Національний технічний університет України "КПІ"
Інститут прикладного системного аналізу
Кафедра Системного проектування

Розрахунково-графічна робота
з курсу: " Технології комп’ютерного проектування "
на тему:
"Проектування цифрових автоматів"

виконала:
студентка 3 курсу
групи ДА- 42
Кулівник Крістіна

перевірила:
доц. Гіоргізова-Гай В. Ш.

Київ- 2017

Завдання:

Спроектувати цифровий автомат у вигляді асинхронного автомата Мура з проведенням аналізу результатів, отриманих після кожного етапу проектування.

Таблиця 1. Варіанти завдань для груп ДА-Х1

Номер варианта задания	Последовательность входных сигналов X1X0	Последовательность выходных сигналов Y
12	12	4

Таблиця 2. Варіанти вхідних сигналів X1X0

Номер варіанту	Номер такту															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
12	00	10	11	10	11	10	11	10	00	10	00	10	00	10	00	01

Таблиця 3. Варіанти виходу Y

Номер варіанту	Номер такту															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0

Первинна таблиця переходів :

Q	**Y**	**X2 X1**			
		00	***01***	***10***	***11***
0	***0***	(0)	-	1	-
1	***0***	-	-	(1)	2
2	***1***	-	-	3	(2)
3	***1***	-	-	(3)	4
4	***0***	-	-	5	(4)
5	***0***	-	-	(5)	6
6	***0***	-	-	7	(6)
7	***0***	8	-	(7)	-
8	***0***	(8)	-	9	-
9	***0***	10	-	(9)	-
10	***0***	(10)	-	11	-
11	***0***	12	-	(11)	-
12	***0***	(12)	-	13	-
13	***1***	14	-	(13)	-
14	***0***	(14)	15	-	-
15	***0***	(!)0	(15)	-	-

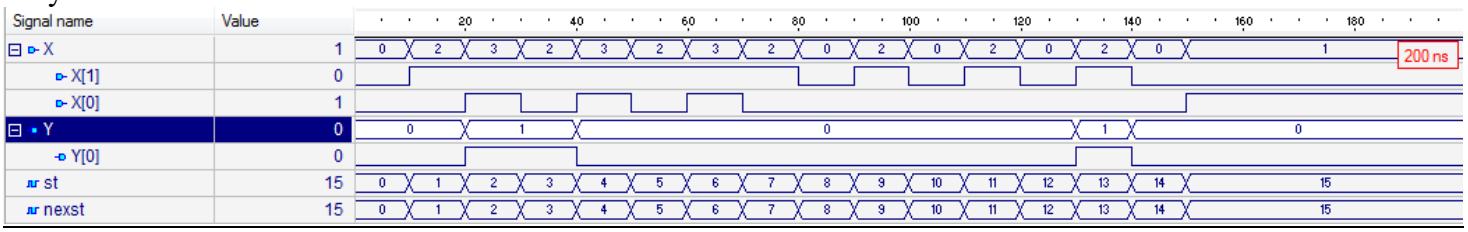
Перевірімо отриману таблицю програмно:

```
        library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity a1ent is
generic (TZ: time :=0ns);
 port(
         X: in bit_vector (1 downto 0);
         Y: out bit_vector(0 downto 0)
        );
end a1ent;

architecture a1arc of a1ent is
-- bit_vector to integer
function vecint (vec1: bit_vector)
return integer is
variable retval:integer:=0;
begin
 for i in vec1'length-1 downto 1 loop
 if (vec1(i)='1') then retval:=(retval+1)*2;
 else retval:= retval*2; end if;
 end loop;
 if vec1(0)='1' then retval:=retval+1;
 else null; end if;
 return retval;
 end vecint;
type stab is array (0 to 15, 0 to 3) of integer;
constant tab_st: stab :=(
 (0,16,1,16),
(16,16,1,2),
(16,16,3,2),
(16,16,3,4),
(16,16,5,4),
(16,16,5,6),
(16,16,7,6),
(8,16,7,16),
(8,16,9,16),
(10,16,9,16),
(10,16,11,16),
(12,16,11,16),
(12,16,13,16),
(14,16,13,16),
(14,15,16,16),
(0,15,16,16));

type outtab is array (0 to 15) of bit_vector(0 downto 0);
constant tab_y : outtab :=("0","0","1","1","0","0","0","0", "0","0","0","0","0","1","0","0");
signal st,nexst: integer:=0;
begin
 process begin
 wait on X, st;
 if st=16 then null;
 else
 nexst<=tab_st(st,vecint(X));
 Y<=transport tab_y(st)after TZ;
```

```
 end if;
 end process;
 st<= transport nexst after TZ;
end a1arc;
```

