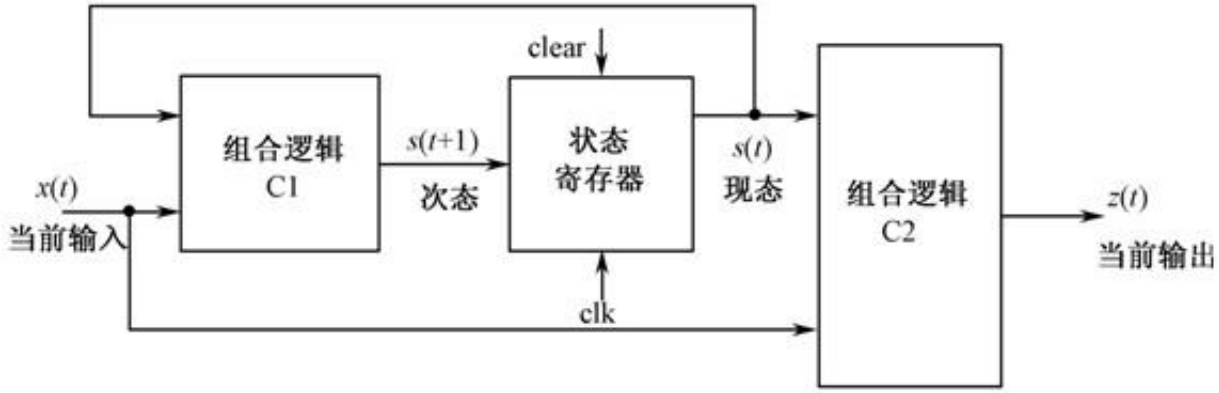
VHDL语言基础（三）

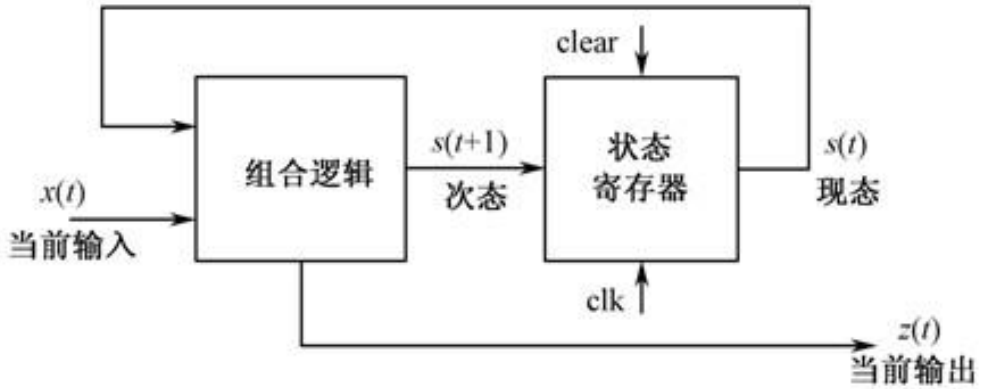
有限状态机

- 状态机：
  - 就是一组触发器的输出状态随着时钟和输入信号按照一定的规律变化的一种机制（过程）
- 时钟同步状态机：
  - 其中状态寄存器是由一组触发器组成的，用来记忆状态机当前所处的状态
  - 如果状态寄存器由**N**个触发器组成，这个状态机最多可记忆**2^N**
  - 所有触发器的时钟端都连接在一个共同的时钟信号上，状态的改变只可能发生在时钟的跳变沿时刻

有限状态机分类



- Mealy型状态机



- Moore型状态机

有限状态机设计思想

设计一个简单的状态机包括：

- 时钟信号：时序逻辑
- 复位信号：使状态机从期望的状态开始
- 输入信号： 激励状态改变
- 输出信号： 由内部状态和输入激励导致的输出结果
- 内部的状态转移逻辑（寄存器逻辑）
  - 要求设计中包含有限个状态
  - 各状态之间存在相互转移的关系（状态转移图）
  - 随着外部输入信号的变化，各状态之间相互转换
  - 由当前状态和输入信号，产生输出信号（输出逻辑）

