Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: архитектура вычислительных систем

Отчет по лабораторной работе №2

ПЛИС-устройства и разработка на языке SystemVerilog

Выполнил: студент: гр. 053506
Слуцкий Никита Сергеевич

Проверил: ст. преподаватель
Шиманский Валерий Владимирович

Минск 2022

СОДЕРЖАНИЕ

1. Введение
2. Постановка задачи и результаты выполнения
3. Выводы
4. Литература

Введение

Цели данной работы:

1. Научиться использовать инструменты программируемых логических интегральных схем (ПЛИС/FPGA)
2. Разрабатывать и симулировать описание аппаратуры на SystemVerilog, а затем синтезировать его для ПЛИС

В ходе выполнения данной лабораторной работы будут получены навыки проектирования простых и нетривиальных логических схем, изучены основы разработки схем конечных автоматов, изучены основы и не только языка SystemVerilog, а также получены навыки работы в ПО Intel Quartus Prime и ModelSim.

Постановка задачи и результаты выполнения

Задание 1: Описание инструментов

Это задание призвано ознакомить с процессом и инструментами разработки. С инструментами ознакомился, необходимые результаты получил. Так как это необязательное и ознакомительное задание, его формулировка и результаты опускаются (но тем не менее задание было выполнено).

Задание 2: Разработка структурного описания конечных автоматов

Номер в журнале 21, Вариант 1

Задание для этой лабораторной работы заключается в разработке конечного автомата в SystemVerilog для контроля задних огней автомобиля 1965 г. Ford Thunderbird. На каждой стороне есть три огня, которые зажигаются последовательно, тем самым указывая направление поворота. На рис. 1 показаны задние огни автомобиля, а на рис. 2 показана последовательность мигания для левых огней (a) и правых огней (b).

