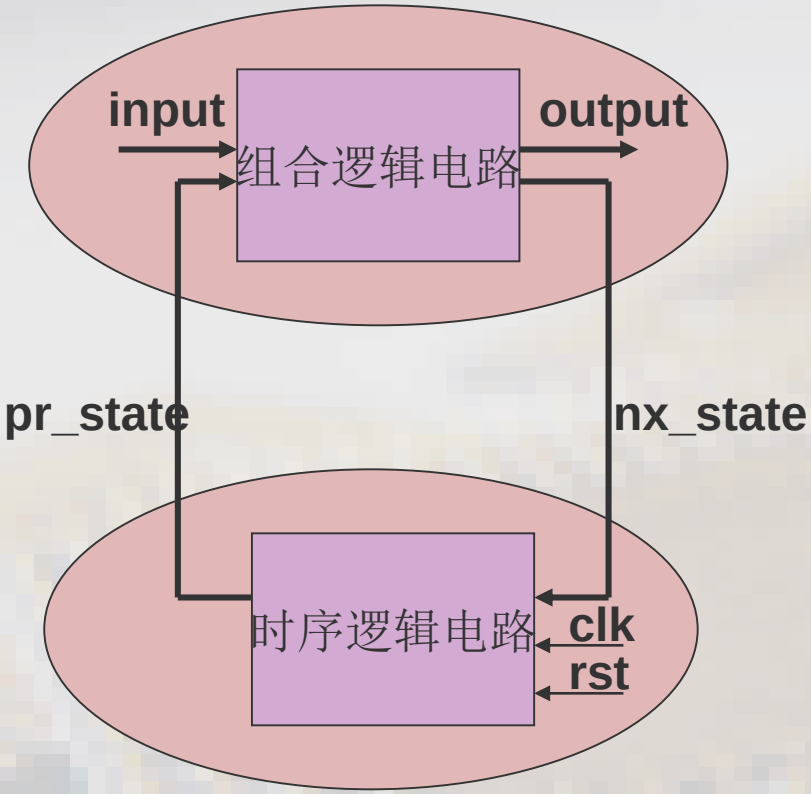
第8章 状态机(Finite State Machine,FSM)

- FSM是为时序电路设计而创建的特殊模型技术，在针对任务顺序非常明确的电路（如交通灯控制器）是非常实用。
- 理论上，任何时序电路都可以建立FSM模型，但并不总是一种高效的方法。如果一味地追求使用FSM来设计时序电路，可能会导致代码冗长和容易出错。例如，任务简单的寄存器就不必使用FSM方式实现。又例如，虽然任务与顺序很明确，但任务数目太多或者性能要求较高时，也不宜用FSM方式实现。
- 状态机的设计包含两个主要过程，一是状态机的建模，二是状态机的编码。