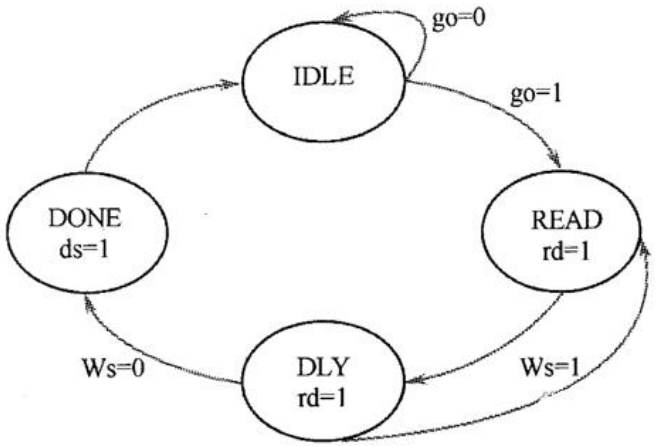
有限状态机状态编码

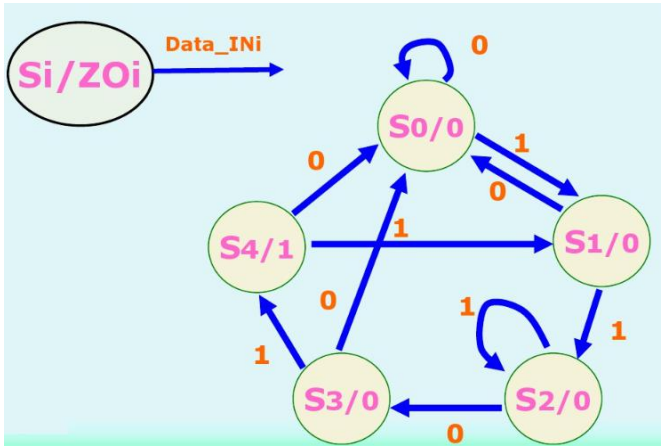
- 二进制编码：有n个寄存器表示2^n个状态
- 一位有效编码方式：使用N位状态寄存器来表达具有N个状态的状态机，每个状态都有它独立的寄存器，并且在任意时刻，只有一位有效，该位称“热点”，一点热one hot编码。

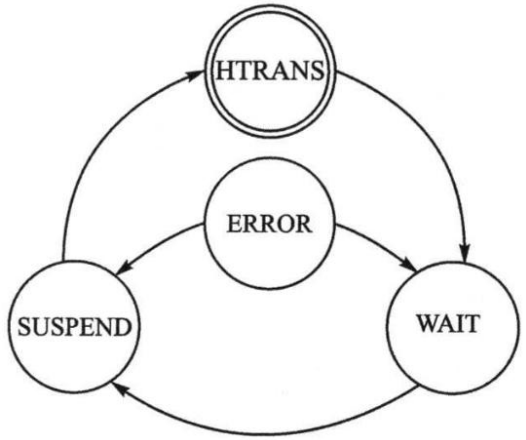
状态	顺序编码	One hot
Idel	00	0001
Rst	01	0010
Write	10	0100
Read	11	1000

有限状态机状态转移图

- 有限个确定状态，每个状态有表示的功能含义
- 在**CLK**作用下，在不同输入条件下，状态之间的相互转移







有限状态机描述方法

- 状态机设计描述的对象
  - 状态变化（现态+次态）（时序逻辑）
  - 输出逻辑（组合逻辑）
- 双进程：
  - 一个进程用于描述状态变化
  - 一个进程用于描述输出逻辑
- 单进程：
  - 用同一个进程描述状态变化、输出逻辑
- 详见QuartusII 状态机设计模版
  - **mealy型**
  - **moore型**