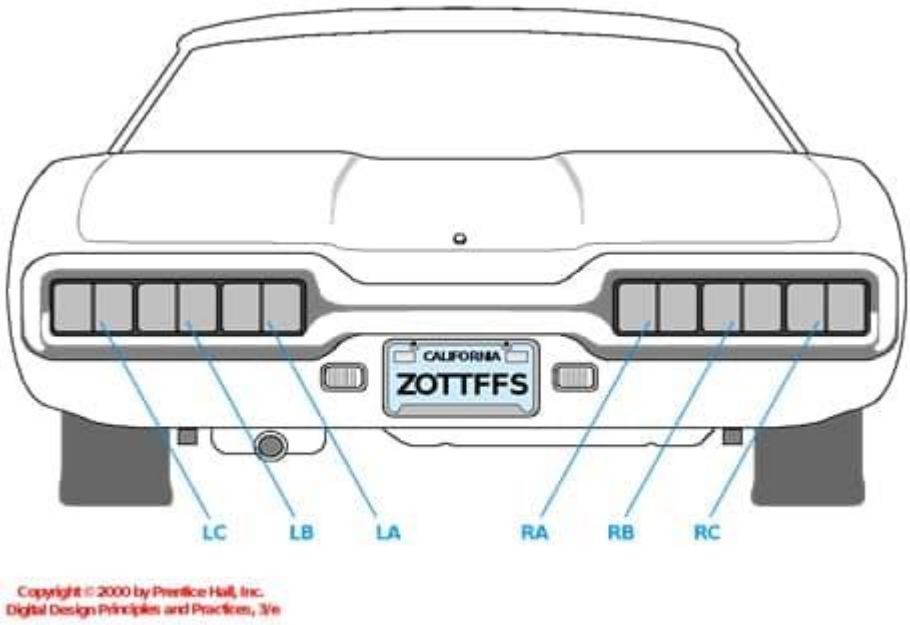

Рисунок 1

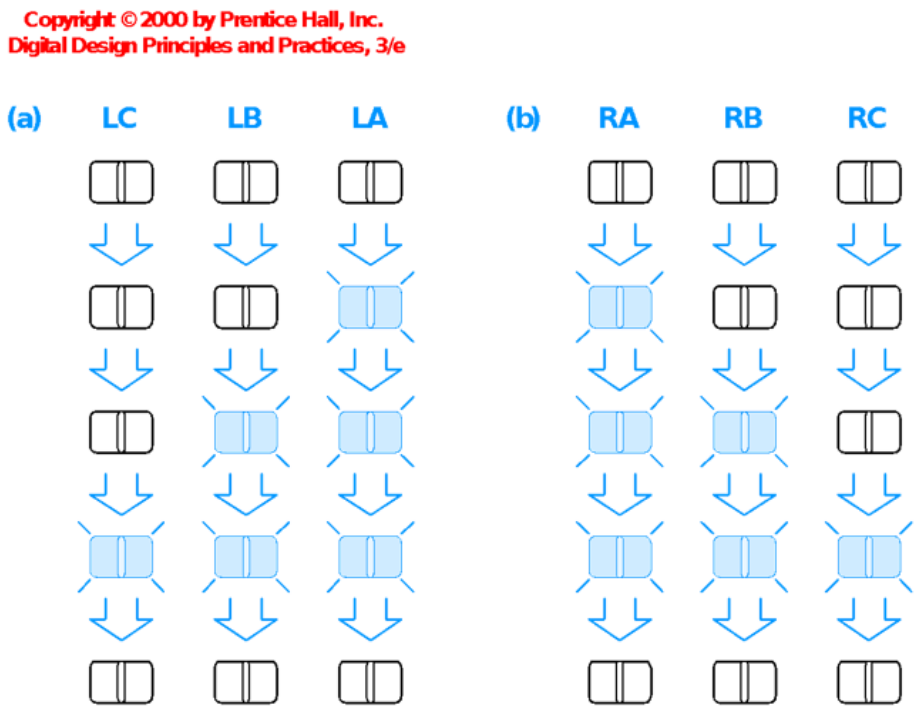

Рисунок 2

КА должен иметь два входных контакта, левый и правый, которые запускают последовательность мигания в цикле после их включения. В любой момент времени включется только один из двух входов (левый или правый). КА должен иметь шесть выходных контактов, LA, LB, LC, RA, RB и RC. После запуска последовательности она должна продолжаться, даже если сигнал на входе сброше. Когда последовательность завершится, она должна на цикл вернуться в состояние с выключенными огнями, прежде чем можно будет начать другую последовательность. Пример ожидаемого поведения см. в файле тестового вектора Thunderbird далее в этой лабораторной работе.

- Получить у преподавателя свой вариант включения поворотных сигналов. Составьте тестовые векторы.
- Составить диаграмму переходов состояний для вашего КА.
- Выбрать кодировку состояний
- Составть схему.

- Написать **структурный код SystemVerilog** для КА
- Смоделировать свой КА с помощью следующего самопроверяющегося тестового стенда и векторов. Изучить тестовый стенд и проследить, как он применяет входы и проверяет выходы

Выполнение:

В соответствии с вариантом задания №1 я построил автомат и прописал состояния в нём. Это представлено на рисунке ниже.

	A	B	C	D	E	F	G
1	**Вариант 1**	Сигнал левого поворота			Сигнал правого поворота		
2	Этап	LC	LB	LA	RA	RB	RC
3	1	0	0	0	0	0	0
4	2	0	0	1	1	0	0
5	3	0	1	1	1	1	0
6	4	1	1	1	1	1	1
7	5	0	0	0	0	0	0

