**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной Техники**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Верификация и тестирование встраиваемых систем»**

**ТЕМА: Описание цифровых автоматов**

**Вариант 1**

|  |  |  |  |
| --- | --- | --- | --- |
| Студенты гр. 6309 | |  | Васин А. М. |
| Преподаватель | |  | Мурсаев А. Х. |
|  |
|  |  |  |  |

Санкт-Петербург

2021

# Цель работы

Исследовать виды автоматов и общую структуру программ с автоматами, разработать программу для тестирования автомата, включающую исследуемый автомат и генератор тестирующих воздействий.

Автомат, который необходимо реализовать представлен ниже.

|  |  |  |  |
| --- | --- | --- | --- |
| Исходное состояние и вход | | Выходное состояние и выход | |
| S(t) | x(t) | S(t+1) | Y(t+1) |
| S0 | 2’b00 | S0 | Y0 |
| S0 | x1 | S1 | Y0 |
| S0 | x2 | S3 | Y2 |
| S1 | 2’b00 | S1 | Y0 |
| S1 | x1 | S2 | Y0 |
| S1 | x2 | S0 | Y0 |
| S2 | 2’b00 | S2 | Y0 |
| S2 | x1 | S3 | Y0 |
| S2 | x2 | S1 | Y0 |
| S3 | 2’b00 | S3 | Y0 |
| S3 | x1 | S0 | Y1 |
| S3 | x2 | S2 | Y0 |

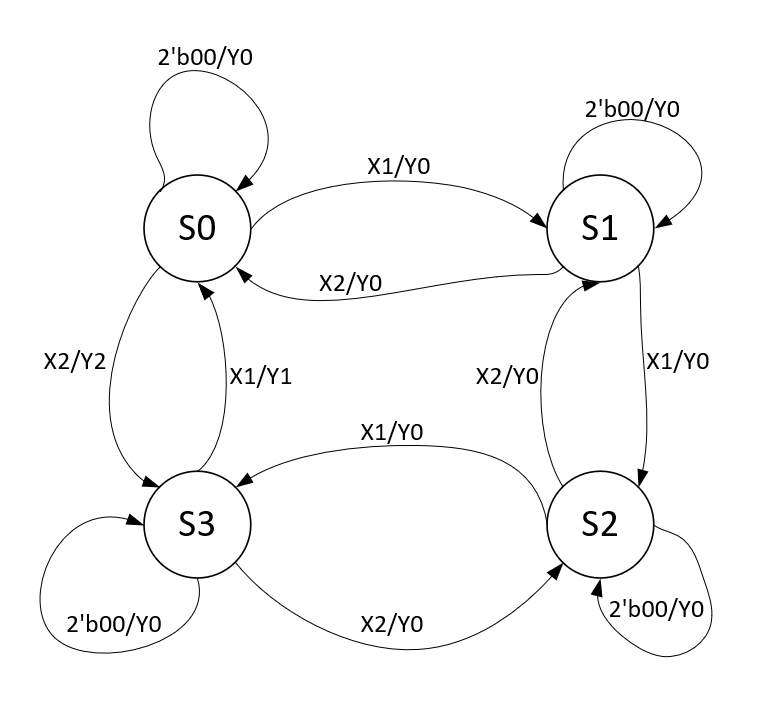
Кодирование состояний:

Кодирование входных данных:

Кодирование выходных данных:

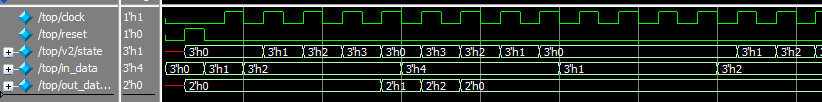
# Ход работы

1. По заданию нарисовали схему автомата:



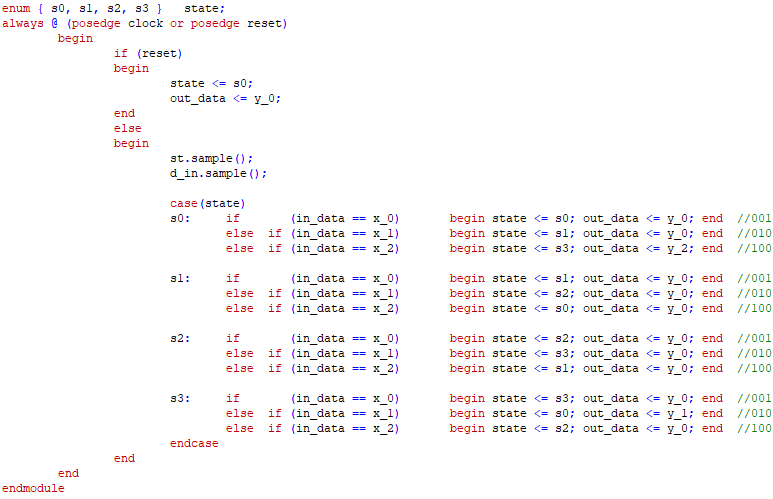
1. Описали автомат в обычном виде.

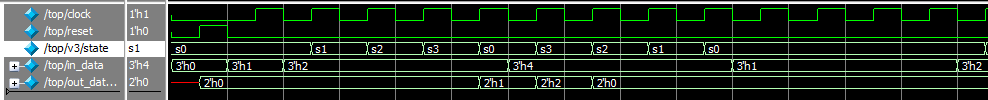




Как мы видим, на рисунке выше, автомат, разработанный и описанный, работает верно: out\_data соответствует вышеописанной таблице.

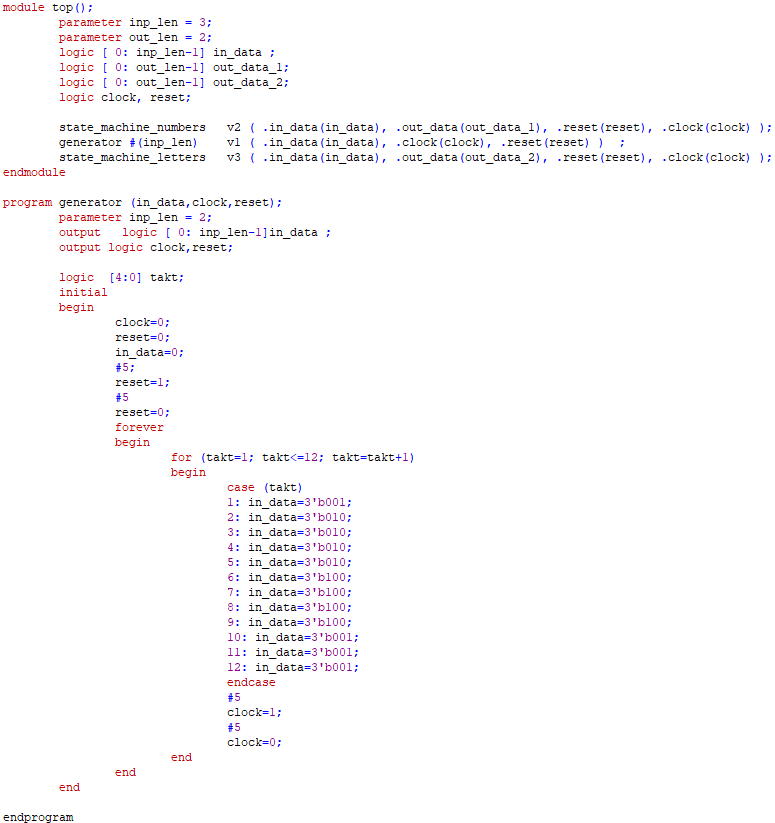
1. Описали автомат в символьном виде.