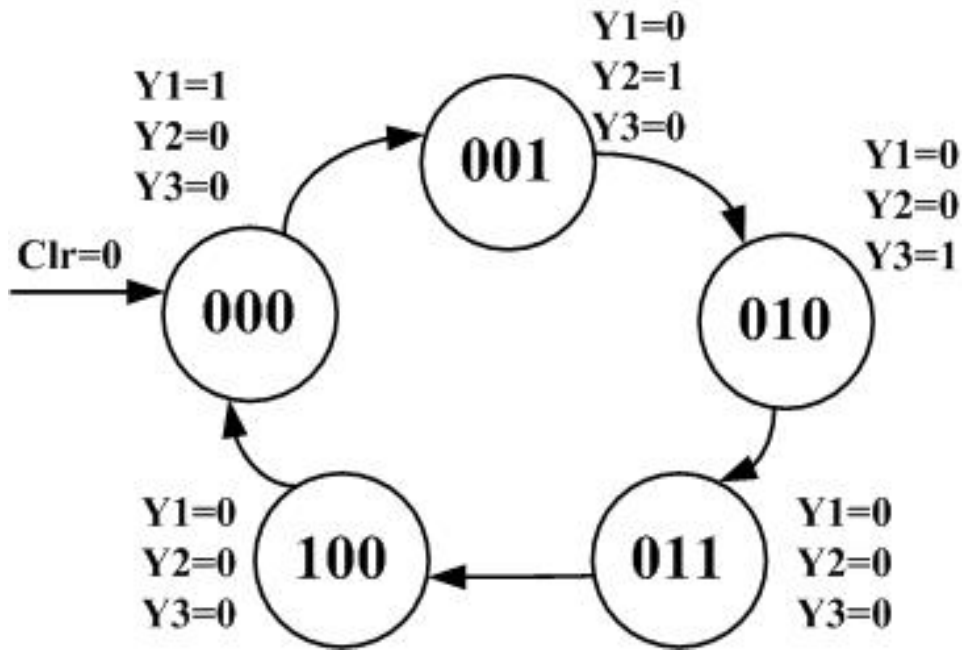
```
architecture rtl of four_state_mealy_state_machine is

    -- Build an enumerated type for the state machine
    type state_type is (s0, s1, s2, s3);

    -- Register to hold the current state
    signal state : state_type;

    -- Determine the output based only on the current state
    -- and the input (do not wait for a clock edge).
    process (state, input)
    begin
            case state is
                when s0=>
                    if input = '1' then
                        output <= "00";
                    else
                        output <= "01";
                    end if;
                when s1=>
                    if input = '1' then
                        output <= "01";
                    else
                        output <= "11";
                    end if;
                when s2=>
                    if input = '1' then
                        output <= "10";
                    else
                        output <= "10";
                    end if;
                when s3=>
                    if input = '1' then
                        output <= "11";
                    else
                        output <= "10";
                    end if;
            end case;
    end process;

end rtl;
```

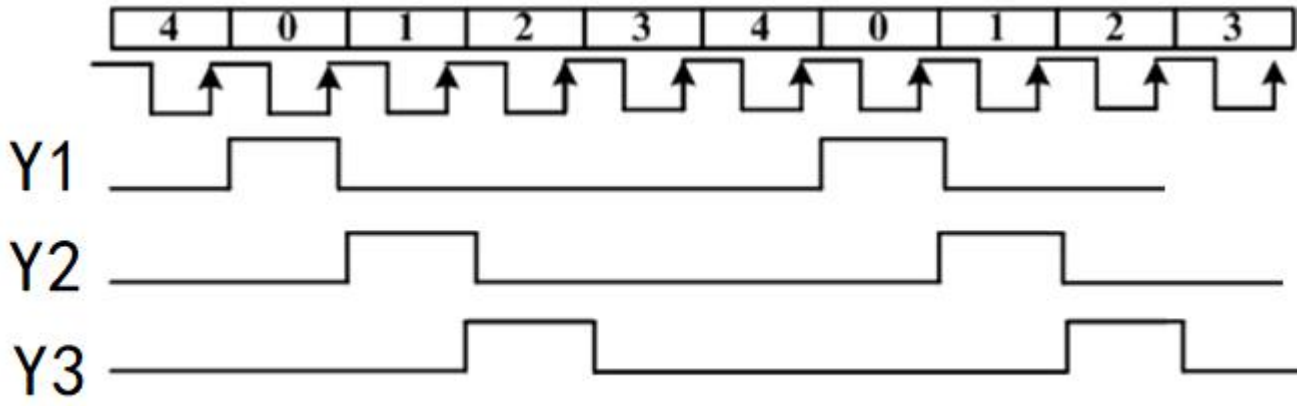
```
eset)

: '1' then
te <= s0;

ıg_edge(clk)) then

Determine the next state synchronously, based on
he current state and the input
e state is
  when s0=>
      if input = '1' then
          state <= s1;
      else
          state <= s0;
      end if;
  when s1=>
      if input = '1' then
          state <= s2;
      else
          state <= s1;
      end if;
  when s2=>
      if input = '1' then
          state <= s3;
      else
          state <= s2;
      end if;
  when s3=>
      if input = '1' then
          state <= s3;
      else
          state <= s1;
      end if;
  end case;

end if;
end process;
```