Рисунок 3

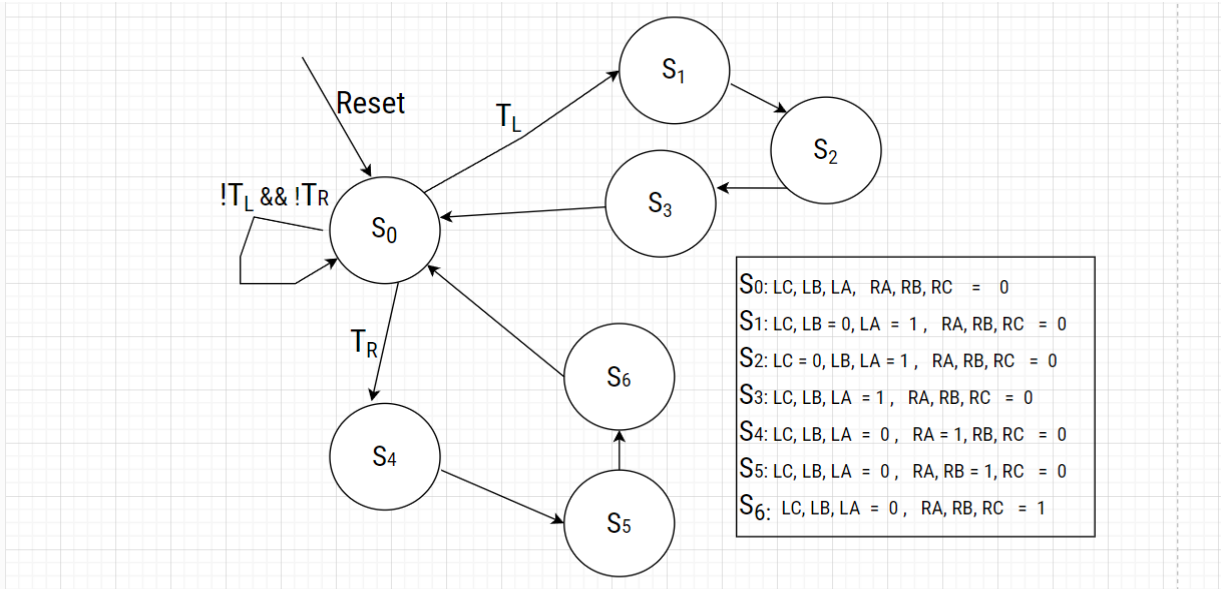

Рисунок 4

В соответствии с смоделированным на листе автоматом я написал модуль CarsLights на языке SystemVerilog:

```
module CarsLights(
input logic clk, input logic reset,
input logic L, input logic R,
output logic LA, output logic LB, output logic LC,
output logic RA, output logic RB, output logic RC
);
   typedef enum logic [2:0] {S0, S1, S2, S3, S4, S5, S6} statetype;

   statetype state, nextstate;

   // регистр состояния
   always_ff @(posedge clk, posedge reset)
      if (reset) state <= S0;
      else state <= nextstate;

   // логика следующего состояния
   always_comb
      case (state)
      S0: if (L) nextstate = S1;
         else if (R) nextstate = S4;
         else nextstate = S0 ;
      S1: nextstate = S2;
      S2: nextstate = S3;
      S3: nextstate = S0;
      S4: nextstate = S5;
      S5: nextstate = S6;
      S6: nextstate = S0;
      default: nextstate = S0;
   endcase


   // выходная логика
   assign LA = (state == S3 | state == S2 | state == S1);
   assign LB = (state == S2 | state == S3);
   assign LC = (state == S3);

   assign RA = (state == S4 | state == S5 | state == S6);
   assign RB = (state == S5 | state == S6);
   assign RC = (state == S6);

endmodule
```

Синтезированная в RTL Viewer программы Quartus схема имеет следующий вид:

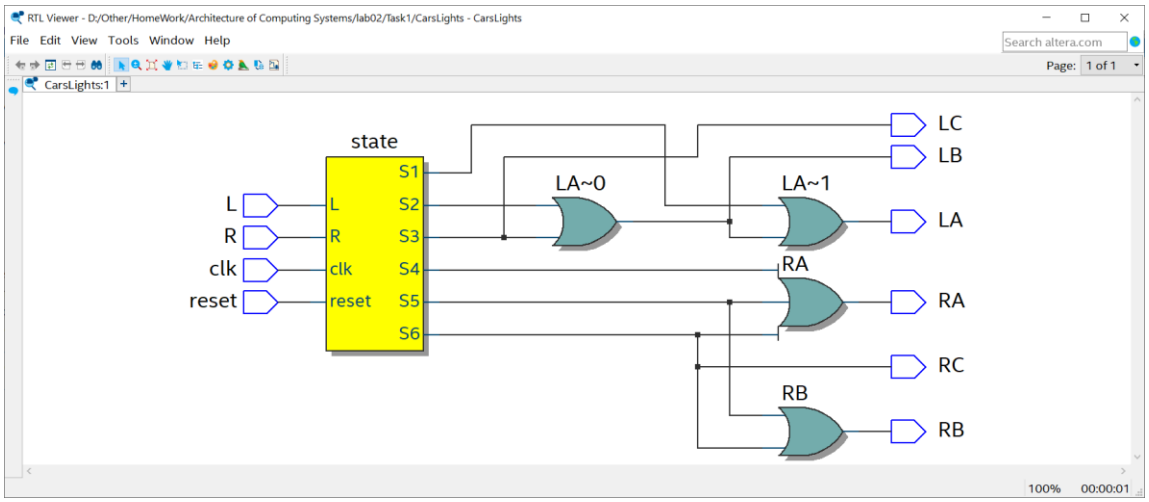

Рисунок 5

Соотношение сигналов и контактов на устройстве выполнено в соответствии с документацией от Intel. Оно представлено на скриншоте из окна Pin Planner программы Quartus:

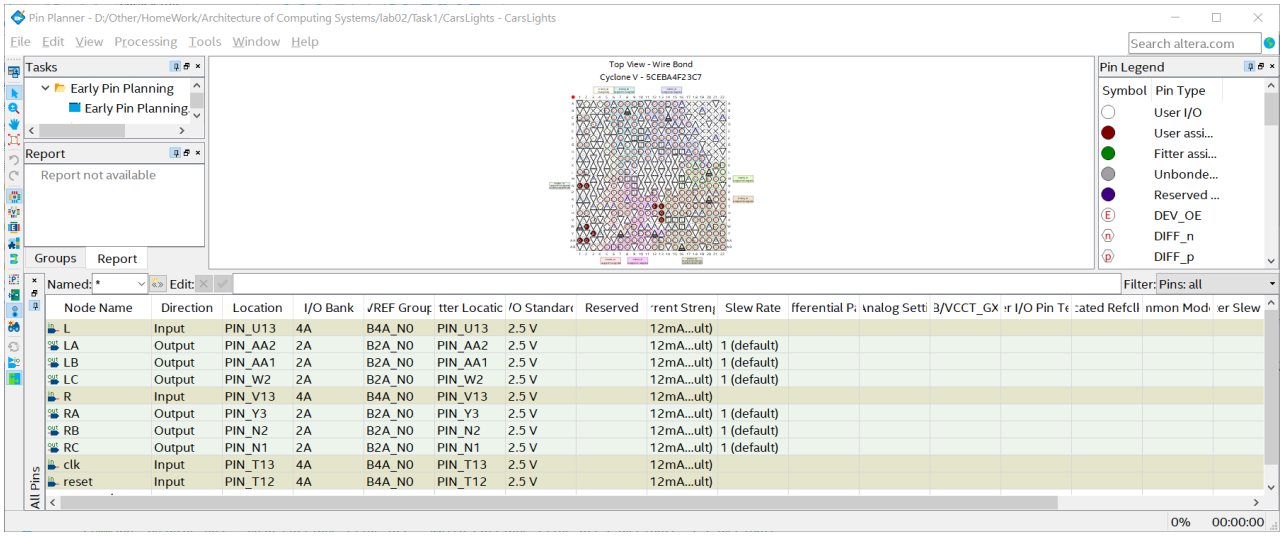

Рисунок 6

Далее я написал модуль testbench для синтеза и проверки в ModelSim.

