Федеральное государственное бюджетное образовательное учреждение высшего образования
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
Институт №3 «Системы управления, информатика и электроэнергетика»
Кафедра 307 «Цифровые технологии и информационные системы»

ОТЧЕТ

О выполнении задания по предмету

«Программирование ПЛИС»

«Конечный автомат»

Выполнили студенты:
Петров В. А. гр. М3О-312Б-20
Куприянов И. А. гр. М3О-321Б-20
Смехов Г. М. гр. М3О-312Б-20
Елисеев И. О. гр. М3О-321Б-20
Попов Д. С. Гр. М3О-321Б-20
Проверил:
Старший преподаватель каф. 307:
Коробков М.А.

Москва 2023 г.

Цель работы

Разработать программу для устройства с ПЛИС «Altera Cyclone IV E», выполняющую функции конечного автомата «Поиск последовательности 101».

Теоретическая часть

Для выполнения данной лабораторной работы используется макетная плата RZ-EASYFPGA A2.2 Altera Cyclone IV FPGA, написание кода программы выполняется в среде программирования Quartus Prime, компиляция выполняется в Icarus Verilog.

Конечный автомат – это некоторая абстрактная модель, содержащая конечное число состояний чего-либо. Используется для представления и управления потоком выполнения каких-либо команд.

В данной работе рассмотрена разница между использованием автоматов Мура и Мили.

Автомат Мили – конечный автомат, выходная последовательность которого зависит от состояния автомата и входных сигналов. Это означает, что в графе состояний каждому ребру соответствует некоторое значение (выходной символ). В вершины графа автомата Мили записываются выходящие сигналы, а дугам графа приписывают условие перехода из одного состояния в другое, а также входящие сигналы.

Автомат Мура – конечный автомат, выходное значение сигнала в котором зависит лишь от текущего состояния данного автомата, и не зависит напрямую, в отличие от автомата Мили, от входных значений.

Задача конечного автомата «Поиск последовательности 101» звучит следующим образом: существует устройство в виде конечного автомата, которое анализирует последовательность двоичных сигналов. На вход поступает сигнал в виде нуля или единицы. По каждому тактовому импульсу на вход поступает следующий бит. При определении последовательности 101 на выходе устройство выдается единичный

сигнал, в противном случае ноль. Необходимо разработать автомат, определяющий, когда на выходе выдавать единичный сигнал. На вход А поступает значение на обрабатывающее устройство. На выходе У устанавливается логическая единица, когда обнаружено устройство 101. Конечный автомат «Поиск последовательности 101» — это пример конечного автомата, который может быть использован для поиска нужных последовательностей бит в непрерывном потоке данных.

Практическая часть

Реализация с помощью автомата Мура

Структура конечного автомата представлена на схеме ниже. Состояния, в которых он может находиться:

При XX0 – состояние 0;

При XX1 – состояние 1;

При X10 – состояние 2;

При 101 – состояние 3;

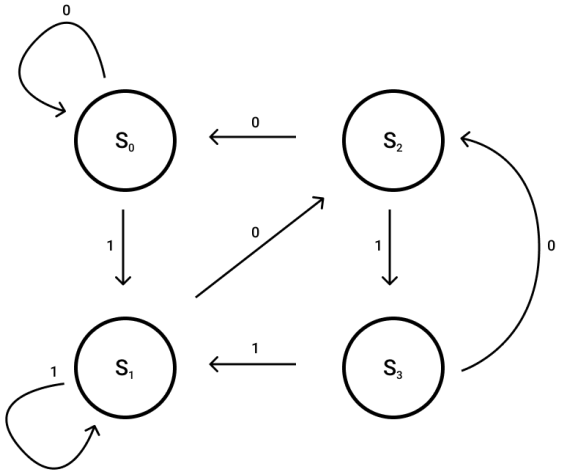

Рисунок 1 - схема состояний автомата Мура

Где:

Таблица 1 - таблица значений для состояний S_0, S_1, S_2 и S_2

S_0	0	0
S_1	0	1
S_2	1	0
S_3	1	1

Для реализации конечного автомата с помощью автомата Мура была написана следующая таблица истинности:

Таблица 2 - таблица истинности для автомата Мура

R	I	D_0’	D_1’	D_0	D_1
1	x	x	x	0	0
0	0	0	0	0	0
0	1	0	0	0	1
0	1	0	1	0	1
0	0	1	0	0	0
0	0	1	1	1	0
0	1	1	1	0	1
0	1	1	0	0	0
0	1	1	0	1	1
0	0	0	1	1	0

Следовательно, формулы для D_0 и D_1 выглядят следующим образом:

$$D_0=\left(\overline{R}I\,\overline{D_0^{'}}\,D_1^{'}\right)+\left(\overline{R}\,I\,D_0^{'}\,\overline{D_1^{'}}\right)+\left(\overline{R}\,\overline{I}\,D_0^{'}D_1^{'}\right)=\overline{R}\,D_1^{'}\overline{I}+\overline{R}\,D_0^{'}D_1^{'}\,I$$

$$D_1=\left(\overline{R}\,I\,\overline{D_0^{'}}\,\overline{D_1^{'}}\right)+\left(\overline{R}\,I\,\overline{D_0^{'}}\,D_1^{'}\right)+\left(\overline{R}\,I\,D_0^{'}\,\overline{D_1^{'}}\right)+\left(\overline{R}\,I\,D_0^{'}D_1^{'}\right)=\overline{R}\,I$$

Итоговая таблица состояний представлена ниже:

Таблица 3 - итоговая таблица состояний системы

D_0’	D_1’	y
0	0	0
0	1	0
1	0	0
1	1	1

$$y=D_0^{'}D_1^{'}$$

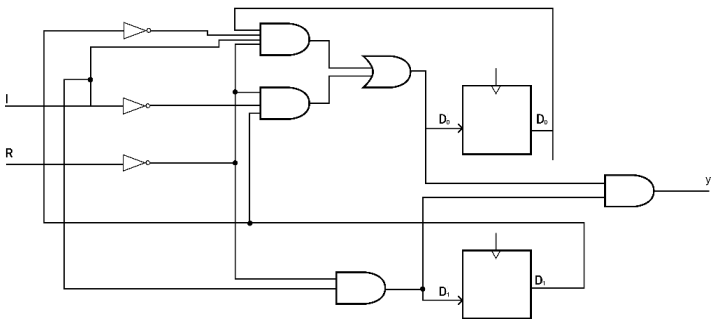

Рисунок 2 – итоговая структурная схема автомата Мура

Реализация с помощью автомата Мили

Схема состояний автомата Мили представлена ниже:

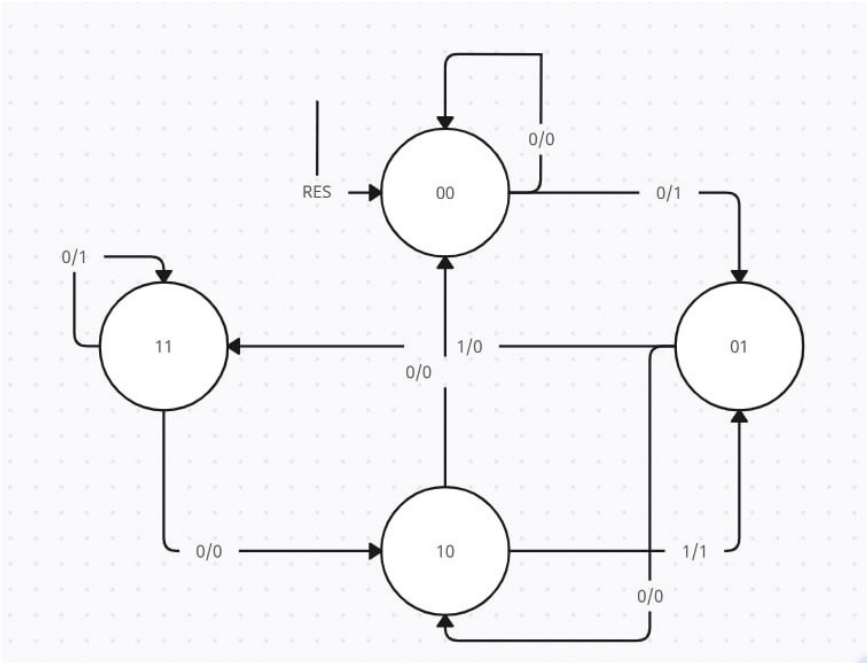

Рисунок 3 - схема состояний автомата Мили

Для реализации конечного автомата Мили была написана следующая таблица истинности:

Таблица 4 - таблица входных значений для автомата Мили

R	I	D_0’	D_1’	D_0	D_1
1	x	x	x	0	0
0	0	0	0	0	0
0	1	0	0	1	1
0	0	0	1	1	0
0	1	0	1	1	1
0	0	1	0	0	0
0	1	1	0	0	1
0	0	1	1	1	0
0	1	1	1	1	1

Где:

$$D_0=\left(\overline{R}I\,\overline{D_0'}\,\overline{D_1'}\right)+\left(\overline{R}\,\overline{I}\,\overline{D_0'}D_1'\right)+\left(\overline{R}I\,\overline{D_0'}D_1'\right)+\left(\overline{R}\,\overline{I}\,D_0'D_1'\right)=\overline{R}D_1'+\overline{R}I\,\overline{D_0'}$$

$$D_1=\left(\overline{R}\,I\,\overline{D_0^{'}}\,\overline{D_1^{'}}\right)+\left(\overline{R}\,I\,\overline{D_0^{'}}\,D_1^{'}\right)+\left(\overline{R}\,I\,D_0^{'}\,\overline{D_1^{'}}\right)+\left(\overline{R}\,I\,D_0^{'}\,D_1^{'}\right)=\overline{R}\,I$$

Таблица выходных значений представлена ниже:

Таблица 5 - таблица выходных значений для автомата Мили

I	$D_0^{'}$	$D_1^{'}$	y
x	0	0	0
x	0	1	0
1	1	0	1
0	1	0	0
x	1	1	0

Где:

$$y=I\,D_0^{'}\,\overline{D_1^{'}}$$

Следовательно, структурная схема автомата Мили выглядит следующим образом:

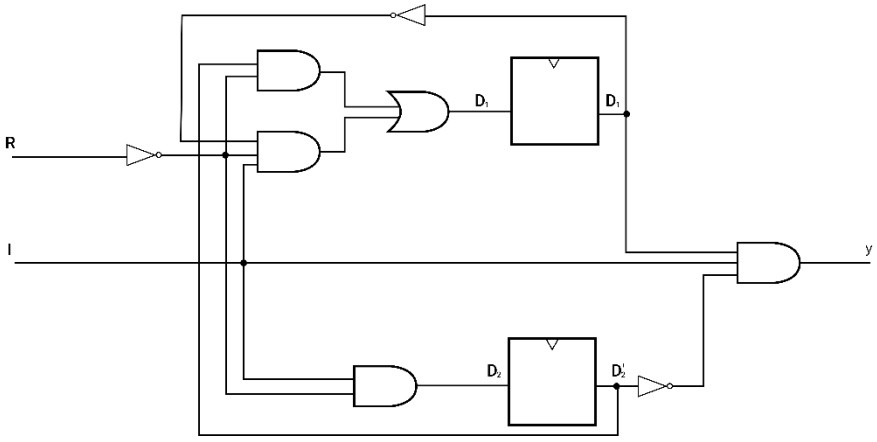

Рисунок 4 - итоговая структурная схема автомата Мили

Код программы

Реализация алгоритма «Поиск последовательности 101» через автомат Мура. Код программы представлен ниже:

```
module find (input logic  clk, rst, i, output logic y);
reg [1:0] statef, statez;
logic privf, privz;
logic u_3z, u_4z, u_2, ili_2z;
always_ff @ (posedge clk)
begin
      statez <= privz;
      statef <= privf;
end

always_comb
begin
u_4z <= statez & ~statef & i & ~rst;


u_3z <= ~i & ~rst & statef;

ili_2z <= u_4z | u_3z;

u_2 <= ~rst & i;

privf <= u_2;
privz <= ili_2z;
y <= privz & privf;
end
endmodule
```

