嵌套状态机的描述

嵌套状态机，与普通状态机一样，也可以用两种风格进行描述：

- 一段式
- 两段式或三段式

分述如下：

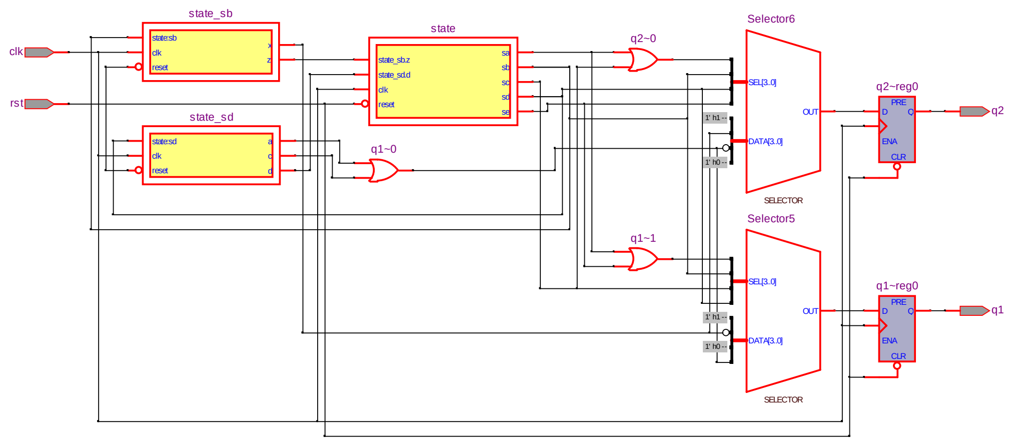
◆ 一段式描述方法：

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


ENTITY fsm IS

        PORT
        (
                clk : in std_logic;
                rst : in std_logic;
                q1 : out std_logic;
                q2 : out std_logic
        );

END fsm;

ARCHITECTURE fsm_architecture OF fsm IS

type state_all_type is (sa, sb, sc, sd, se);
signal state : state_all_type;
--signal next_state : state_all;

type state_sb_type is (x, y, z);
signal state_sb : state_sb_type;
--signal next_state_sb : state_sb_type;

type state_sd_type is (a, b, c, d);
signal state_sd : state_sd_type;
--signal next_state_sd : state_sd_type;

BEGIN

process(clk, rst)
```

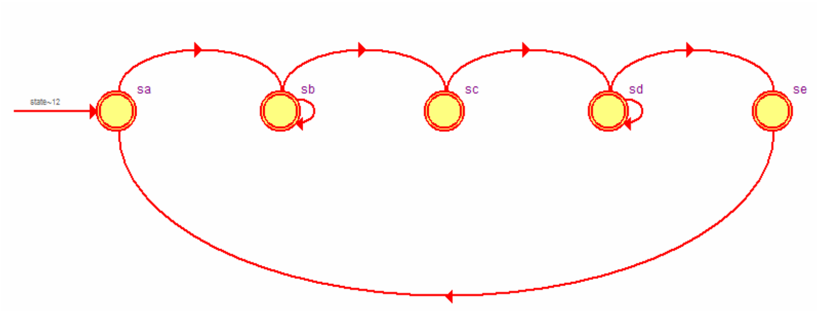
```
begin
    if rst = '0' then
        state <= sa;
        state_sb <= x;
        state_sd <= a;
        q1 <= '0';
        q2 <= '0';

    elsif clk'event and clk = '1' then

        case state is

            when sa =>
                state <= sb;
                q1 <= '1';
                q2 <= '1';

            when sb =>
                case state_sb is
                    when x =>
                        state_sb <= y;
                        q1 <= '0';
                        q2 <= '1';
                    when y =>
                        state_sb <= z;
                        q1 <= '1';
                        q2 <= '0';
                    when z =>
                        state_sb <= x;
                        state <= sc;
                        q1 <= '1';
                        q2 <= '0';
                    when others =>
                        state_sb <= x;
                        state <= sc;
                        q1 <= '0';
                        q2 <= '0';
                end case;