实例一：分频器

- 分频器输出时序，如下图
- 画出状态转移图
  - 5个状态 “000”~“100” 顺序表示
  - 每个状态下的输出逻辑 ：Y1、Y2、Y3

状态	输出
000	100
001	010
010	001
011	000
100	000
xxx	000

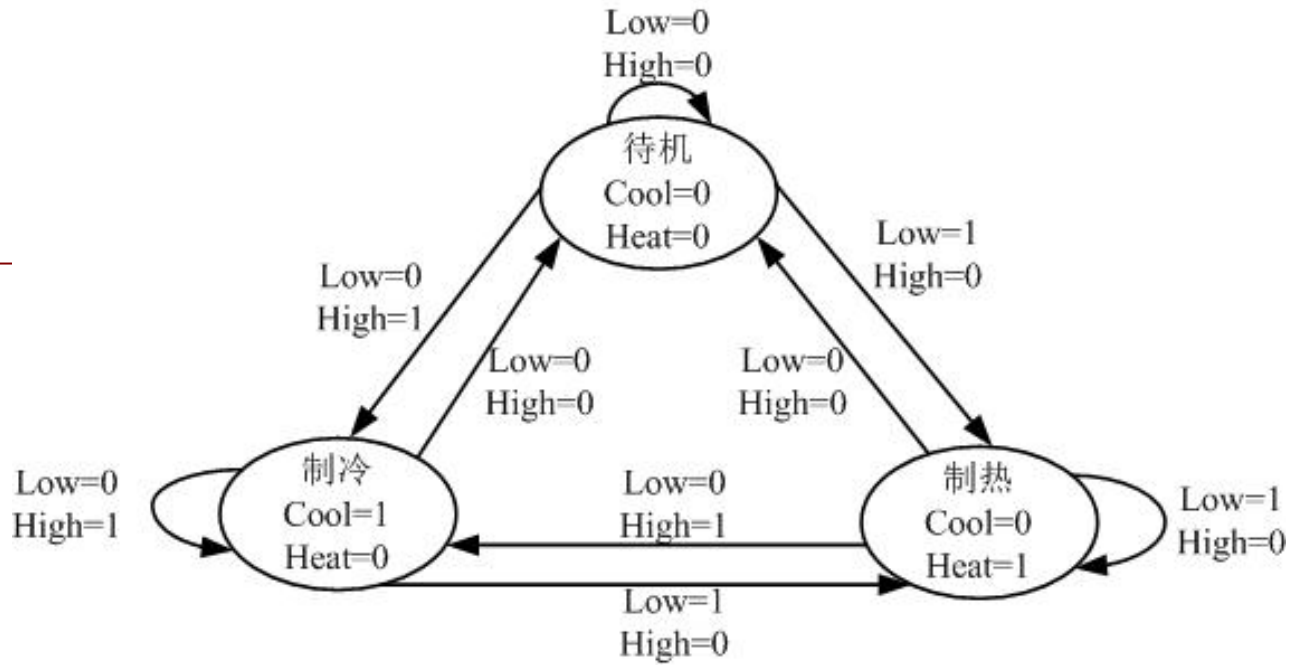




实例二：空调控制逻辑

- 输入信号：high 和 low连接两个温度传感器
  - 温度在18度～24度之间，high和low输入为‘L’
  - 温度高于24度，high输入为‘H’/ low 输入为‘L’
  - 温度低于18度，low 输入为‘H’/ high输入为‘L’
- 判断采样间隔：输入CLK 频率决定
- 输出信号：cool 和 heat 控制空调制冷和制热

