**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной Техники**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Языки проектирования аппаратуры»**

**ТЕМА: Описание цифровых автоматов**

**Вариант 4**

|  |  |  |  |
| --- | --- | --- | --- |
| Студенты гр. 6307 | |  | Васин А. М. |
|  |  |  | Кичерова А. Д. |
|  |  |  |
|  |  |  | Ладыженский Р. С. |
|  |  |  |
| Преподаватель | |  | Мурсаев А. Х. |
|  |
|  |  |  |  |

Санкт-Петербург

2020

# Цель работы

Разработать описание цифрового автомата в отдельном модуле. Разработать генератор тестового воздействия на базе синтаксиса языка VerilogHDL.

Автомат, который необходимо реализовать:

|  |  |  |  |
| --- | --- | --- | --- |
| 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 | S0 | Y0 |
| S0 | x2 | S1 | Y1 |
| S0 | x3 | S2 | Y1 |
| S1 | 2’b00 | S1 | Y1 |
| S1 | x1 | S0 | Y0 |
| S1 | x2 | S2 | Y1 |
| S1 | x3 | S2 | Y0 |
| S2 | 2’b00 | S2 | Y1 |
| S2 | x1 | S1 | Y0 |
| S2 | x2 | S1 | Y0 |
| S2 | x3 | S0 | Y1 |

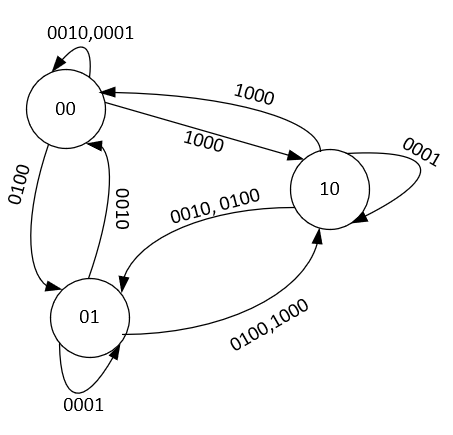
Кодирование состояний:

Кодирование входных данных:

Кодирование выходных данных:

# Ход работы

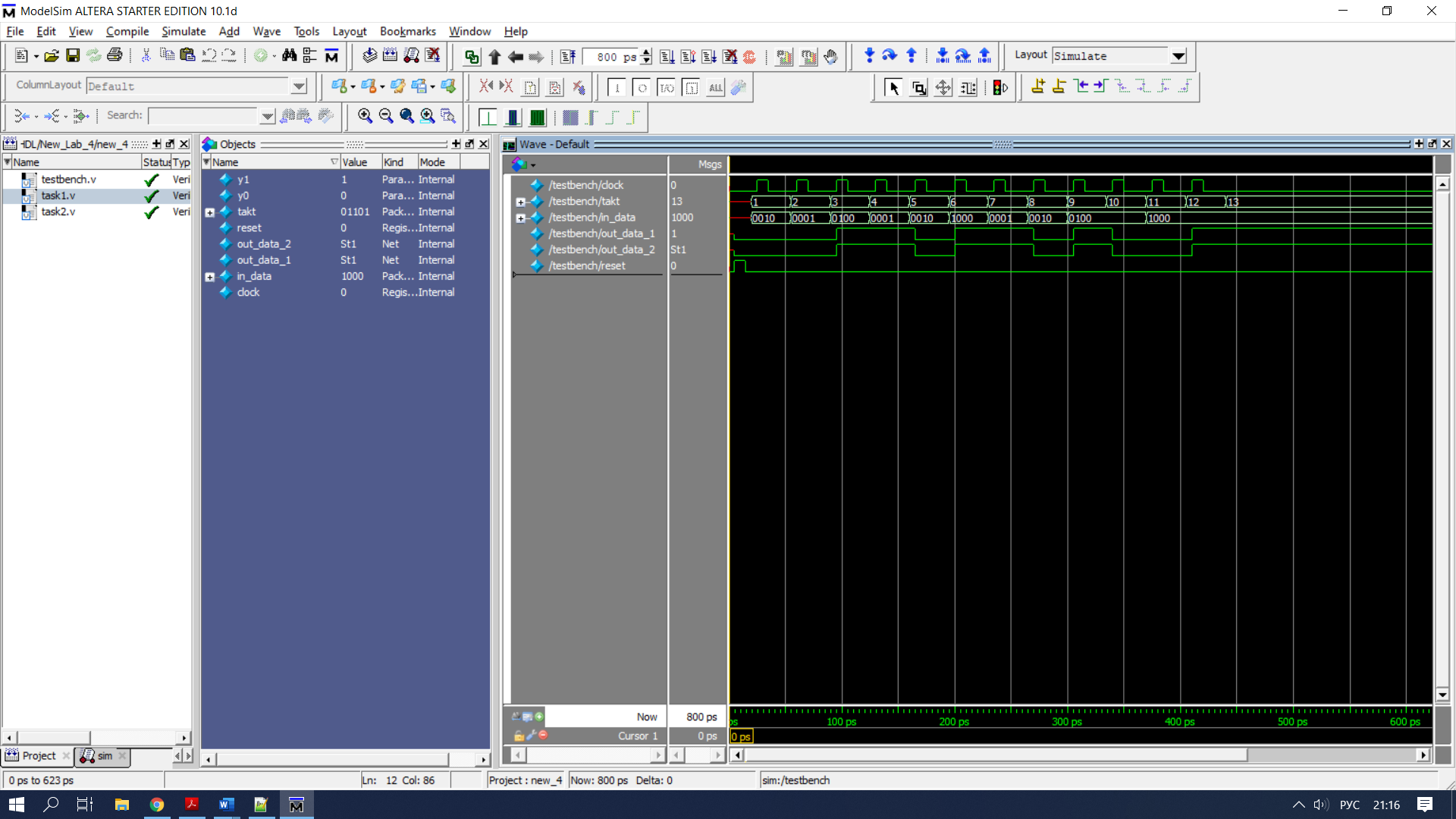
1. По заданию нарисовали схему автомата:



1. Создали последовательность обхода автомата с ожидаемыми результатами.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Входной код | Ожидаемое состояние | Ожидаемый выходной код |
|  | - | 00 | - |
| 1 | 0010 | 00 | 0 |
| 2 | 0001 | 00 | 0 |
| 3 | 0100 | 01 | 1 |
| 4 | 0001 | 01 | 1 |
| 5 | 0010 | 00 | 0 |
| 6 | 1000 | 10 | 1 |
| 7 | 0001 | 10 | 1 |
| 8 | 0010 | 01 | 0 |
| 9 | 0100 | 10 | 1 |
| 10 | 0100 | 01 | 0 |
| 11 | 1000 | 10 | 0 |
| 12 | 1000 | 00 | 1 |

1. Создали новый проект, в котором создали два файла описания автомата (Приложение А): в обычном виде и в символьном виде. Также добавили в проект файл тестирования автомата. Результаты работы программы представлены ниже.



Как мы видим, на рисунке выше, автомат, разработанный и описанный, работает верно: out\_data\_1 (стандартное описание автомата) и out\_data\_2 (символьное описание автомата) идентичны и соответствуют вышеописанной таблице.

**Вывод**: в ходе выполнения данной работы мы спроектировали конечный автомат Мили в двух видах: в стандартном виде и в символьном виде. Сформировали тестовую последовательность полного обхода автомата и убедились в правильности работы автомата.

# Контрольные вопросы

1. **Какие преимущества дает объявление состояния по имени. Как это сделать к классическом Verilog.**

Объявление состояний по имени позволяет получить большую наглядность кода и лучшее взаимопонимания разработчиков, тем самым минимизировать количество ошибок, связанных с неверным пониманием процесса или банальной опечаткой. Иногда так же полезно присваивание собственных имен входным и выходным кодовым комбинациям. Полезно использовать имена так или иначе отражающие смысл объявляемых значений.

«Классический» Verilog не предполагает перечислимого типа данных. Тем не менее, проблему присвоения собственных имен значениям данных можно решить, используя соответствующие декларации. Предпочтительно использовать конструкцию **parameter,** потому что тогда легко изменить кодирование при включении модуля в иерархический проект.

К тому же, использование символьных имен состояний позволяет переложить задачу оптимизации на синтезатор, в таком случае конкретные коды состояний будут выбраны синтезатором автоматически в соответствии с существующими задачами и ограничениями. Такой механизм реализован в САПР Quartus II.

Так же хочется отметить, что разработчики языка добавили полноценную поддержку символьных имен состояний в языке SystemVerilog посредствам перечислимых типов.

1. **Как изменится текст описания спроектированного вами автомата, если потребуются выходы с иными временными характеристиками (например, добавление выходов, связанных только с состоянием для автоматов Мили)?**

Благодаря гибкому описанию автомата в языке Verilog в общем случае для добавления данного функционала потребуется добавить один **always** блок.