8.1 引言



- 状态机的组成:如图。
- 状态机的种类：

  Mealy型：当前状态、当前输入相关

  Moore型：仅当前状态相关

- VHDL代码结构：

  时序逻辑部分：process内部

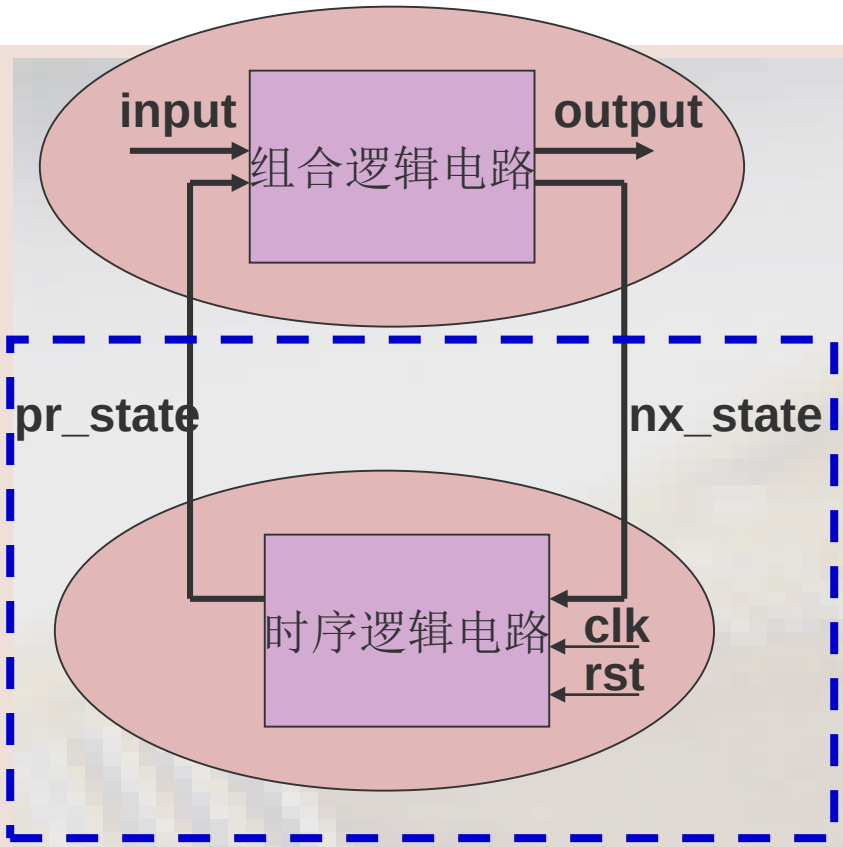
  组合逻辑部分：

- 在使用FSM方式设计VHDL代码时，通常会在结构体的开始部分插入一个用户自定义的枚举数据类型，其中包含所有可能出现的电路状态。

8.2 设计风格#1

一种结构清晰、易于实现的FSM设计风格：

- FSM中的时序逻辑部分和组合逻辑部分分别独立设计；
- 定义一个枚举数据类型，内部包含所有FSM需要的状态；



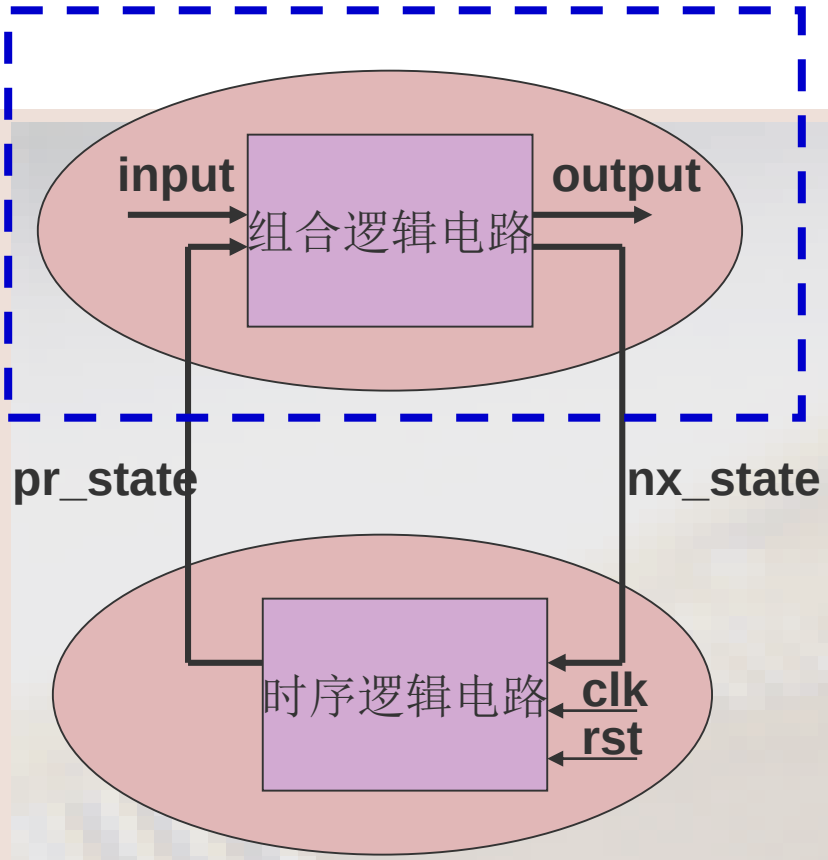
FSM中时序逻辑部分的设计特点：

- 确定的输入/输出端口
- 典型的模板可供使用

```
----------lower section----------
process (clock, reset)
begin
  if (reset = '1')  then
       pr_state <= state0;
   elsif (clock'event and clock='1')  then
       pr_state <= nx_state;
   end if;
end process;
```

表现为位宽

- 标准的设计
- 寄存器数目少：默认的编码方式下，$\log_2 n$



FSM中组合逻辑部分的设计特点：

- 并发代码、顺序代码皆可；
- 顺序代码方式的设计模板

```
----------upper section--------------
process (input, pr_state)
begin
    case pr_state is
    when state0=>       //多个条件转移分支
        if  (input=...)  then
            output <= <value>;