            when sc =>
                state <= sd;
                q1 <= '0';
                q2 <= '1';
```

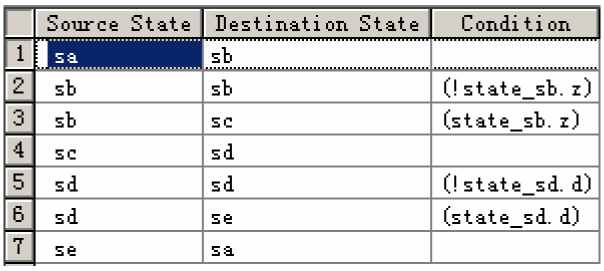
```
                        when sd =>
                            case state_sd is
                                when a =>
                                    state_sd <= b;
                                    q1 <= '1';
                                    q2 <= '0';
                                when b =>
                                    state_sd <= c;
                                    q1 <= '0';
                                    q2 <= '1';
                                when c =>
                                    state_sd <= d;
                                    q1 <= '1';
                                    q2 <= '0';
                                when d =>
                                    state_sd <= a;
                                    state <= se;
                                    q1 <= '0';
                                    q2 <= '1';
                                when others =>
                                    state_sd <= a;
                                    state <= se;
                                    q1 <= '0';
                                    q2 <= '0';
                            end case;

                        when se =>
                            state <= sa;
                            q1 <= '1';
                            q2 <= '0';

                        when others =>
                            state <= sa;
                            q1 <= '0';
                            q2 <= '0';

                    end case;
                end if;

end process;


