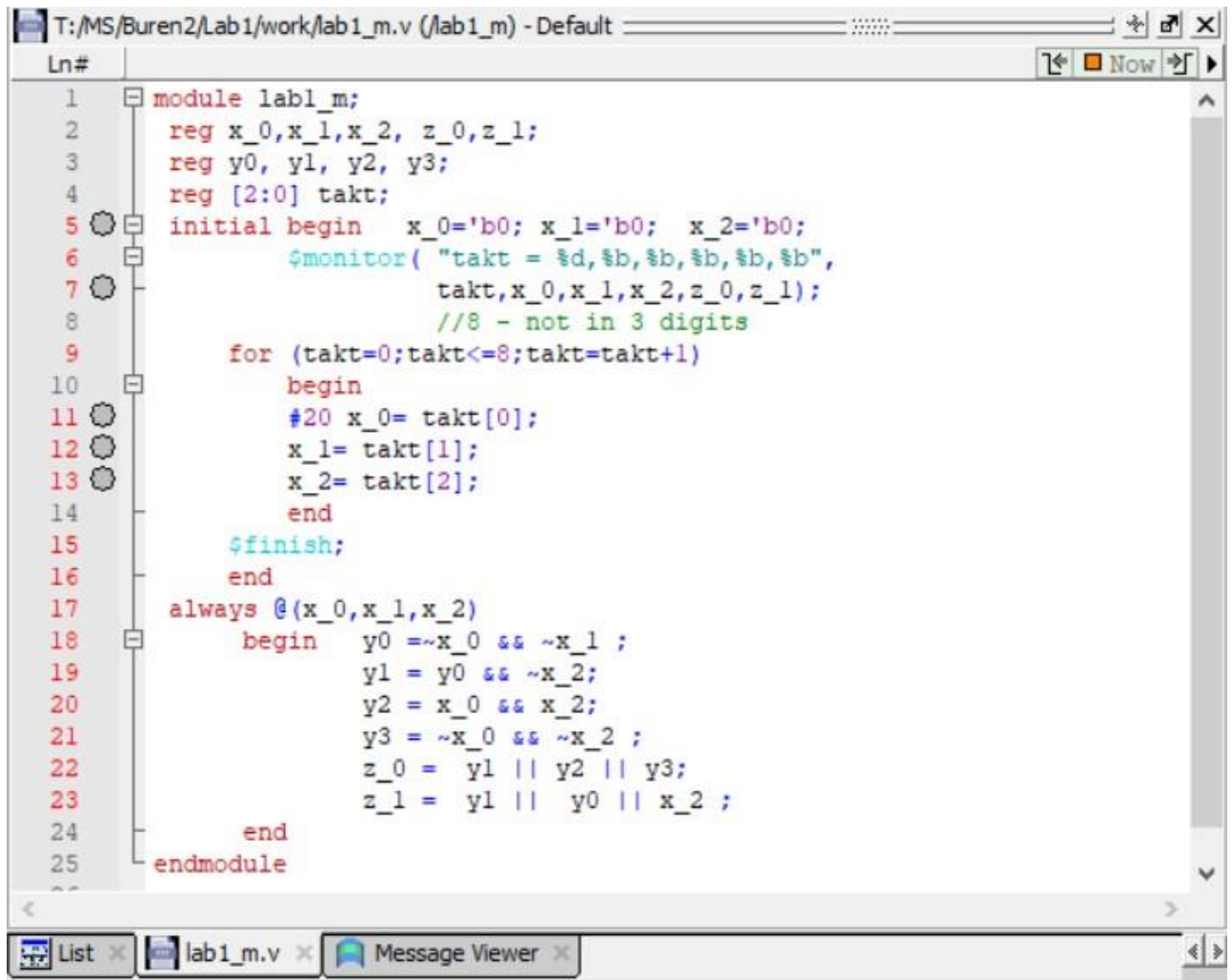


Рисунок 8. Дальнейшее отображение окна `List`

7. Применение автоматического тестирования модуля



```
1 module lab1_m;
2   reg x_0,x_1,x_2, z_0,z_1;
3   reg y0, y1, y2, y3;
4   reg [2:0] takt;
5   initial begin    x_0='b0; x_1='b0; x_2='b0;
6                   $monitor( "takt = %d,%b,%b,%b,%b,%b",
7                             takt,x_0,x_1,x_2,z_0,z_1);
8                   //8 - not in 3 digits
9                   for (takt=0;takt<=8;takt=takt+1)
10                      begin
11                        #20 x_0= takt[0];
12                        x_1= takt[1];
13                        x_2= takt[2];
14                      end
15                   $finish;
16                end
17   always @(x_0,x_1,x_2)
18     begin    y0 =~x_0 && ~x_1 ;
19             y1 = y0 && ~x_2;
20             y2 = x_0 && x_2;
21             y3 = ~x_0 && ~x_2 ;
22             z_0 = y1 || y2 || y3;
23             z_1 = y1 || y0 || x_2 ;
24          end
25 endmodule
```