            nx_state<=state1;
        else  ....;
        end if;
    when  state1=>
        if  (input=...)  then
            output <= <value>;
            nx_state<=state2;
        else  ....;
        end if;
        ......
        end case;
end process;
```

- process的敏感信号列表
- 完整列出IN/OUT组合
- 无边沿赋值，无寄存器生成

设计风格#1的状态机模板

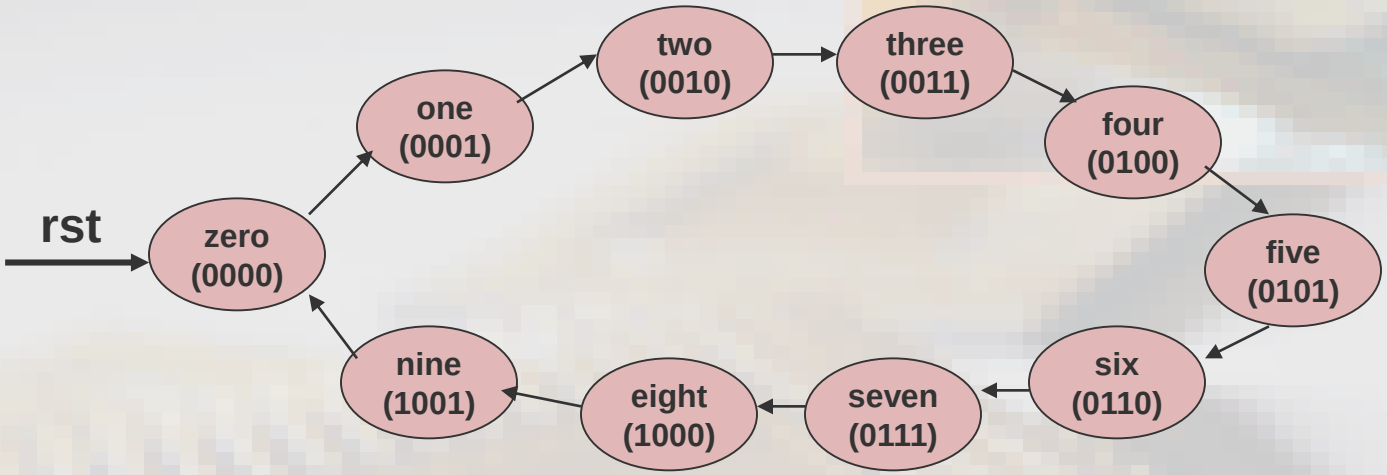
```
library ieee;
use ieee.std_logic_1164.all;
entity <entity_name> is
    port (input: in <data_type>;
            reset, clock: in std_logic;
           output: out <data_type>);
end <entity_name>;
architecture <arch_name> of <entity_name> is
  type state is (state0, state1, state2, state3, ...);
  signal pr_state, nx_state: state;
begin
-----------lower section--------------
process (clock, reset)
begin
   if (reset = '1')  then
      pr_state <= state0;
    elsif (clock'event and clock='1') then
      pr_state <= nx_state;
    end if;
end process;
```