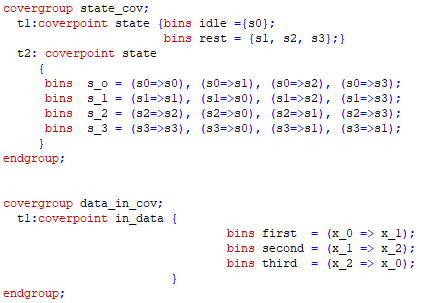


Как мы видим, на рисунке выше, автомат, разработанный и описанный, работает верно: out\_data такой же как и при описании автомата в обычном виде.

1. Создали тестирующий модуль

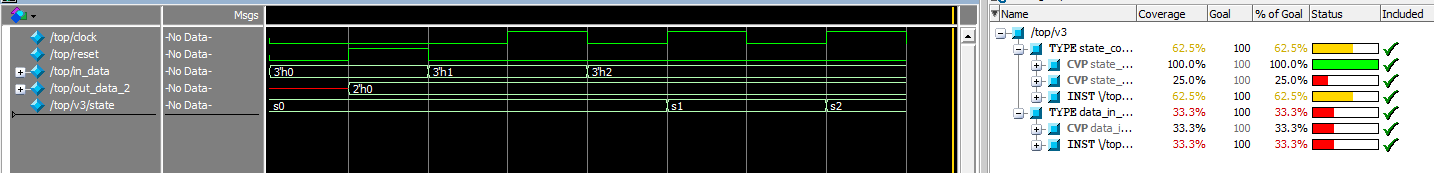


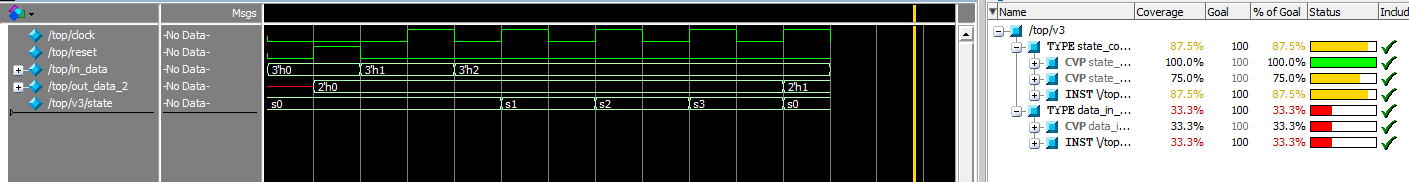
1. Сформировали контроль покрытий состояний и входных данных

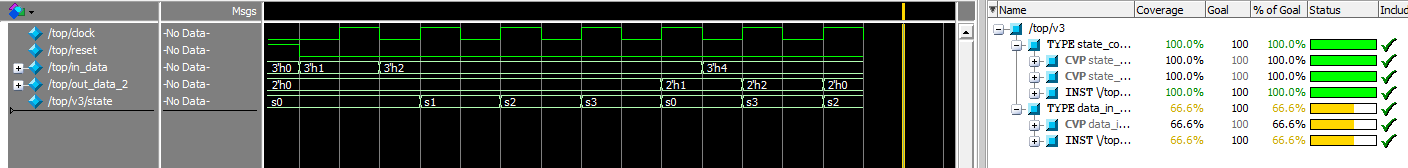


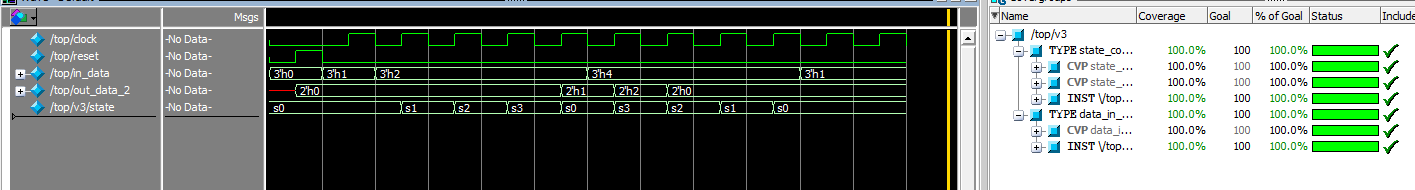
В результате работы тестовой программы и контроля покрытий получили следующие результаты:

Симуляция с шагами clock:

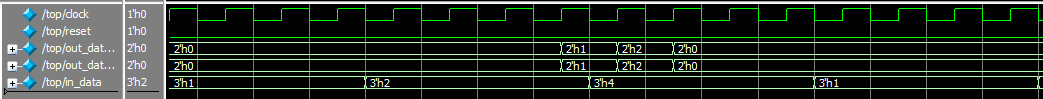




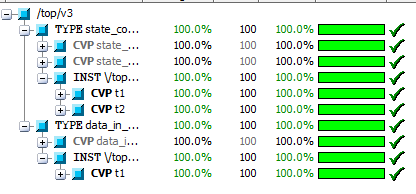


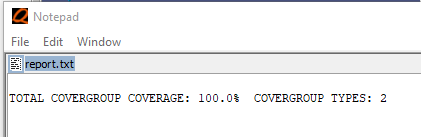


Полное покрытие всех групп было получено за 11 тактовых сигналов.

****

Данные на выходе при соответствующих данных на входе такие же как мы и ожидали.

****

****

Покрытие полное, о чем говорит сгенерированный отчет и вкладка о покрытиях в среде QuestaSim.

Вывод: в процессе выполнения лабораторной работы мы изучили общую структуру программ с автоматами, разработали программу для тестирования автомата, включающую исследуемый автомат и генератор тестирующих воздействий на языке VerilogHDL. Также убедились в полном покрытии тестовых входных сигналов и состояний автомата.

# Приложение А

## TestBench

**module top();**

parameter inp\_len = 3;

parameter out\_len = 2;

logic [ 0: inp\_len-1] in\_data ;

logic [ 0: out\_len-1] out\_data\_1;

logic [ 0: out\_len-1] out\_data\_2;

logic clock, reset;

state\_machine\_numbers v2 ( .in\_data(in\_data), .out\_data(out\_data\_1), .reset(reset), .clock(clock) );

generator #(inp\_len) v1 ( .in\_data(in\_data), .clock(clock), .reset(reset) ) ;

state\_machine\_letters v3 ( .in\_data(in\_data), .out\_data(out\_data\_2), .reset(reset), .clock(clock) );

**endmodule**

**program generator** (in\_data,clock,reset);

parameter inp\_len = 2;

output logic [ 0: inp\_len-1]in\_data ;

output logic clock,reset;

logic [4:0] takt;

initial

begin

clock=0;

reset=0;

in\_data=0;

#5;reset=1;

#5 reset=0;

forever

begin

for (takt=1; takt<=12; takt=takt+1)

begin

case (takt)

1: in\_data=3'b001;

2: in\_data=3'b010;

3: in\_data=3'b010;

4: in\_data=3'b010;

5: in\_data=3'b010;

6: in\_data=3'b100;

7: in\_data=3'b100;

8: in\_data=3'b100;

9: in\_data=3'b100;

10: in\_data=3'b001;

11: in\_data=3'b001;

12: in\_data=3'b001;

endcase

#5 clock=1;

#5 clock=0;

end

end

end

**endprogram**

## State machines

**module state\_machine\_numbers (in\_data, reset, clock, out\_data);**

parameter state\_len = 3;

parameter inp\_len = 3;

parameter out\_len = 2;

input wire [inp\_len - 1:0] in\_data;

input wire reset;

input wire clock;

output reg [out\_len - 1:0] out\_data;

reg [state\_len - 1:0] state;

parameter y\_0 = 2'b00;

parameter y\_1 = 2'b01;

parameter y\_2 = 2'b10;

parameter x\_0 = 3'b001;

parameter x\_1 = 3'b010;

parameter x\_2 = 3'b100;

always @ (posedge clock or posedge reset)

begin

if (reset)

begin

state <= 2'b00;

out\_data <= 2'b00;

end

else

begin

case(state)