- 分析画出状态转移图



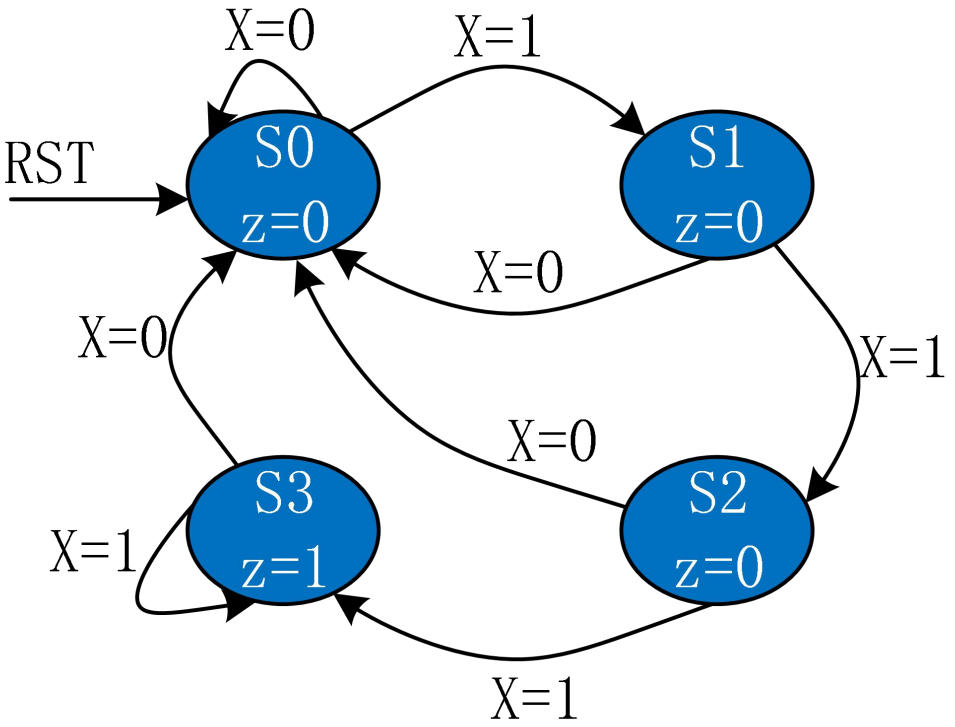
实例三：序列检测

- 设计一个**111**序列信号检测器
- 输入：待测串行信号**X**，时钟**clk**
- 输出：检测状态信号**Z**

输入X　0 1 0 1 1 1 0 1 0 1 1 1 1 1 1 0 0 0 0 0

输出Z　0 0 0 0 0 **1** 0 0 0 0 0 **1 1 1 1** 0 0 0 0 0

- 有限个状态：4种状态： **S0、S1、S2、S3**
- 状态的功能含义： 检测到几个连续的**1**；
  - **S0** ： 未检测到输入**1**
  - **S1** ： 检测到**1bit** “**1**”输入；
  - **S2** ： 检测到**2bit**连续 “**1**”输入；
  - **S3** ： 检测到**3bit**连续 “**1**”输入；
- 输出信号**Z**：只和当前状态有关
  - **S0** ： **z=0**　**S1** ： **z=0**　**S2** ： **z=0**　**S3** ： **z=1**



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY exam_state IS
PORT(  CLK, Xin: IN  STD_LOGIC;
       Zout:  OUT  STD_LOGIC);
END exam_state ;

architecture Behavioral of exam_state is
type state_type is (S0,s1,s2,s3);
signal state: state_type;
begin

Process(state)
Begin
  if state=S3 then Zout <='1' ;
  else   Zout <='0';
  end if;
End process;
```

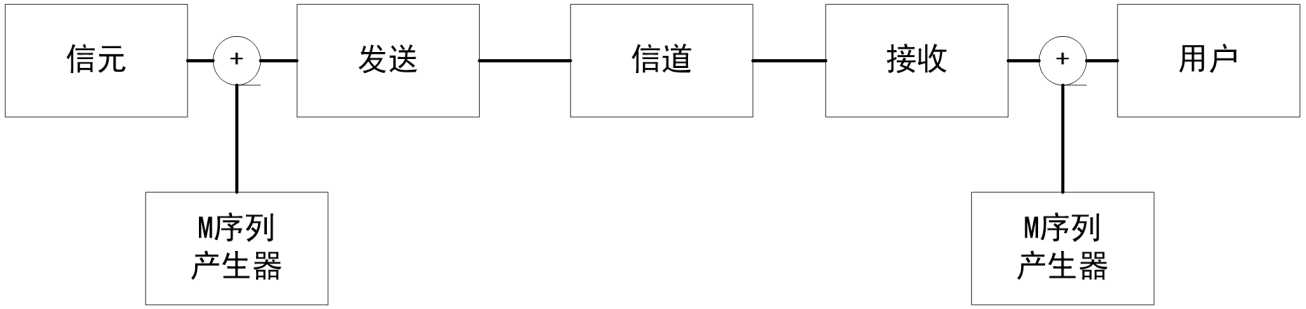
输出逻辑