END fsm_architecture;
```

RTL 视图如下：



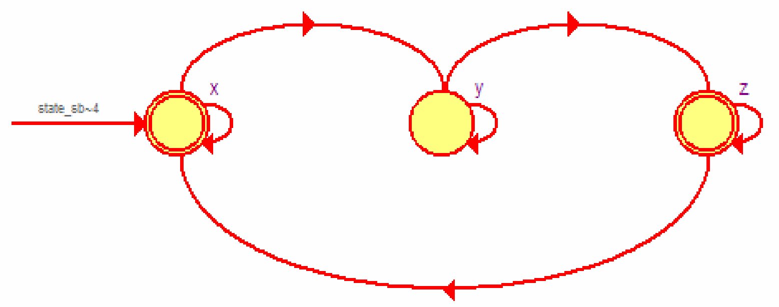
State 状态转换图

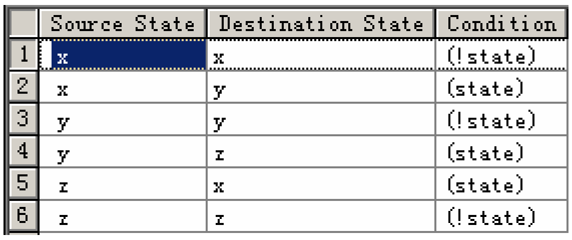




	Source State	Destination State	Condition
1	sa	sb	
2	sb	sb	(!state_sb.z)
3	sb	sc	(state_sb.z)
4	sc	sd	
5	sd	sd	(!state_sd.d)
6	sd	se	(state_sd.d)
7	se	sa	

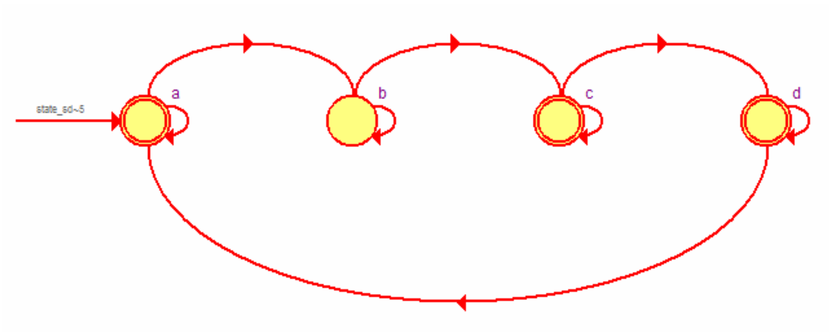
State_sb 子状态转换图

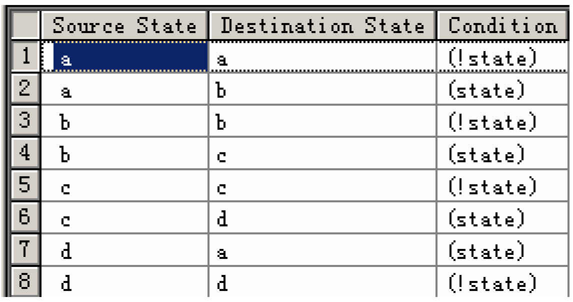




	Source State	Destination State	Condition
1	x	x	(!state)
2	x	y	(state)
3	y	y	(!state)
4	y	z	(state)
5	z	x	(state)
6	z	z	(!state)

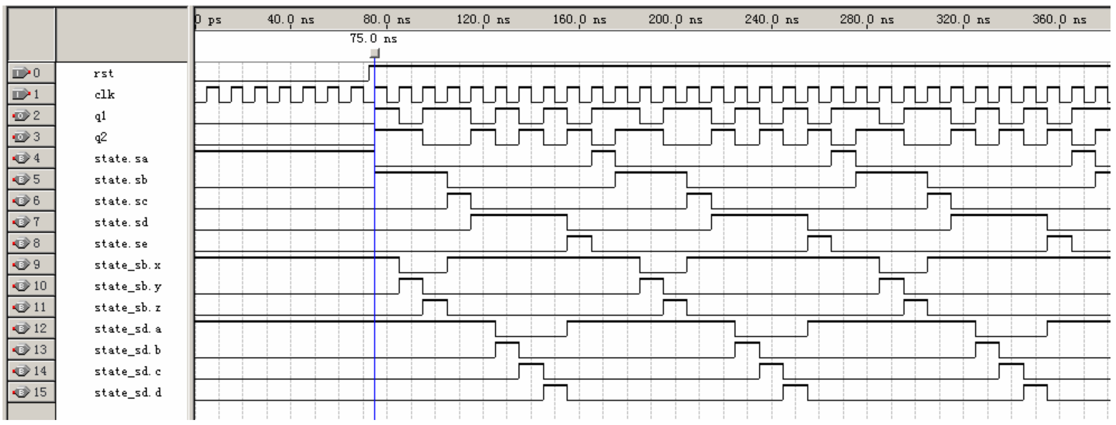
State_sd 子状态转换图





	Source State	Destination State	Condition
1	a	a	(!state)
2	a	b	(state)
3	b	b	(!state)
4	b	c	(state)
5	c	c	(!state)
6	c	d	(state)
7	d	a	(state)
8	d	d	(!state)

仿真结果如下：



需要的资源：

组合逻辑 ： 13

寄存器 ： 14

◆ 三段式描述方法：

LIBRARY ieee;

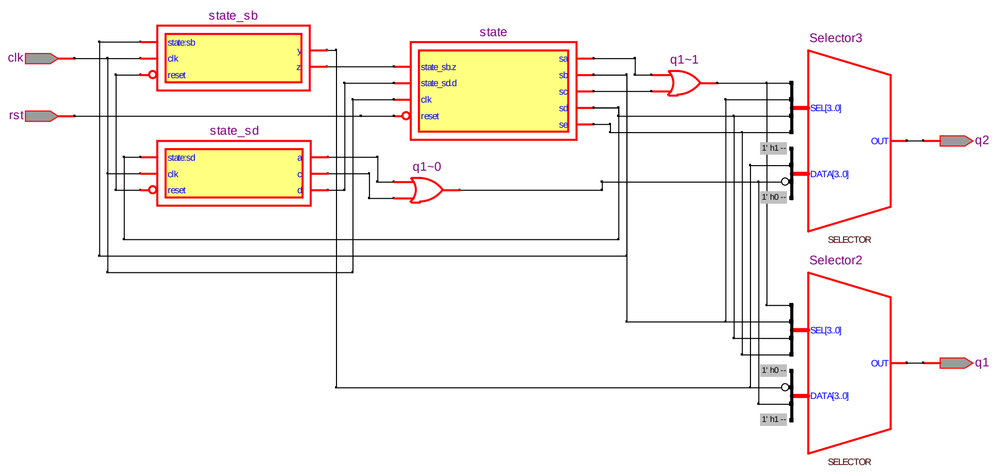
```
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY fsm IS
    PORT
    (
        clk : in std_logic;
        rst : in std_logic;
        q1 : out std_logic;
        q2 : out std_logic
    );