00 0ns, 10 10ns, 11 20ns, 10 30ns, 11 40ns, 10 50ns, 11 60ns, 10 70ns, 00 80ns, 10 90ns, 00 100ns, 10 110ns, 00 120ns, 10 130ns, 00 140ns, 01 150ns

Результат виконання:



Мінімізація кількості внутрішніх станів автомата

Як правило, первинна таблиця переходів містить надлишкові стани. Тому наступним етапом проектування є мінімізація таблиці переходів. Процедура мінімізації складається з наступних кроків:

- знаходження множини сумісних пар станів автомата;
- знаходження максимально сумісної множини станів автомата;
- отримання мінімального замкнутого покриття станів автомата;
- отримання таблиці переходів мінімального автомата.

Несумісність двох станів таблиці переходів визначається наступною лемою:

Два стани q_i *і* q_j *таблиці переходів несумісні тоді і тільки тоді, коли* q_i *і* q_j *несумісні по виходу або коли для деякого стану входу* I_k *несумісні* $N(q_i, I_k)$ *і* $N(q_j, I_k)$.

Два стани q_i *і* q_j *сумісні по виходу, якщо* $Z(q_i, I_k) = Z(q_j, I_k)$ *для усіх* I_k, *для яких обидва стани визначено.*

Тут $Z(q_i, I_k)$ - вихідний сигнал, що отримується у тому випадку, коли до автомата, що знаходиться в стані q_i, прикладається вхідний сигнал I_k, а $N(q_i, I_k)$ наступний стан, в який переходить автомат, що знаходиться в стані q_j, під впливом стану входу I_k .Процедура знаходження несумісних пар станів полягає в наступному:

- виписати усі пари, несумісні по виходах;
- приписати до них усі пари, що породжують вже виписані пари;
- повторювати цей процес до тих пір, поки не перестануть з'являтися нові пари.

Выпишем группы с устойчивым состоянием в одном столбце:

0	15	1	2
8		3	4
10		5	6
12		7	
14		9	
		11	
		13	

Второе условие (одинаковые выходы):

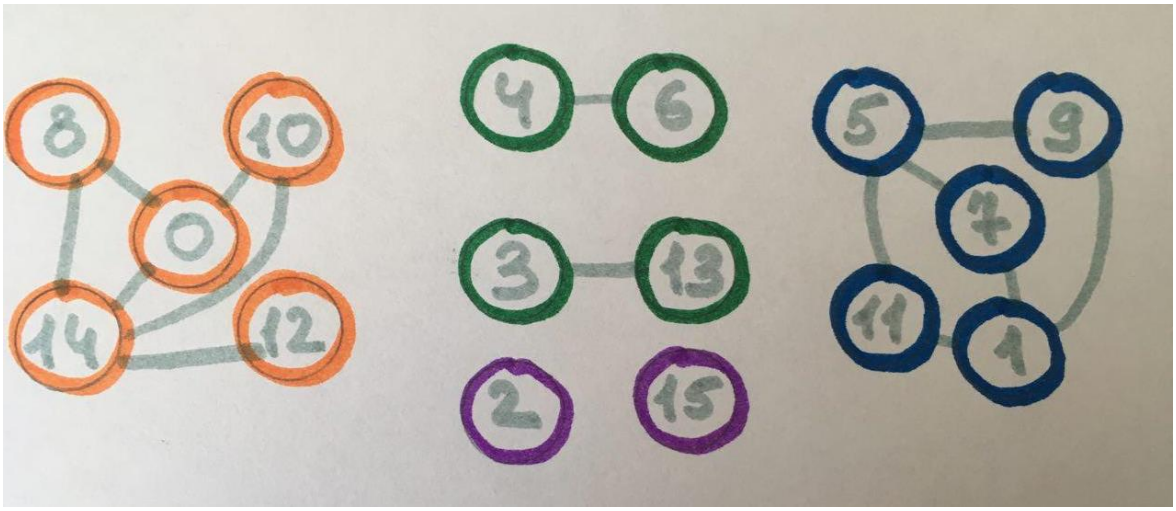
0	15	1	4	3	2
8		5	6	13	
10		7			
12		9			
14		11			