```
process (CLK)
begin
    if (CLK'event and CLK= '1') then
    case (state) is
       when S0 =>
         if xin='1' then state <= S1;
         else state <= s0;
         end if;
       when S1 =>
         if xin='1' then state <= S2;
         else state <= s0;
         end if;
       when S2 =>
         if xin='1' then state <= S3;
         else state <= s0;
         end if;
       when S3 =>
         if xin='1' then state <= S3;
         else state <= s0;
         end if;
       when others => state <=S0;
    end case;
   end if;
end process;
```

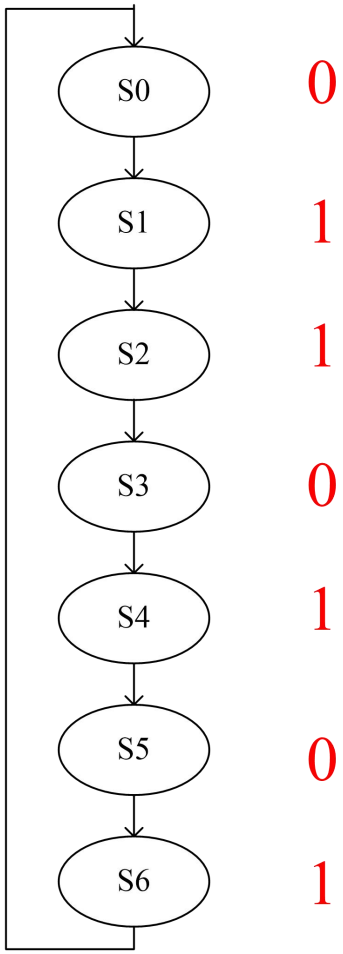
状态转移描述

实例四：序列信号发生器

•序列发生器的应用广泛： **M序列加密系统**



- 设计产生序列信号为**0110101**的发生器
- 根据序列长度**M**，定义**M**个状态；
- 根据要产生的序列信号，定义输出驱动逻辑；



```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY exam_state IS
PORT(  CLK: IN  STD_LOGIC;
       Yout:  OUT  STD_LOGIC);
END exam_state ;

architecture Behavioral of exam_state is
type state_type is (S0,s1,s2,s3,s4,s5,S6);
signal state: state_type;
begin
process (CLK)
   begin
      if (CLK'event and CLK= '1') then
      case (state) is
       when S0 =>
             state <= S1 ;
             Yout <= '0' ;
       when S1 =>
             state <= S2 ;
             Yout <= '1' ;
 when S2 =>
       state <= S3;
       Yout <= '1';
 when S3 =>
       state <= S4;
       Yout <= '0';
 when S4 =>
       state <= S5;
       Yout <= '1';
 when S5 =>
       state <= S6;
       Yout <= '0';
 when S6 =>
       state <= S0;
       Yout <= '1';
 when others =>
       state <=  S1;
 end case;
 END IF;
end process;
end Behavioral;
```

状态转移、输出逻辑的描述

作业练习:

1. 编写一个8路彩灯控制程序，要求彩灯有以下3种演示效果，顺序演示：

a. 8路彩灯同时亮灭；

b. 从左至右逐个亮（每次只有1路亮）

c. 8路彩灯每次4路灯亮，4路灯灭，且亮灭相间，交替亮灭

2. 设计一个汽车尾灯控制器，汽车左右两侧各有3个尾灯，要求控制尾灯按照如下规则亮灭：

a. 汽车直行，两侧指示灯全灭；

b. 右转弯时，左侧指示灯全灭，右侧指示灯循环顺序点亮（000，001，010，100，000，001…）

c. 左转弯时，右侧指示灯全灭，左侧指示灯循环顺序点亮（000，001，010，100，000，001…）

d. 刹车时，两侧的指示灯全亮。