Модуль имеет следующую реализацию:

```
module testbench();
  logic        clk, reset;
  logic        left, right, la, lb, lc, ra, rb, rc;
  logic [5:0]  expected;
  logic [31:0] vectornum, errors;
  logic [7:0]  testvectors[10000:0];

// инстанцировать тестируемое устройство
CarsLights dut(clk, reset, left, right, la, lb, lc, ra, rb, rc);

// generate clock
always
  begin
    clk=1; #5; clk=0; #5;
  end

// на старте теста, загрузите вектора и запустите сброс
initial
  begin
    $readmemb("thunderbird.tv", testvectors);
    vectornum = 0; errors = 0; reset = 1; #22; reset = 0;
  end

// применение тестовых векторов по нарастающему фронту тактового сигнала
always @(posedge clk)
  begin
    #1; {left, right, expected} = testvectors[vectornum];
  end

// проверка результатов по спадающему фронту сигнала clk
always @(negedge clk)
  if (~reset) begin    // skip during reset
    if ({la, lb, lc, ra, rb, rc} !== expected) begin // check result
      $display("Error: inputs = %b", {left, right});
      $display(" outputs = %b %b %b %b %b %b (%b expected)",
        la, lb, lc, ra, rb, rc, expected);
      errors = errors + 1;
    end
    vectornum = vectornum + 1;
    if (testvectors[vectornum] === 8'bx) begin
      $display("%d tests completed with %d errors", vectornum, errors);
      $stop;
    end
  end
endmodule
```

В программе ModelSim были синтезированы файлы и проведена работа с помощью тестового стенда testbench. Скриншоты с осциллограммой и выводом про успешное прохождение тестов представлены на скриншотах ниже.