Третье условие(проверка совместимости следующих состояний):

0-8->1-9	V
0-10->1-11	V
0-12->1-13	X
0-14->1-15	V
8-10 -> 9-11 -> 10-12 -> 11-13	X
8-12 -> 9-13	X
8-14	V
10-12 -> 11-13	X
10-14	V
12-14	V
1-5->2-6->3-7	X
1-7	V
1-9	V
1-11	V
5-7	V
5-9	V
5-11	V
7-9->8-10	X
7-11->8-12->9-13	X
9-11->	X
4-6->5-7	V
3-13	V
2	V
15	V

0-8	0-10	~~0-12~~	0-14
~~8-10~~	~~8-12~~	8-14	
~~10-12~~	10-14	12-14	
~~1-5~~	1-7	1-9	1-11
5-7	5-9	5-11	
~~7-9~~	~~7-11~~		
~~9-11~~			
4-6			
3-13			

З отриманих станів можна скласти граф сумісності, компоненти якого можна об’єднати в один стан.



Об'єднаємо наступні еквівалентні стани:

0 – 0, 10
1 – 8
2 – 12, 14
3 – 4, 6
4 – 3, 13
5 – 2
6 – 1, 11
7 – 5, 7
8 – 9
9 – 15

Спрощена таблиця переходів:

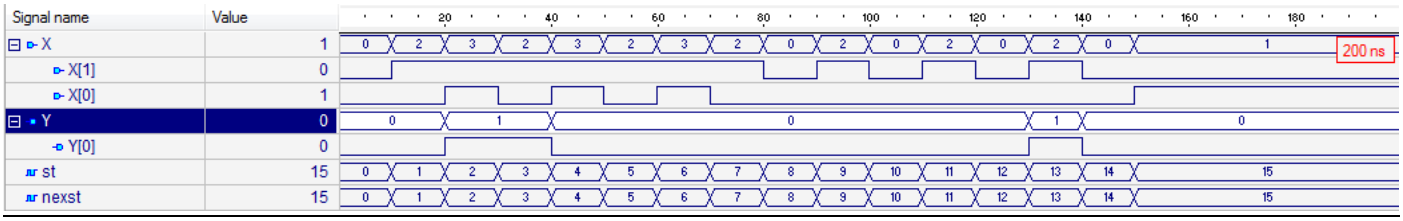
Q	**Вихідний склад**	**Y**	**X2 X1**			
			00	***01***	***10***	***11***
0	{0, 10}	***0***	(0)	-	6	-
1	{8}	***0***	(1)	-	8	-
2	{12, 14}	***0***	(2)	9	4	-
3	{4, 6}	***0***	-	-	7	(3)
4	{3, 13}	***1***	2	-	(4)	3
5	{2}	***1***	-	-	4	(5)
6	{1, 11}	***0***	2	-	(6)	5
7	{5, 7}	***0***	1	-	(7)	3
8	{9}	***0***	0	-	(8)	-
9	{15}	***0***	0	(9)	-	-

Перевіримо правильність мінімазації програмно:

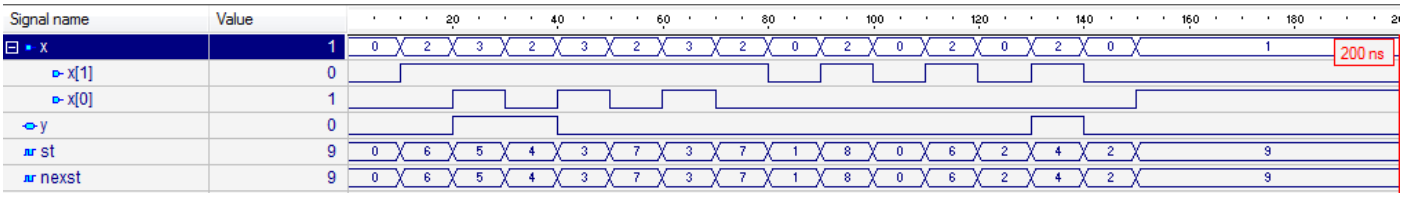
```
  library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity avtomat is
generic (
TZ: time :=0ns);
port(
x : in STD_LOGIC_VECTOR(1 downto 0);
y : inout STD_LOGIC
);

end avtomat;
```

```
--}} End of automatically maintained section

architecture avtomat of avtomat is
function vecint (vec1: std_logic_vector)
return integer is
variable retval:integer:=0;

        begin

                for i in vec1'length-1 downto 0 loop
                        if (vec1(i)='1') then retval:=retval*2+1;
                        else retval:= retval*2; end if;
                end loop;
                return retval;
        end vecint;
type stab is array (0 to 9, 0 to 3) of integer;
type outtab is array (0 to 9) of std_logic;
constant tab_st: stab :=
((0,20,6,20),
(1,20,8,20),
(2,9,4,20),
(20,20,7,3),
(2,20,4,3),
(20,20,4,5),
(2,20,6,5),
(1,20,7,3),
(0,20,8,20),
(0,9,20,20));
Constant tab_z : outtab :=('0','0','0','0','1','1','0','0','0','0');
signal st,nexst: integer:=0;
begin
process
begin
y<=tab_z(st);
wait on x, st;
if st = 20 then null;
else
nexst<=tab_st(st,vecint(x));
wait for TZ;
end if;
end process;

st<= transport nexst after TZ;

end avtomat;
```

До мінімізації:



Після мінімізації :



Подальша мінімізація неможлива.