两个进程并发执行

```
-----------upper section--------------
process (input, pr_state)
 begin
      case pr_state is
      when state0=>
            if  (input=...)  then
               output <= <value>;
               nx_state<=state1;
            else  ....;
            end if;
       when  state1=>
            if  (input=...)  then
               output <= <value>;
               nx_state<=state2;
            else  ....;
            end if;
          ......
          end case;
end process;
end <arch_name>
```

例8.1 BCD计数器

- **功能描述**：状态转移图



- **特点分析**：摩尔型状态机
- **设计分析**：状态较多，枚举很不方便，仅作为例子参考，不推荐实际应用。

代码实现：

```
library ieee;
use ieee.std_logic_1164.all;
entity counter is
    port (                          --无输入数据--
          rst, clk: in std_logic;
        count: out std_logic_vector(3 downto 0));
end counter;
architecture state_machine of counter is
  type state is (zero, one, two, three, four, five,
six, seven, eight, nine);
  signal pr_state, nx_state: state;
begin
-----------lower section-
process (clk, rst)
begin
   if (rst = '1')  then
      pr_state <= zero;
    elsif (clk'event and clock='1') then
      pr_state <= nx_state;
    end if;
end process;
```

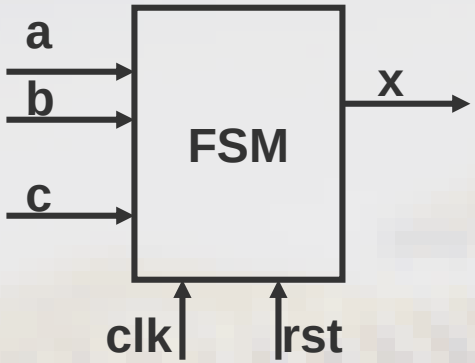
两个进程
并发执行

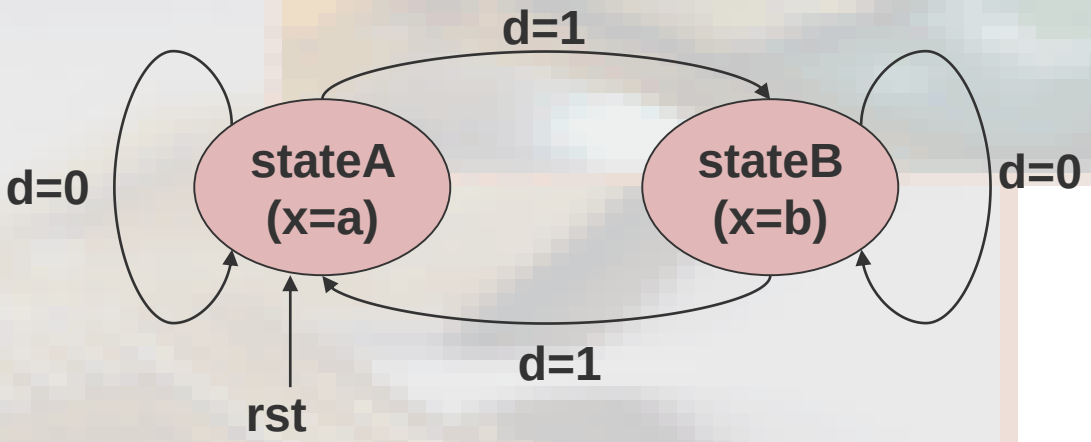
```
-----------upper section---------------
process ( pr_state)            ---无输入数据---
begin
      case pr_state is
      when zero=>
                count <= "0000";
                nx_state<=one;
       when  one=>
                count <= "0001";
                 nx_state<=two;
       when   two=>
                 count<="0010";
                 nx_state<=three;
          .....
         when  nine=>
                  count<="1001";
                   nx_state<=zero;
        end case;
end process;
end state_machine;
```