END fsm;

ARCHITECTURE fsm_architecture OF fsm IS

type state_all_type is (sa, sb, sc, sd, se);
signal state : state_all_type;
signal next_state : state_all_type;

type state_sb_type is (x, y, z);
signal state_sb : state_sb_type;
signal next_state_sb : state_sb_type;

type state_sd_type is (a, b, c, d);
signal state_sd : state_sd_type;
signal next_state_sd : state_sd_type;

BEGIN

process(clk, rst)
begin
    if rst = '0' then
        state <= sa;
    elsif clk'event and clk = '1' then
        state <= next_state;
    end if;
end process;

process(clk, rst)
begin
    if rst = '0' then
        state_sb <= x;
    elsif clk'event and clk = '1' then
```

```
        state_sb <= next_state_sb;
    end if;
end process;

process(clk, rst)
begin
    if rst = '0' then
        state_sd <= a;
    elsif clk'event and clk = '1' then
        state_sd <= next_state_sd;
    end if;
end process;

process(state, state_sb, state_sd)
begin

    case state is
        when sa =>
            next_state <= sb;
            next_state_sb <= x;
            next_state_sd <= a;

        when sb =>
            case state_sb is
                when x =>
                    next_state_sb <= y;
                    next_state <= sb;
                    next_state_sd <= a;
                when y =>
                    next_state_sb <= z;
                    next_state <= sb;
                    next_state_sd <= a;
                when z =>
                    next_state_sb <= x;
                    next_state <= sc;
                    next_state_sd <= a;
                when others =>
                    next_state_sb <= x;
                    next_state <= sa;
                    next_state_sd <= a;
            end case;

        when sc =>
            next_state <= sd;
```

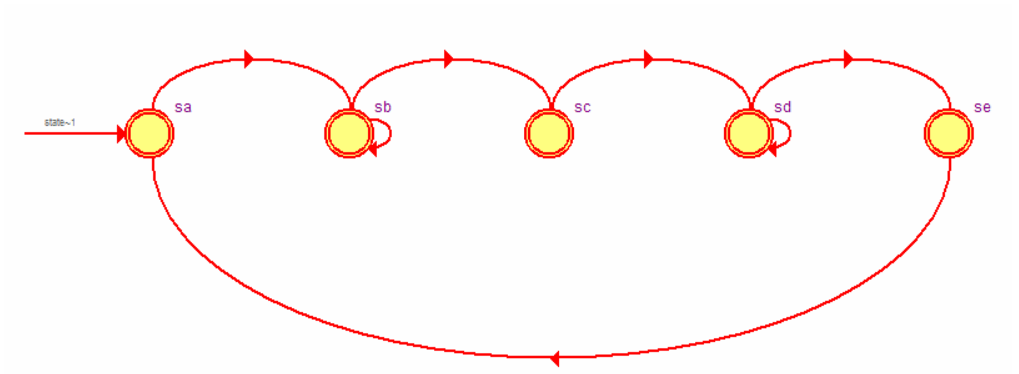
```
                next_state_sb <= x;
                next_state_sd <= a;

            when sd =>
                case state_sd is
                    when a =>
                        next_state_sd <= b;
                        next_state <= sd;
                        next_state_sb <= x;
                    when b =>
                        next_state_sd <= c;
                        next_state <= sd;
                        next_state_sb <= x;
                    when c =>
                        next_state_sd <= d;
                        next_state <= sd;
                        next_state_sb <= x;
                    when d =>
                        next_state_sd <= a;
                        next_state <= se;
                        next_state_sb <= x;
                    when others =>
                        next_state_sd <= a;
                        next_state <= sa;
                        next_state_sb <= x;
                end case;