2'b00: if (in\_data == x\_0) begin state <= 2'b00; out\_data <= y\_0; end //001

else if (in\_data == x\_1) begin state <= 2'b01; out\_data <= y\_0; end //010

else if (in\_data == x\_2) begin state <= 2'b11; out\_data <= y\_2; end //100

2'b01: if (in\_data == x\_0) begin state <= 2'b01; out\_data <= y\_0; end //001

else if (in\_data == x\_1) begin state <= 2'b10; out\_data <= y\_0; end //010

else if (in\_data == x\_2) begin state <= 2'b00; out\_data <= y\_0; end //100

2'b10: if (in\_data == x\_0) begin state <= 2'b10; out\_data <= y\_0; end //001

else if (in\_data == x\_1) begin state <= 2'b11; out\_data <= y\_0; end //010

else if (in\_data == x\_2) begin state <= 2'b01; out\_data <= y\_0; end //100

2'b11: if (in\_data == x\_0) begin state <= 2'b11; out\_data <= y\_0; end //001

else if (in\_data == x\_1) begin state <= 2'b00; out\_data <= y\_1; end //010

else if (in\_data == x\_2) begin state <= 2'b10; out\_data <= y\_0; end //100

endcase

end

end

**endmodule**

**module state\_machine\_letters (in\_data, reset, clock, out\_data);**

parameter state\_len = 3;

parameter inp\_len = 3;

parameter out\_len = 2;

input wire [inp\_len - 1:0] in\_data;

input wire reset;

input wire clock;

output reg [out\_len - 1:0] out\_data;

enum { s0, s1, s2, s3 } state;

parameter y\_0 = 2'b00;

parameter y\_1 = 2'b01;

parameter y\_2 = 2'b10;

parameter x\_0 = 3'b001;

parameter x\_1 = 3'b010;

parameter x\_2 = 3'b100;

covergroup state\_cov;

t1:coverpoint state {bins idle ={s0};

bins rest = {s1, s2, s3};}

t2: coverpoint state

{

bins s\_o = (s0=>s0), (s0=>s1), (s0=>s2), (s0=>s3);

bins s\_1 = (s1=>s1), (s1=>s0), (s1=>s2), (s1=>s3);

bins s\_2 = (s2=>s2), (s2=>s0), (s2=>s1), (s2=>s3);

bins s\_3 = (s3=>s3), (s3=>s0), (s3=>s1), (s3=>s1);

}

endgroup;

covergroup data\_in\_cov;

t1:coverpoint in\_data {

bins first = (x\_0 => x\_1);

bins second = (x\_1 => x\_2);

bins third = (x\_2 => x\_0);

}

endgroup;

state\_cov st;

data\_in\_cov d\_in;

initial

begin

st = new;

d\_in = new;

end

always @ (posedge clock or posedge reset)

begin

if (reset)

begin

state <= s0;

out\_data <= y\_0;

end

else

begin

st.sample();

d\_in.sample();

case(state)

s0: if (in\_data == x\_0) begin state <= s0; out\_data <= y\_0; end //001

else if (in\_data == x\_1) begin state <= s1; out\_data <= y\_0; end //010

else if (in\_data == x\_2) begin state <= s3; out\_data <= y\_2; end //100

s1: if (in\_data == x\_0) begin state <= s1; out\_data <= y\_0; end //001

else if (in\_data == x\_1) begin state <= s2; out\_data <= y\_0; end //010

else if (in\_data == x\_2) begin state <= s0; out\_data <= y\_0; end //100

s2: if (in\_data == x\_0) begin state <= s2; out\_data <= y\_0; end //001

else if (in\_data == x\_1) begin state <= s3; out\_data <= y\_0; end //010

else if (in\_data == x\_2) begin state <= s1; out\_data <= y\_0; end //100

s3: if (in\_data == x\_0) begin state <= s3; out\_data <= y\_0; end //001

else if (in\_data == x\_1) begin state <= s0; out\_data <= y\_1; end //010

else if (in\_data == x\_2) begin state <= s2; out\_data <= y\_0; end //100

endcase

end

end

**endmodule**