Тестирование

На вход автомату подаётся последовательность чисел a и rst, данные для теста 1 и 2 представлены в таблице ниже:

Таблица 6 - данные для тестов 1 и 2

Clk	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
a	0	0	0	1	1	0	0	0	1	1	0	0	1	1	1	1	0	0	1	1	1
rst	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ожидаемый y	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	1	1

В 1 и 2 тестировании изменение сигнала *a* запаздывает относительно изменения *clk*, таким образом, изменение y запаздывает относительно изменения a на 1 такт.

Тест 1. Проверка функциональности

```
module Testbenc();
logic  clk, rst, a, y;
 find (clk, rst, a, y);


 initial
 begin
 rst = 1; clk = 1; a = 0; #100;
 clk = 0; #100;
 rst = 0;  clk = 1; #100;
 clk = 0;  a = 1; #100;
 clk = 1; #100;
 clk = 0;  a = 0; #100;
 clk = 1;  a = 0; #100;
 clk = 0;  a = 0; #100;
 clk = 1;  a = 1; #100;
 clk = 0;  a = 1; #100;
 clk = 1;  a = 0; #100;
 clk = 0;  a = 0; #100;
 clk = 1;  a = 1; #100;
 clk = 0;  a = 1; #100;
 clk = 1;  a = 1; #100;
 clk = 0;  a = 1; #100;
 clk = 1;  a = 0; #100;
 clk = 0;  a = 0; #100;
 clk = 1;  a = 1; #100;
 clk = 0;  a = 1; #100;
 clk = 1;  a = 0; #100;
 clk = 0;  a = 0; #100;
 clk = 1;  a = 1; #100;
```

```
    clk = 0;   a = 1; #100;
    clk = 1;   a = 1; #100;
    clk = 0;   a = 1; #100;
    clk = 1;   a = 1; #100;
    clk = 0;   a = 1; #100;
    clk = 1;   a = 0; #100;
    clk = 0;   a = 0; #100;
    clk = 1;   a = 0; #100;
    clk = 0;   a = 0; #100;
    clk = 1;
    end
  endmodule
```