所需寄存器个数：　上限$\lceil \log_2 10 \rceil = 4$

例8.2　简单的FSM#1

功能描述：





代码实现：

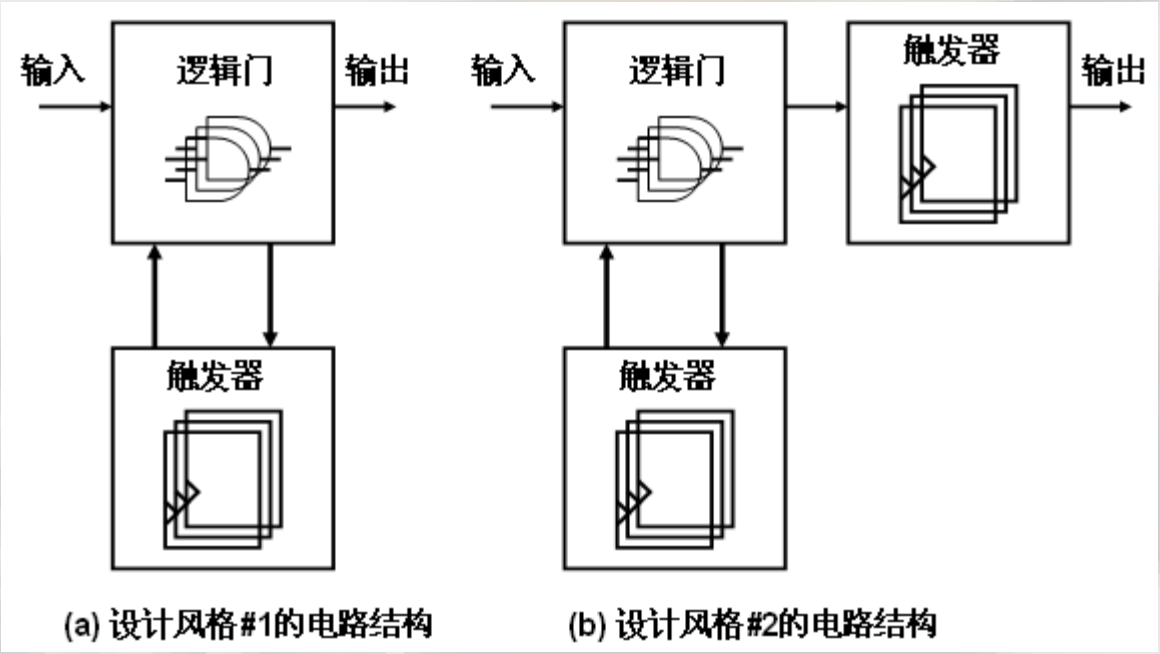
```
entity simple_fsm is
   port ( a, b, c, d, rst, clk: in BIT;
          x: out BIT);
end simple_fsm;
architecture simple_fsm of  simple_fsm is
  type state is (stateA, stateB);
  signal pr_state, nx_state: state;
begin
------------lower section---------------
process (clk, rst)
begin
   if (rst = '1')  then
      pr_state <= stateA;
    elsif (clk'event and clock='1') then
      pr_state <= nx_state;
    end if;
end process;
```

```
------------upper section---------------
process ( a, b, c, d, pr_state)
begin
      case pr_state is
      when stateA=>
              x <=a;
              if (d='1') then nx_state<=stateB;
              else nx_state<=stateA;
              end if;
       when  stateB=>
               x<=b;
               if (d='1') then nx_state<=stateA;
               else nx_state<=stateB;
               end if;
          end case;
end process;
end simple_fsm;
```