При добавлении выходов, связанных только с состоянием для автомата Мили в список чувствительности данного блока будет включено только текущее состояние, от которого будут комбинаторно зависеть выходные значения.

В случае добавления в автомат Мура выходов, связанных с конкретным переходом в список чувствительности необходимо поместить текущее состояние, сигналы, инициирующие переход и сигнал синхронизации. Выходные значения будут изменяться по сигналу синхронизации.

В любом случае для данной ситуации не потребуется изменять уже спроектированные блоки.

# Приложение А

**module testbench;**

parameter

y0=1'b0,

y1=1'b1;

reg [3:0] in\_data;

reg reset, clock;

reg [4:0] takt;

wire out\_data\_1;

wire out\_data\_2;

task1 #(2,4,1) v1(.in\_data(in\_data),.data\_out(out\_data\_1),.reset(reset), .clock(clock));

task2 #(2,4,1, y0,y1) v2(.in\_data(in\_data),.data\_out(out\_data\_2),.reset(reset), .clock(clock));

initial

begin

clock='b0;

reset='b0;

#5 reset='b1;

#10 reset='b0;

#5;

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

begin

case (takt)

1: in\_data=4'b0010;

2: in\_data=4'b0001;

3: in\_data=4'b0100;

4: in\_data=4'b0001;

5: in\_data=4'b0010;

6: in\_data=4'b1000;

7: in\_data=4'b0001;

8: in\_data=4'b0010;

9: in\_data=4'b0100;

10:in\_data=4'b0100;

11:in\_data=4'b1000;

12:in\_data=4'b1000;

endcase

#5 clock='b1;

#10 clock='b0;

#20;

end // for

end

**endmodule**

**module task1 (in\_data, reset, clock, data\_out);**

parameter state\_len = 2;

parameter inp\_len = 4;

parameter out\_len = 1;

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

input wire reset;

input wire clock;

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

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

always @ (posedge clock or posedge reset)

begin

if (reset)

begin

state <= 2'b00;

data\_out <= 1'b0;

end

else

case(state)

2'b00: if (in\_data[0]) begin state <= 2'b00; data\_out <= 1'b0; end //0001

else if (in\_data[1]) begin state <= 2'b00; data\_out <= 1'b0; end //0010

else if (in\_data[2]) begin state <= 2'b01; data\_out <= 1'b1; end //0100

else begin state <= 2'b10; data\_out <= 1'b1; end //1000

2'b01: if (in\_data[0]) begin state <= 2'b01; data\_out <= 1'b1; end //0001

else if (in\_data[1]) begin state <= 2'b00; data\_out <= 1'b0; end //0010

else if (in\_data[2]) begin state <= 2'b10; data\_out <= 1'b1; end //0100

else begin state <= 2'b10; data\_out <= 1'b0; end //1000

2'b10: if (in\_data[0]) begin state <= 2'b10; data\_out <= 1'b1; end //0001

else if (in\_data[3]) begin state <= 2'b00; data\_out <= 1'b1; end //1000

else begin state <= 2'b01; data\_out <= 1'b0; end //0100 or 0010

endcase

end //always

**endmodule**

**module task2 (in\_data, reset, clock, data\_out);**

parameter state\_len = 2;

parameter inp\_len = 4;

parameter out\_len = 1;

parameter [out\_len-1:0]

y0 = 1'b0,

y1 = 1'b1;

parameter [state\_len-1:0]

s0 = 2'b00,

s1 = 2'b01,

s2 = 2'b10;

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

input wire reset;

input wire clock;

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

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

always @ (posedge clock or posedge reset)

begin

if (reset)

begin

state <= s0;

data\_out <= y0;

end

else

case(state)

s0: if (in\_data[0]) begin state <= s0; data\_out <= y0; end

else if (in\_data[1]) begin state <= s0; data\_out <= y0; end

else if (in\_data[2]) begin state <= s1; data\_out <= y1; end

else begin state <= s2; data\_out <= y1; end

s1: if (in\_data[0]) begin state <= s1; data\_out <= y1; end

else if (in\_data[1]) begin state <= s0; data\_out <= y0; end

else if (in\_data[2]) begin state <= s2; data\_out <= y1; end

else begin state <= s2; data\_out <= y0; end

s2: if (in\_data[0]) begin state <= s2; data\_out <= y1; end

else if (in\_data[3]) begin state <= s0; data\_out <= y1; end

else begin state <= s1; data\_out <= y0; end

endcase

end //always

**endmodule**