            when se =>
                next_state <= sa;
                next_state_sb <= x;
                next_state_sd <= a;
        end case;
end process;

process(state, state_sb, state_sd)
begin

case state is
        when sa =>
            q1 <= '0';
            q2 <= '1';

        when sb =>
            case state_sb is
```

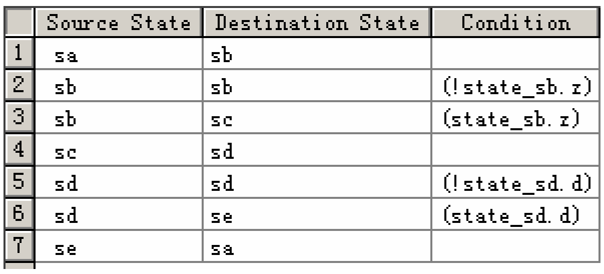
```
                when x =>
                    q1 <= '1';
                    q2 <= '0';
                when y =>
                    q1 <= '0';
                    q2 <= '1';
                when z =>
                    q1 <= '1';
                    q2 <= '0';
                when others =>
                    q1 <= '0';
                    q2 <= '0';
            end case;

        when sc =>
            q1 <= '0';
            q2 <= '1';

        when sd =>
            case state_sd is
                when a =>
                    q1 <= '1';
                    q2 <= '0';
                when b =>
                    q1 <= '0';
                    q2 <= '1';
                when c =>
                    q1 <= '1';
                    q2 <= '0';
                when d =>
                    q1 <= '0';
                    q2 <= '1';
                when others =>
                    q1 <= '0';
                    q2 <= '0';
            end case;

        when se =>
            q1 <= '1';
            q2 <= '0';
        when others =>
            q1 <= '0';
            q2 <= '0';
end case;
end process;