所需寄存器个数： 1个，用于存储两个状态编码。

8.3 设计风格#2

在很多应用中（如波形整齐、流水线技术等），需要同步的寄存器输出，即需先使用寄存器存储起来，然后在时钟边沿时才进行更新，如图b：



(a) 设计风格#1的电路结构　　(b) 设计风格#2的电路结构

设计风格#2的状态机模板——使用辅助信号如temp

```
library ieee;
use ieee.std_logic_1164.all;
entity <entity_name> is
    port (input: in <data_type>;
            reset, clock: in std_logic;
           output: out <data_type>);
end <entity_name>;
architecture <arch_name> of <entity_name> is
  type state is (state0, state1, state2, state3, ...);
  signal pr_state, nx_state: state;
  signal temp: <data_type>;
begin
-----------lower section-------
process (clock, reset)
begin
   if (reset = '1')  then
      pr_state <= state0;
    elsif (clock'event and clock='1') then
      output<=temp;
      pr_state <= nx_state;
    end if;
end process;
```

temp信号将输出结果存储起来，只有当所需时钟边沿到来时才被赋值给输出端口

```
-----------upper section---------------
process (pr_state)      ----Mealy型状态机
begin
      case pr_state is
      when state0=>
            temp <= <value>;
            if (condition) then nx_state<=state1;
             ....;
            end if;
       when  state1=>
            temp <= <value>;
            if  (condition)  then nx_state<=state2;
            ....;
            end if;
          .....
        end case;
end process;
end <arch_name>
```

例8.3：用FSM#2实现例8.2：　--同步输出

```
entity simple_fsm is
   port ( a, b, c, d, rst, clk: in BIT;
          x: out BIT);
end simple_fsm;
architecture simple_fsm of  simple_fsm is
  type state is (stateA, stateB);
  signal pr_state, nx_state: state;
  signal temp: BIT;
begin
------------lower section---------------
process (clk, rst)
begin
   if (rst = '1')  then
     pr_state <= stateA;
   elsif (clk'event and clock='1') then
     pr_state <= nx_state;
     x<=temp;
   end if;
end process;
```

```
------------upper section---------------
process ( a, b, c, d, pr_state)
begin
     case pr_state is
     when stateA=>
              temp<=a;
              if (d='1') then nx_state<=stateB;
              else nx_state<=stateA;
              end if;
      when  stateB=>
               temp<=b;
               if (d='1') then nx_state<=stateA;
               else nx_state<=stateB;
               end if;
         end case;
end process;
end simple_fsm;
```

所需寄存器个数：　2个，一个用于存储两个状态编码，另一个用于存储同步输出结果。

8.4 状态机的编码风格—二进制码和独热码

（1）状态机编码的概念：在设计状态机时，需要对状态机的状态进行编码，从而决定需要使用的寄存器数目。

（2）状态机编码的方式： 二进制编码、一位热独码、两位独热码或其它编码方式，**默认的方式是二进制编码**。

例：8状态FSM的状态编码

编码风格			
状态	二进制	双热	独热
State0	000	00011	00000001
State1	001	00101	00000010
State2	010	01001	00000100
State3	011	10001	00001000
State4	100	00110	00010000
State5	101	01010	00100000
State6	110	10010	01000000
State7	111	01100	10000000

（3）不同的编码方式所需要的触发器数目和组合逻辑资源是不同的：

- 二进制编码：需要寄存器数目最少，但所需组合逻辑最多，速度最慢；这些特性对于PAL和门阵列结构是可以接受的。但是因为FPGA具有许多触发器和较少的组合逻辑资源，高编码的状态变量会导致低的FPGA速度和密度的实现效率。
- 一位独热编码（One-Hot Encoding）：就是使每个状态占用状态寄存器的一位。这种编码方法看起来好像很浪费资源，例如，对于一位热独码编码来说，一个具有16个状态的在限状态机需要16个触发器，但如果使用二进制编码，则只需要4个触发器。但是，一位有效编码方法可以简化组合逻辑和逻辑之间的内部连接。一位热独码编码可以产生较小的并且更快的有效状态机。这对于顺序逻辑资源比组合逻辑资源更丰富的可编程ASIC来说，是比较有效的编码方式。特别是对于FPGA结构来说，一位热独码编码是最好的状态编码方式。另外，One-hot编码所需的 组合逻辑最少，触发器最多，工作时钟频率可以做到最高。