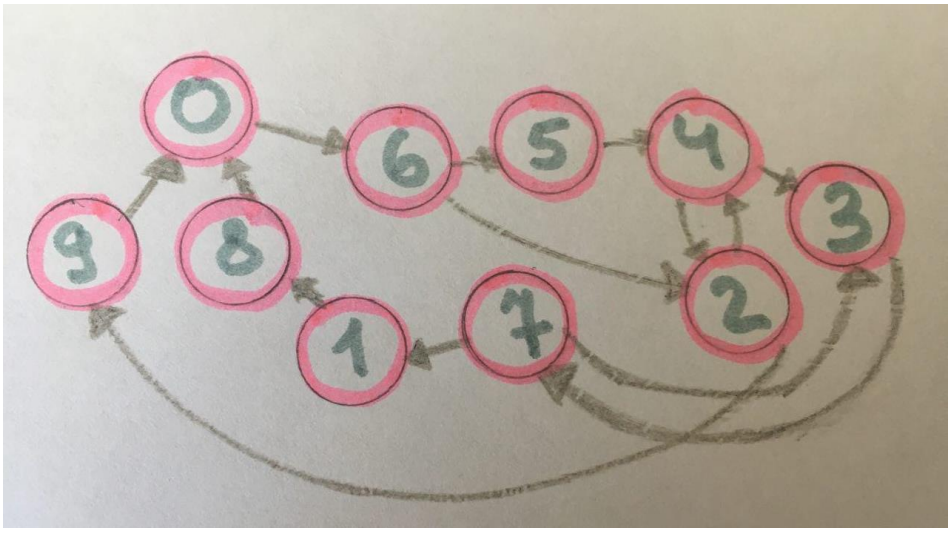
Кодування станів автомата

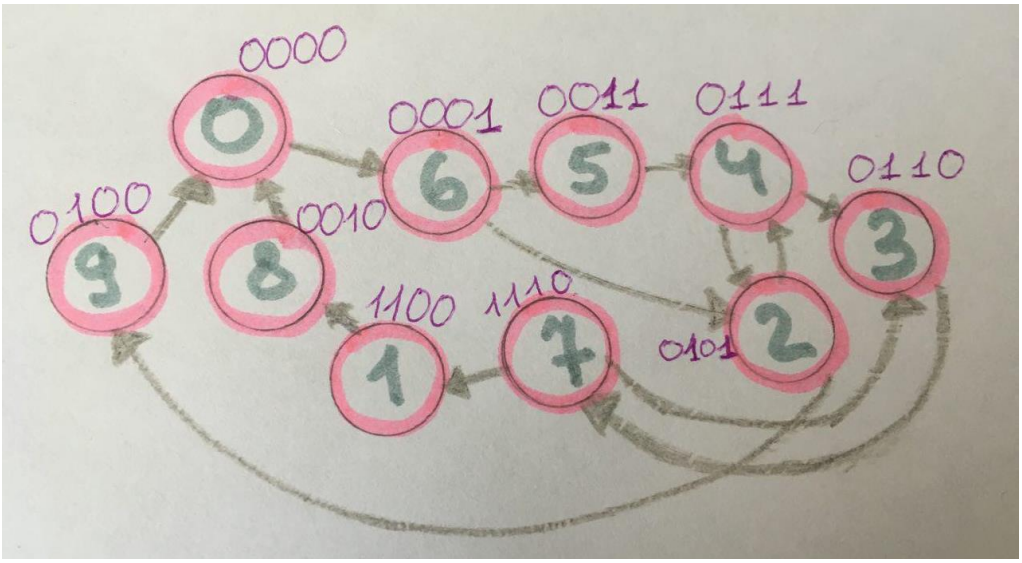
Автомат працює стійко, якщо в процесі його роботи не виникають критичні змагання елементів пам'яті. Якщо можливе виникнення критичних змагань, вважатимемо, що автомат нестійкий. Очевидно, що нестійка робота автомата недопустима. Тому першочерговим завданням, яке повинне вирішуватися при кодуванні внутрішніх станів автомата, є забезпечення його стійкої роботи. Це завдання вирішується з використанням спеціального, протигоночного кодування.

У даній роботі буде розглянутий простий метод кодування, що відноситься до першої групи. Цей метод носить назву: "Метод сусіднього кодування". Суть цього методу полягає в тому, що стани, між якими є переходи, кодуються сусіднім кодом. Змагання при цьому не виникають, оскільки на переходах змінює своє значення тільки один елемент пам'яті. Основним недоліком такого методу є те, що не усі автомати без попередніх еквівалентних перетворень таблиць переходів допускають сусіднє кодування.

- **не можна** закодувати стану автомата сусіднім кодом, якщо його граф містить контура непарної довжини.
- **стан автомата неможливо закодувати сусіднім кодом,** якщо його граф містить між сусідніми станами другого порядку більше двох вершин. Два стани автомата є сусідніми другого порядку, якщо між ними знаходиться один стан.

Побудуємо граф станів автомата:





Закодуємо стана автомату:

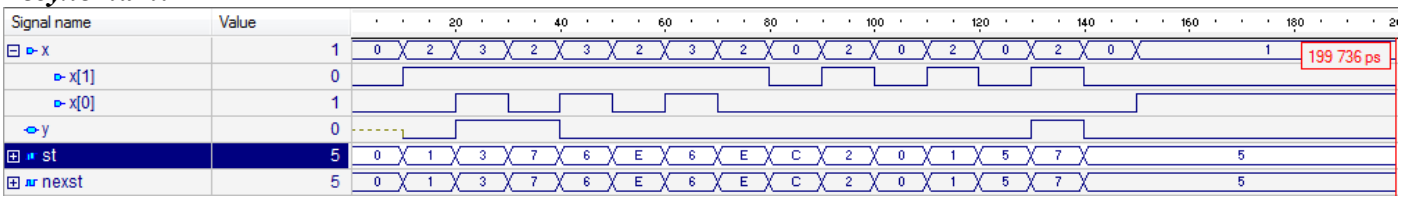
Q	q3q2q1q0	Состояния входа				Y
		00	01	10	11	
0	0000	(0000)	-	0001	-	0
1	1100	(1100)	-	0010	-	0
2	0101	(0101)	0100	0111	-	0
3	0110	-	-	1110	(0110)	0
4	0111	0101	-	(0111)	0110	1
5	0011	-	-	0111	(0011)	1
6	0001	0101	-	(0001)	0011	0
7	1110	1100	-	(1110)	0110	0
8	0010	0000	-	(0010)	-	0
9	0100	0000	(0100)	-	-	0

Перевіремо правильність кодування:

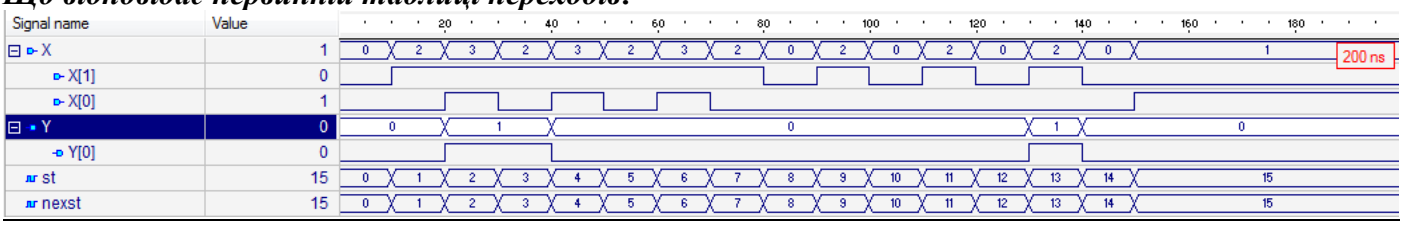
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity kod_grey is
        generic (Lin: integer :=1;
        K: integer :=15; M: integer :=3;
        Lst : integer := 3;
        TZ: time :=0ns);
         port(x: in bit_vector (Lin downto 0);y : inout STD_LOGIC);
end kod_grey;
architecture kod_grey of kod_grey is
function vecint (vec1: bit_vector) return integer is
                variable retval:integer:=0;
        begin
                for i in vec1'length-1 downto 1 loop
                        if (vec1(i)='1') then retval:=(retval+1)*2;
```