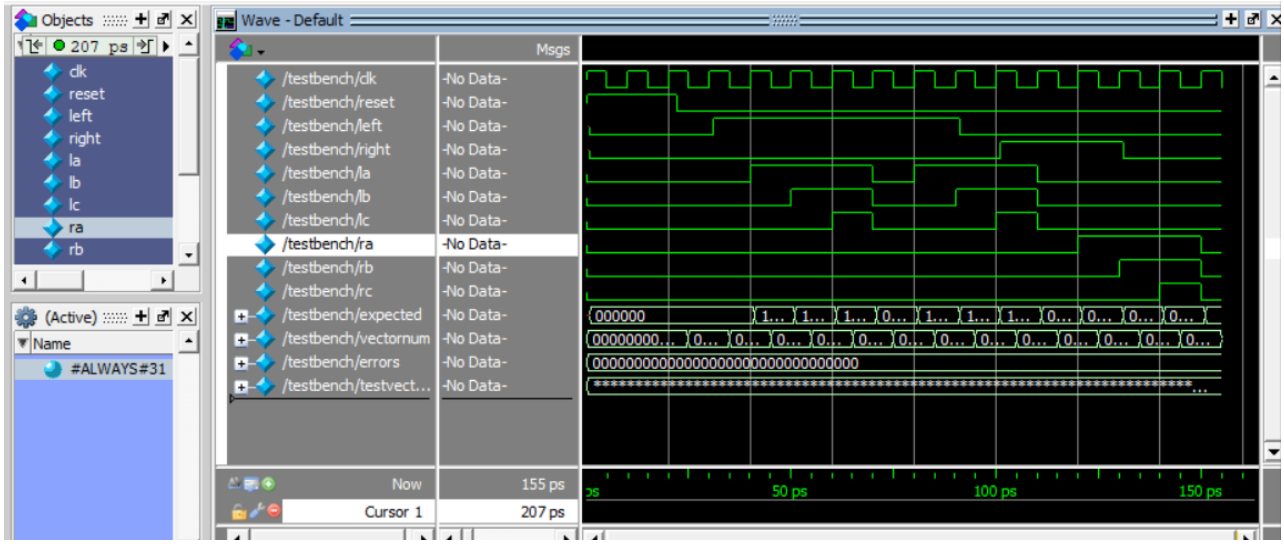

Рисунок 7

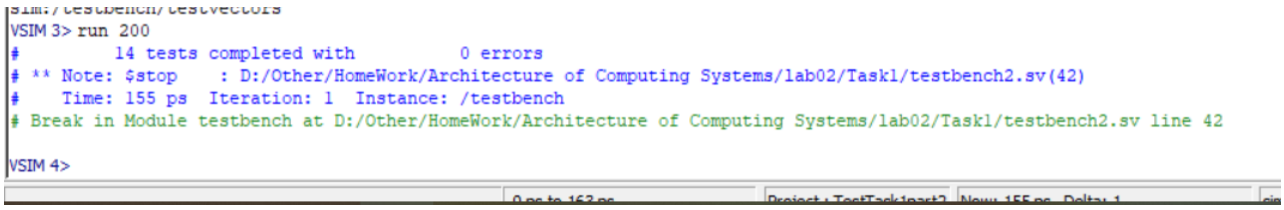

Рисунок 8

Делаю вывод, что модуль выполняет то, что от него ожидаю, что он соответствует спроектированному автомату. Благодаря RTL Viewer можно посмотреть, как реализовать модуль с использованием логических компонентов из изученной в первой лабораторной работы программы Logisim.

Задание 2 Разработка поведенческой модели конечных автоматов

Цель этого задания заключается в разработке конечного автомата, используя поведенческий SystemVerilog.

Приключенческая игра

Приключенческая игра-квест, которую вы будете разрабатывать, имеет семь комнат и один объект - меч. Игра начинается в локации "Пещера какофонии". Чтобы пройти игру, вы сначала должны последовать на восток через локацию "Извилистый туннель", а затем на юг к "Стремительной реке". После чего, на западе в локации "Тайник с мечом" вы должны найти "Вострый меч". Меч позволит вам безопасно пройти на восток от "Стремительной реки" через "Логово дракона" в локацию "Победный свод", и на этом моменте вы с победой завершите игру. Если же вы пойдете через "Логово дракона" без "Вострого меча", то будете повержены опасным драконом и будете брошены на "Тёмное кладбище", где игра завершится с поражением. Игра останется на кладбище или в победном своде до тех пор, пока вы не перезапустите её. Общая карта игры показана на следующем изображении.