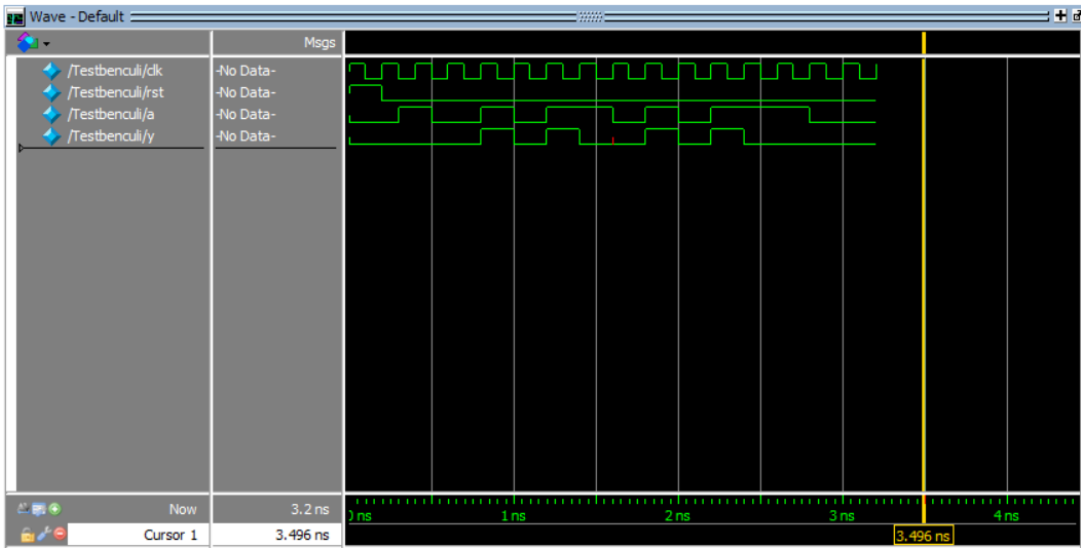

Рисунок 5 - тестирование кода программы в Modelsim

Тест 2. Проверка системы тактирования

```
module Testbenc();
logic  clk =0, rst=0, a=0, y=0;

      ulitka uli (clk, rst, a, y);

            always
         begin
           clk = clk ^ 1; #100;
         end

            initial
            begin
                  rst = 1; a = 0; #200;
      rst = 0; #100;
      a = 1; #100;
      a = 1; #100;
      a = 0; #100;
      a = 0; #100;
      a = 0; #100;
      a = 0; #100;
      a = 1; #100;
      a = 1; #100;
```

```
        a = 0; #100;
        a = 0; #100;
        a = 1; #100;
        a = 1; #100;
        a = 1; #100;
        a = 1; #100;
        a = 0; #100;
        a = 0; #100;
        a = 1; #100;
        a = 1; #100;
        a = 0; #100;
        a = 0; #100;
        a = 1; #100;
        a = 1; #100;
        a = 1; #100;
        a = 1; #100;
        a = 1; #100;
        a = 1; #100;
        a = 0; #100;
           end
endmodule
```

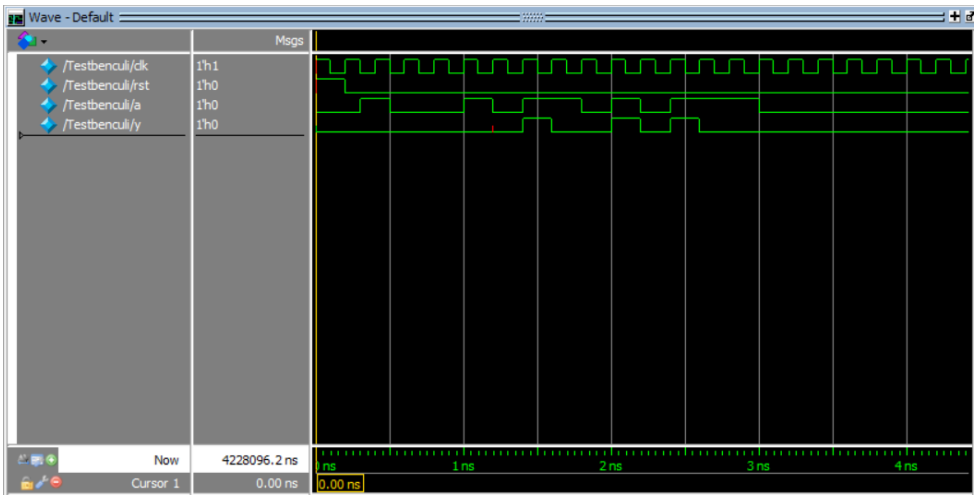

Рисунок 6 - - тестирование кода программы в Modelsim

Тест 3. Выгрузка из файла тестовых векторов

В данном тесте происходит выгрузка векторов из файла example.txt, содержание представлено ниже:

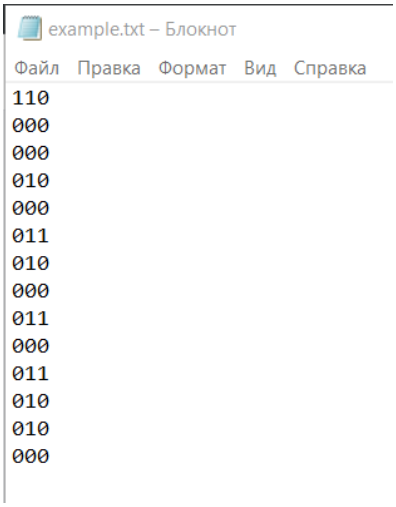

Рисунок 7 - файл example.txt, содержащий векторы

Модуль тестирования представлен ниже:

```
module Testbenc();
 logic clk, reset1;
 logic none, reset, a, y, yexpected;
 logic [31:0] vectornum, errors;
 logic [3:0] testvectors[10000:0];


 ulitka dut(clk,reset, a, y);
 always
       begin
             clk = 0; #5; clk = 1; #5;
       end


initial
 begin
       $readmemb("C:/Users/HUAWEI/Desktop/example.txt", testvectors);
       vectornum = 0; errors = 0;
       reset1 = 1; #27; reset1 = 0;
 end

always @(negedge clk)
 begin
    {none, reset, a, yexpected} = testvectors[vectornum];
       $display("IIInputs = %b", {reset, a, yexpected});
 end
always @(posedge clk)
 if (~reset1) begin // skip during reset
       if (y !== yexpected) begin // check result
             $display("Error: inputs = %b", {reset, a});
             $display(" outputs = %b (%b expected)", y, yexpected);
             errors = errors + 1;
       end
       vectornum = vectornum + 1;
       if (testvectors[vectornum] === 4'bx) begin
```

```
                $display("%d tests completed with %d errors", vectornum,
errors);
                $finish;
            end
        end
    endmodule
```

Результаты тестирования:

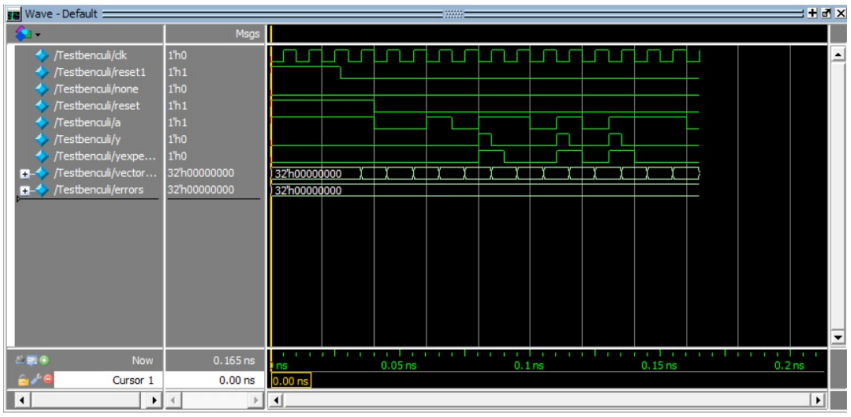

Рисунок 8 - результаты тестирования кода в Modelsim

Список считанных векторов и результат тестирования:

```
# IIInputs = 000
# IIInputs = 010
# IIInputs = 001
# IIInputs = 010
# IIInputs = 010
# IIInputs = 011
# IIInputs = 000
#          14 tests completed with          0 errors
# ** Note: $finish    : D:/intelFPGA_lite/21.1/Testbenculi.sv(36)
#    Time: 160 ps  Iteration: 1  Instance: /Testbenculi
```

Рисунок 9 - список считанных векторов и результат тестирования

Вывод

В результате выполнения лабораторной работы написана программа для ПЛИС «Altera Cyclone IV», выполняющая функции конечного автомата «Поиск последовательности 101». Составлены автоматы Мили и Мура, а также структурные схемы и таблицы истинности для них. Отмечено, что в автомате Мура изменение состояния происходит на следующем такте после ввода данных. Программа протестирована: выполнена проверка функциональности, проверка системы тактирования, а также проверка системы путем выгрузки тестовых векторов из файла.