```
                else  retval:= retval*2; end if;
            end loop;
            if vec1(0)='1' then
                retval:=retval+1;
            else null; end if;
            return retval;
        end vecint;
        type stab is array (0 to K, 0 to M) of bit_vector(Lst downto 0);
        type outtab is array (0 to K) of std_logic;
constant tab_st: stab :=(
("0000","1111","0001","1111"),          --0000-0
("0101","1111","0001","0011"),          --0001-6
("0000","1111","0010","1111"),          --0010-8
("1111","1111","0111","0011"),          --0011-5
("0000","0100","1111","1111"),          --0100-9
("0101","0101","0111","1111"),          --0101-2
("1111","1111","1110","0110"),          --0110-3
("0101","1111","0111","0110"),          --0111-4
("1111","1111","1111","1111"),          --1000
("1111","1111","1111","1111"),          --1001
("1111","1111","1111","1111"),          --1010
("1111","1111","1111","1111"),          --1011
("1100","1111","0010","1111"),          --1100-1
("1111","1111","1111","1111"),          --1101
("1100","1111","1110","0110"),          --1110-7
("1111","1111","1111","1111"));         --1111
constant tab_z : outtab :=('0', '0', '0', '1', '0', '0','0','1','0','0','0','0','0','0','0','0');
 signal st,nexst: bit_vector(Lst downto 0):="0000";
begin
        process
        begin
            wait on x, st;
            if st="1111" then null;
            else
                nexst<=tab_st(vecint(st),vecint(x));
                y<=tab_z(vecint(st));
            end if;
end process;
        st<= transport nexst after TZ;
end kod_grey;
```

Результат:



Що відповідає первинній таблиці переходів:



Визначення функцій збудження елементів пам'яті і функцій виходів автомата

Функції збудження елементів пам'яті залежать:

q(n)	q(n+1)	R(n)	S(n)
0	0	*	0
0	1	0	1
1	0	1	0
1	1	0	*

№	Код Грея	Y	X1\X2																													
			00								01								10								11							
			R3	**S3**	R2	S2	**R1**	**S1**	R0	S0	**R3**	**S3**	R2	S2	**R1**	**S1**	R0	S0	**R3**	**S3**	R2	S2	**R1**	**S1**	R0	S0	**R3**	**S3**	R2	S2	**R1**	**S1**	R0	S0
0	0000	0	*	0	*	0	*	0	*	0	*	*	*	*	*	*	*	*	*	0	*	0	*	0	0	1	*	*	*	*	*	*	*	*
1	1100	0	0	*	0	*	*	0	*	0	*	*	*	*	*	*	*	*	1	0	1	0	0	1	*	0	*	*	*	*	*	*	*	*
2	0101	0	*	0	0	*	*	0	0	*	*	0	0	*	*	0	1	0	*	0	0	*	0	1	0	*	*	*	*	*	*	*	*	*
3	0110	0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	0	1	0	*	0	*	*	0	*	0	0	*	0	*	*	0
4	0111	1	*	0	0	*	1	0	0	*	*	*	*	*	*	*	*	*	*	0	0	*	0	*	0	*	*	0	0	*	0	*	1	0
5	0011	1	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	0	0	1	0	*	0	*	*	0	*	0	0	*	0	*
6	0001	0	*	0	0	1	*	0	0	*	*	*	*	*	*	*	*	*	*	0	*	0	*	0	0	*	*	0	*	0	0	1	0	*
7	1110	0	0	*	0	*	1	0	*	0	*	*	*	*	*	*	*	*	0	*	0	*	0	*	*	0	1	0	0	*	0	*	*	0
8	0010	0	*	0	*	0	1	0	*	0	*	*	*	*	*	*	*	*	*	0	*	0	0	*	*	0	*	*	*	*	*	*	*	*
9	0100	0	*	0	1	0	*	0	*	0	*	0	0	*	*	0	*	0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

Q	**q3q2q1q0**	**Состояния входа**			
		00	**01**	**10**	**11**
0	0000	(0000)	-	0001	-
1	1100	(1100)	-	0010	-
2	0101	(0101)	0100	0111	-
3	0110	-	-	1110	(0110)
4	0111	0101	-	(0111)	0110
5	0011	-	-	0111	(0011)
6	0001	0101	-	(0001)	0011
7	1110	1100	-	(1110)	0110
8	0010	0000	-	(0010)	-
9	0100	0000	(0100)	-	-

R_3	X2\X1			
$Q_3Q_2Q_1Q_0$	00	01	10	11
0000	*	*	*	*
1100	0	*	1	*
0101	*	*	*	*
0110	*	*	0	*
0111	*	*	*	*
0011	*	*	*	*
0001	*	*	*	*
1110	0	*	0	1
0010	*	*	*	*
0100	*	*	*	*

R_3	X2\X1							
$Q_3Q_2Q_1Q_0$	000	010	101	110	001	011	101	110
000	*	*	*	*				
110	0	*	1	*				
010	*	*	*	*				
011	*	*	0	*				
011	*	*	*	*				
001	*	*	*	*				
000	*	*	*	*				
111	0	*	0	1				
0010	*	*	*	*				
0100	*	*	*	*				