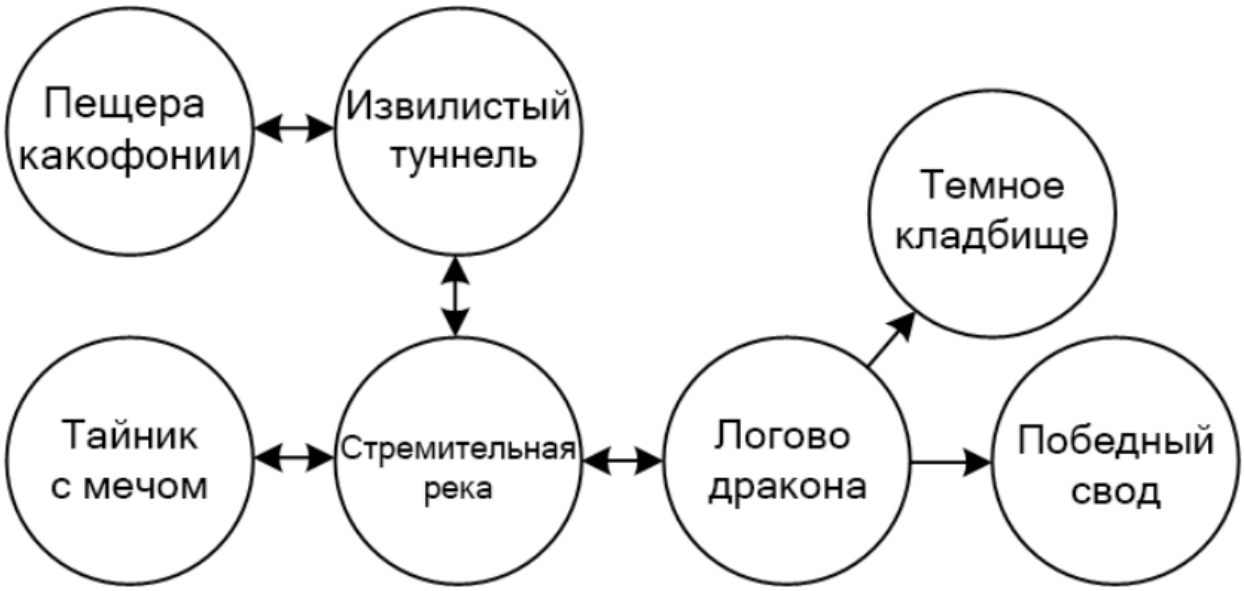

Рисунок 9

Код конечного автомата для движения:

```
module GameMove(
    input logic clk,
    input logic reset,
    input logic west, north, east, south,
    output logic [1:0] room_number;
);
    typedef enum logic [2:0] {S0, S1, S2, S3, S4, S5, S6} statetype;
    statetype state, nextstate;


    always_ff @(posedge clk, posedge reset)
       if (reset) state <= S0;
       else state <= nextstate;


    always_comb
       case (state)
       // cave
       S0: if (east) nextstate = S1;
           else nextstate = S0;

       // tonnel
       S1: if (west) nextstate = S1;
           else if (south) nextstate = S2;
           else nextstate = S1;

       // river
       S2: if (north) nextstate = S1;
           else if (east) nextstate = S2;
           else if (west) nextstate = S4;
           else nextstate = S1;

       // secret room
       S3: if (east) nextstate = S2;
           else nextstate = S3;

       // drogon
       S4: if (east) nextstate = S6;
           else if (north) nextstate = S5;
           else if (west) nextstate = S2;

       // trap
       S5: nextstate = S5;

       // final
       S6: nextstate = S6;

       default: nextstate = S0;
```

Тестовый код:

```
module testbench();
  logic clk, reset;
  logic W, N, S, E, y1, y2, y1_expected, y2_expected;
  logic [31:0] vectornum, errors;
  logic [5:0] testvectors[10000:0];
// инстанцировать тестируемое устройство
move dut(clk, reset, W, N, S, E, y1, y2);
// generate clock
always
 begin
   clk=1; #5; clk=0; #5;
 end
// на старте теста, загрузите вектора и запустите сброс
initial
 begin
   $readmemb("testWin.tv", testvectors);
   vectornum = 0; errors = 0; reset = 1; #22; reset = 0;
 end
// применение тестовых векторов по нарастающему фронту тактового сигнала
always @(posedge clk)
 begin
   #1; {W, N, S, E, y1_expected, y2_expected} =
       testvectors[vectornum];
 end
always @(negedge clk)
 if (~reset) begin // skip during reset
   if (y1 !== y1_expected | y2 !== y2_expected)
   begin
      $display("Error: inputs = %b", {W, N, S, E});
      $display(" outputs = %b %b (%b %b expected)", y1, y2, y1_expected, y2_expected);
      errors = errors + 1;
   end
   else if((y1 === y1_expected & y2 === y2_expected)&(y1 === 1'b1 & y2 === 1'b0))
   begin
      $display("WIN");
   end
   else if((y1 === y1_expected & y2 === y2_expected)&(y1 === 1'b0 & y2 === 1'b1))
   begin
      $display("DIE");
   end
 vectornum = vectornum + 1;
   if (testvectors[vectornum] === 6'bx) begin
      $display("%d tests completed with %d errors", vectornum, errors);
      //$display("OK");
      $stop;
   end
 end
endmodule
```

Выводы

В результате выполнения лабораторной работы №2 на практике были изучены основы работы с программным продуктом Quartus и ModelSim и применены существующие знания для разработки простых и не очень логических схем. Были спроектированы автоматы для решения поставленных задач, а также уже реализованы на языке описания архитектуры SystemVerilog в среде Intel Quartus. Тесты пройдены, схемы синтезированы, реализованные модули готовы к поставленным в условиях задачам. Работа к сдаче готова.

Литература

Харрис, Дэвид; Харрис, Сара «Цифровая схемотехника и архитектура компьютера. RISC-V»

Джон Хопкрофт, Раджив Мотвани, Джеффри Ульман «Введение в теорию автоматов, языков, вычислений»