END fsm_architecture;
```

RTL 视图如下：



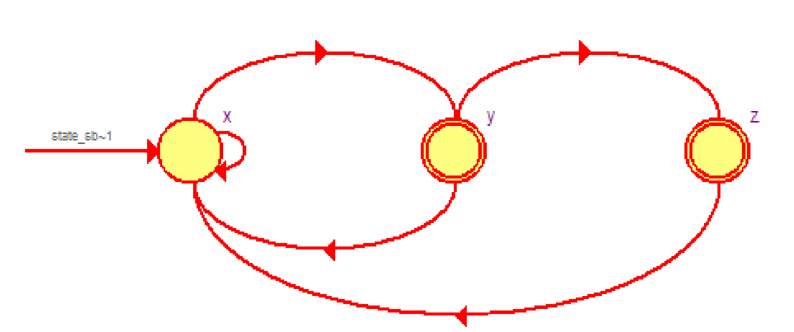
State 状态转换图

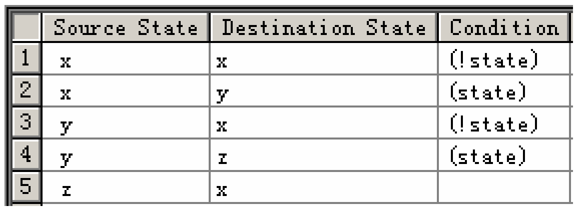




	Source State	Destination State	Condition
1	sa	sb	
2	sb	sb	(!state_sb.z)
3	sb	sc	(state_sb.z)
4	sc	sd	
5	sd	sd	(!state_sd.d)
6	sd	se	(state_sd.d)
7	se	sa	

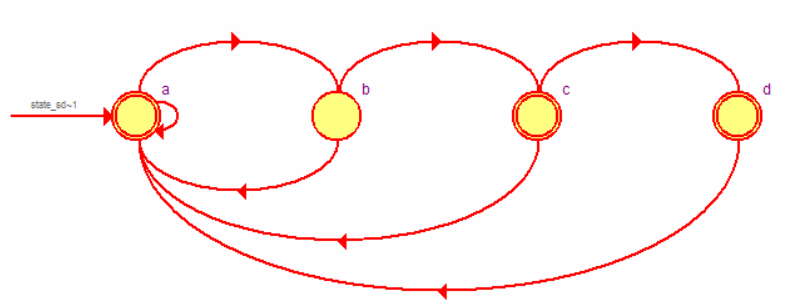
State_sb 状态转换图

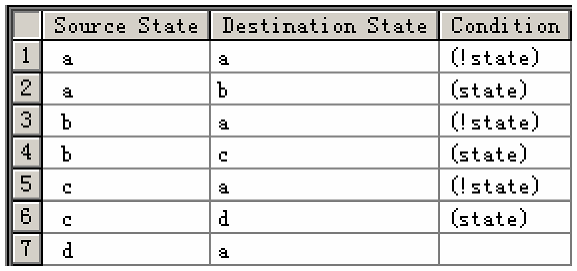




	Source State	Destination State	Condition
1	x	x	(!state)
2	x	y	(state)
3	y	x	(!state)
4	y	z	(state)
5	z	x	

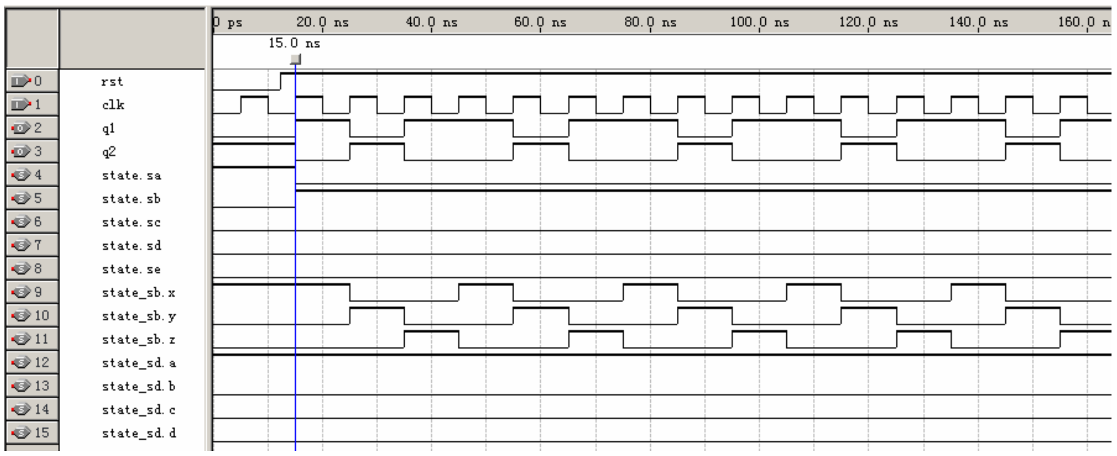
State_sd 子状态转换图





	Source State	Destination State	Condition
1	a	a	(!state)
2	a	b	(state)
3	b	a	(!state)
4	b	c	(state)
5	c	a	(!state)
6	c	d	(state)
7	d	a	

仿真结果



消耗资源：

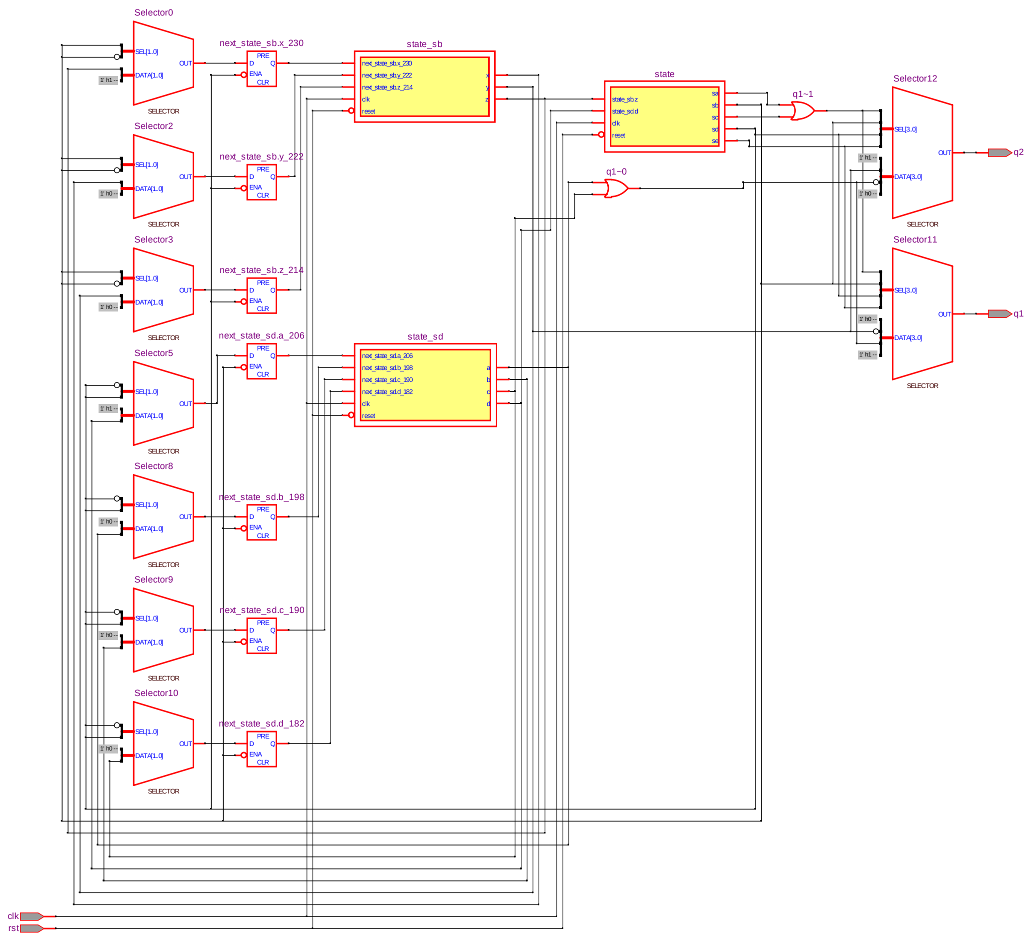
组合逻辑	：	16
寄存器	：	12

◆　三段式描述方法（2）：