Рисунок 9. Отображение окна Source с кодом lab1_m.v

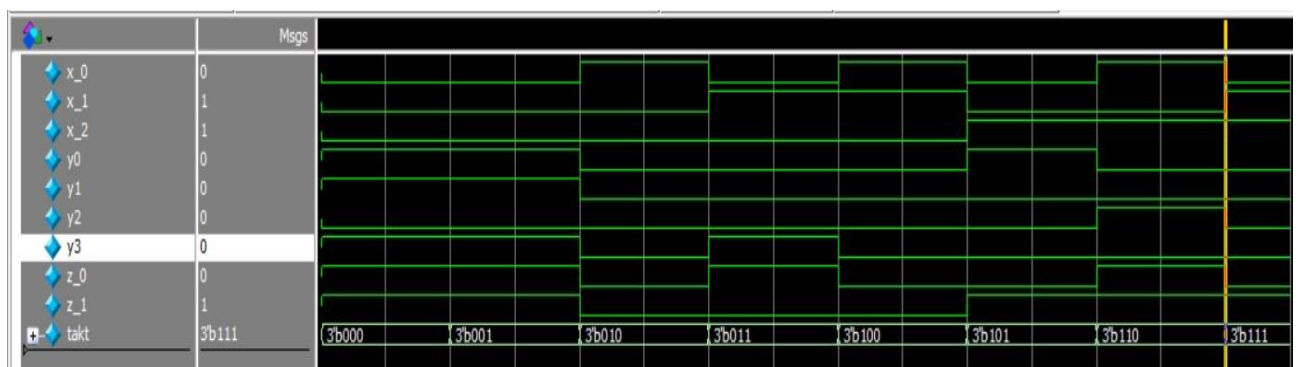


Рисунок 10. Временная диаграмма моделирования модуля

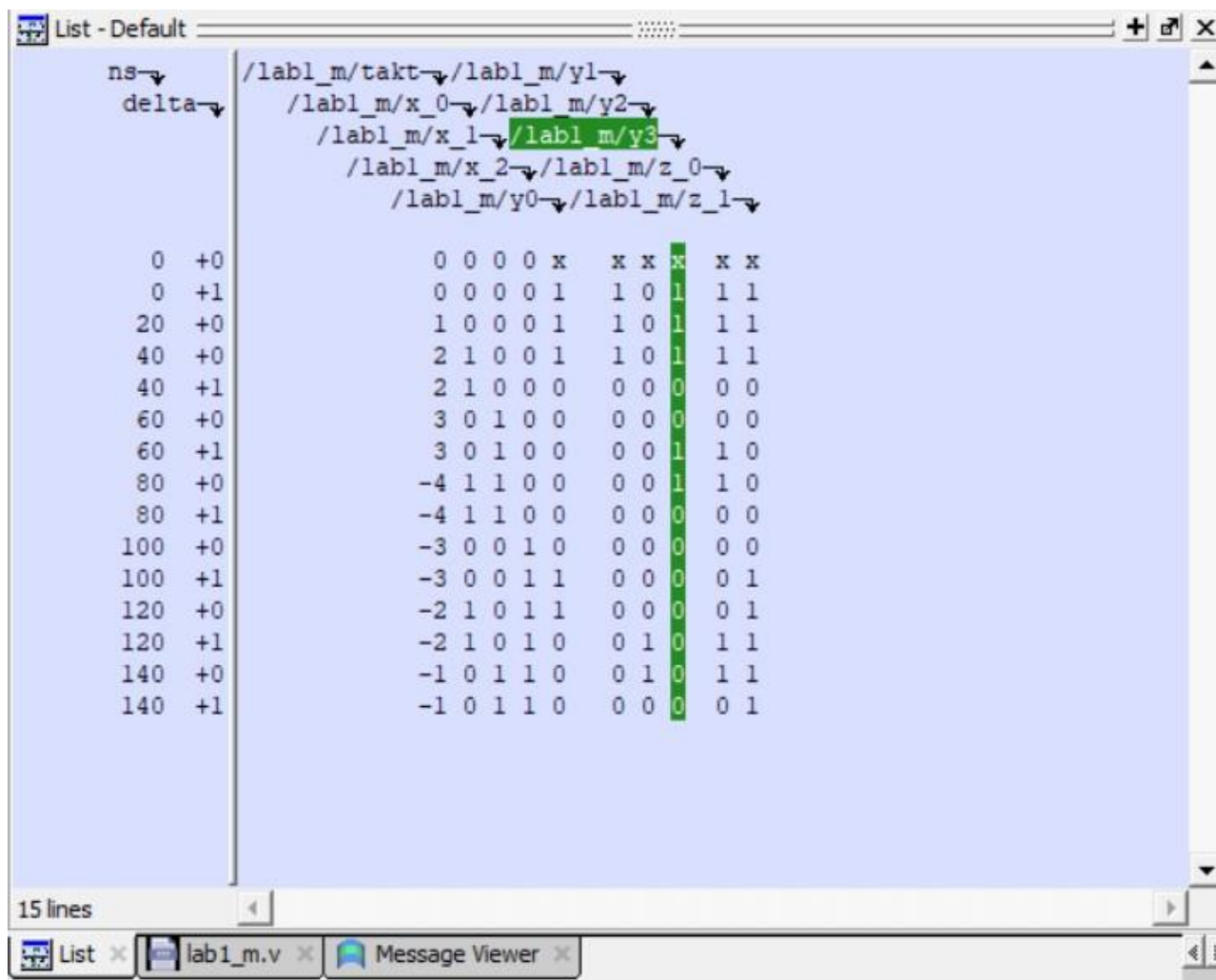


Рисунок 11. Окно List моделирования модуля

8. Модификация программы *lab1_m.v*

В первом варианте программы *lab1_m.v* применялись блокирующие присваивания. Главным отличием блокирующего присваивания от неблокирующего является действия программы после присвоения переменной нового значения: при блокирующем присваивании новое значение применяется сразу (блоки *always* связанные с этой переменной перезапускаются сразу), при неблокирующем новое значение применяется только после окончания блока кода.

В нашем случае весь исполняемый код находится внутри блока *always*. Все операторы в этом блоке выполняются последовательно. При блокирующем присваивании все переменные *y* получают своё значение **сразу**, из-за этого все параметры при вычислении переменных *z* определены => на первой же итерации

все переменные определены и выведены на диаграмму моделирования зелёным цветом (рисунок 10).

```
always @(x_0,x_1,x_2)
begin
    y0 = ~x_0 && ~x_1 ;
    y1 = y0 && ~x_2;
    y2 = x_0 && x_2;
    y3 = ~x_0 && ~x_2 ;
    z_0 = y1 || y2 || y3;
    z_1 = y1 || y0 || x_2 ;
end
```

Рисунок 12. Рассматриваемы блок "always"

При замене типа присваивания (= на <=) при вычислении переменных *z* переменные *y* будут неопределёнными (они станут определёнными только при следующей итерации) поэтому переменные *z* тоже будут неопределёнными после первой итерации, что видно по моделированию (Рисунок 13).