```
                    when sb =>
                        case state_sb is
                            when x =>
                                next_state_sb <= y;
                                next_state <= sb;
    --                           next_state_sd <= a;
                            when y =>
                                next_state_sb <= z;
                                next_state <= sb;
    --                           next_state_sd <= a;
                            when z =>
                                next_state_sb <= x;
                                next_state <= sc;
    --                           next_state_sd <= a;
                            when others =>
                                next_state_sb <= x;
                                next_state <= sa;
    --                           next_state_sd <= a;
                        end case;
                    when sd =>
                    case state_sd is
                        when a =>
                            next_state_sd <= b;
                            next_state <= sd;
--                           next_state_sb <= x;
                        when b =>
                            next_state_sd <= c;
                            next_state <= sd;
--                           next_state_sb <= x;
                        when c =>
                            next_state_sd <= d;
                            next_state <= sd;
--                           next_state_sb <= x;
                        when d =>
                            next_state_sd <= a;
                            next_state <= se;
--                           next_state_sb <= x;
                        when others =>
                            next_state_sd <= a;
```

```
                                        next_state <= sa;
--                                        next_state_sb <= x;
                        end case;
```

在本描述程序中，在 state_sb 子状态中，并不明确给出 state_sd 子状态的值；同样，在 state_sd 子状态中，也不明确给出 state_sb 子状态的值，其综合后的 RTL 视图如下：



可以很清楚的看到，比起三段式描述方法（1），由于生成了很多锁存器，因此会占用更多的资源（从综合报告中，可以看出，所使用的逻辑资源为 25，寄存器资源还是 12）。因此，虽然仿真结果一致，但并不推荐使用这种写法。

小结：

✧ 两种写法所需要的资源不一样，采取三段式写法少用了两个寄存器，但组合逻辑资源增多；
✧ 在三段式写法中，在每一个状态中，都必须把所有的子状态都清楚的表示出来，否则在综合的过程中会出现锁存器；
✧ 采用的综合、仿真工具，全部是由 Quartus II 8.0 自带的工具。