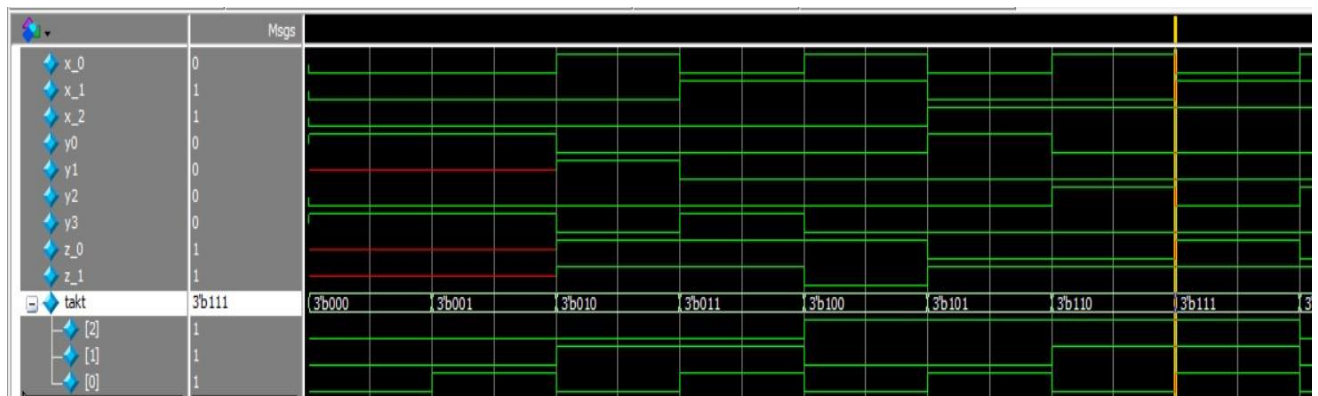


Рисунок 13. Диаграмма моделирования модифицированного кода Lab1_m.v

Следует отметить, что *y1* тоже будет неопределённой так как при её вычислении применяется неопределённая *y0*.

```

T:/MS/Buren2/Lab1/work/lab1_m.v (/lab1_m) - Default
Ln#
1  module lab1_m;
2      reg x_0,x_1,x_2, z_0,z_1;
3      reg y0, y1, y2, y3;
4      reg [2:0] takt;
5  initial begin  x_0='b0; x_1='b0; x_2='b0;
6                  $monitor( "takt = %d,%b,%b,%b,%b,%b",
7                              takt,x_0,x_1,x_2,z_0,z_1);
8                      //8 - not in 3 digits
9      for (takt=0;takt<=8;takt=takt+1)
10     begin
11         #20 x_0= takt[0];
12         x_1= takt[1];
13         x_2= takt[2];
14     end
15     $finish;
16 end
17 always @(x_0,x_1,x_2)
18     begin  y0 <=~x_0 && ~x_1 ;
19             y1 <= y0 && ~x_2;
20             y2 <= x_0 && x_2;
21             y3 <= ~x_0 && ~x_2 ;
22             z_0 <= y1 || y2 || y3;
23             z_1 <= y1 || y0 || x_2 ;
24     end
25 endmodule

```

Рисунок 14. Модифицированный код Lab1_m.v

Введение задержки на модуль.

Для учёта неидеальности ПЛИС при моделировании вводятся задержки. В данном случае введена задержка в 5 единиц модельного времени.

```
module lab1_m;
  reg x_0,x_1,x_2, z_0,z_1;
  reg y0, y1, y2, y3;
  reg [2:0] takt;
  initial begin    x_0='b0; x_1='b0; x_2='b0;
                  $monitor( "takt = %d,%b,%b,%b,%b,%b",
                              takt,x_0,x_1,x_2,z_0,z_1);
                  //8 - not in 3 digits
                  for (takt=0;takt<=8;takt=takt+1)
                    begin
                      #20 x_0= takt[0];
                      x_1= takt[1];
                      x_2= takt[2];
                    end
                  $finish;
                end
  always @(x_0,x_1,x_2)
    begin
      #5
      y0 =~x_0 && ~x_1 ;
      y1 = y0 && ~x_2;
      y2 = x_0 && x_2;
      y3 = ~x_0 && ~x_2 ;
      z_0 = y1 || y2 || y3;
      z_1 = y1 || y0 || x_2 ;
    end
endmodule
```

Рисунок 15. Код с введённой задержкой (#5)

Введённую задержку можно наблюдать на временной диаграмме:

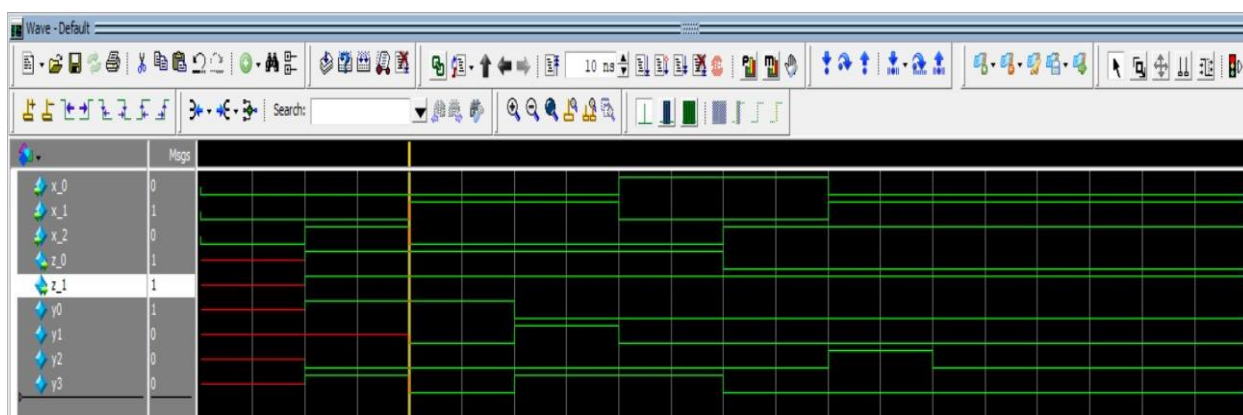


Рисунок 16. Временная диаграмма модуля с временными задержками

Наличие задержек больше всего заметно по нескольким неопределённым значениям переменных. Если приглядеться, эти переменные стоят в блоке *always* после оператора задержки #5. Также эти задержки можно наблюдать по сдвигу изменений переменных относительно изменений входных сигналов.

Выводы.

В ходе выполнения лабораторной работы была изучена структура описания модуля на языке Verilog. Были изучены средства моделирования модулей в среде ModelSim. Также были рассмотрены различия между блокирующим и неблокирующим присваиванием.

Примечание: для избегания бесконечного цикла в программе с автоматическим заданием тестовых последовательностей следует увеличить разрядность переменной `takt` на 1 (`[3:0]` а не `[2:0]`). Это позволит избежать переполнения счётчика.

ПРИЛОЖЕНИЕ

Kod lab1.v:

```
`timescale 10 ns / 1 ns

module lab1 (x_0,x_1,x_2,z_0,z_1);
    parameter delay = 1;
    input x_0, x_1, x_2;
    output z_0, z_1;
    wire x_0, x_1, x_2;
    wire y0, y1, y2, y3;
    reg z_0, z_1;
    always @(y0, y1, y2, y3, x_2)
    begin # delay
        z_0 = y1 || y2 || y3;
        z_1 = y1 || y0 || x_2;
    end
    assign #delay
        y1 = y0 && ~x_2,
        y2 = x_0 && x_2,
        y3 = ~x_0 && ~x_2,
        y0=~x_0 && ~x_1;
endmodule
```

Kod lab1_m.v:

```
module lab1_m;
  reg x_0,x_1,x_2, z_0,z_1;
  reg y0, y1, y2, y3;
  reg [2:0] takt;
  initial begin    x_0='b0; x_1='b0; x_2='b0;
                  $monitor( "takt = %d,%b,%b,%b,%b,%b",
                           takt,x_0,x_1,x_2,z_0,z_1);
                  //8 - not in 3 digits
                  for (takt=0;takt<=8;takt=takt+1)
                    begin
                      #20 x_0= takt[0];
                      x_1= takt[1];
                      x_2= takt[2];
                    end
                  $finish;
                  end
  always @(x_0,x_1,x_2)
    begin
      #5
      y0 =~x_0 && ~x_1 ;
      y1 = y0 && ~x_2;
      y2 = x_0 && x_2;
      y3 = ~x_0 && ~x_2 ;
      z_0 = y1 || y2 || y3;
      z_1 = y1 || y0 || x_2 ;
    end
endmodule
```