•双热码：介于二进制编码和独热编码之间，每一次状态变化都会带来两个位的跳变，使用n个寄存器可以实现对n(n-1)/2个状态进行编码。

•Gray码：就面积与速度的折中考虑来说是最好的选择,当然Gray码还有其他很多好的特性,暂时不属于这次讨论的范畴。

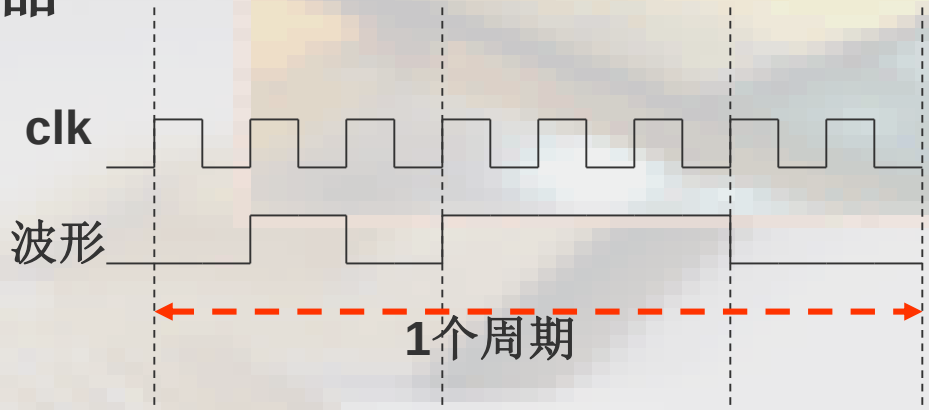
•其他编码方式

（4）状态机编码方式的选择：一般的综合工具对状态机进行综合时都可以让用户对这三种编码进行选择。对于大的基于FPGA的状态机的实现来说，一位有效编码是最佳的方法。对于较少状态的有限状态机（小于8个状态），二进制编码可能更有效。为了提高状态机的设计性能，可以将大的状态机（大于32个状态）分为几个小的状态机，每个状态机都使用合适的编码方式。

第9章 部分习题讲解（1）

例9.9中“信号发生器”

- 功能描述：

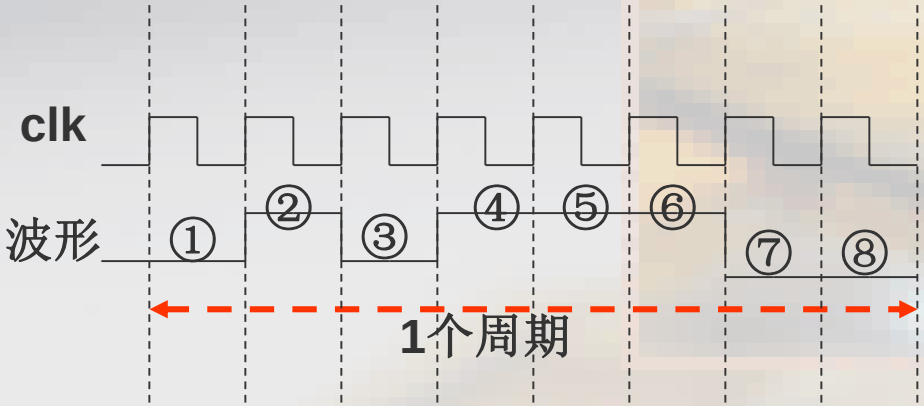


- 特点分析：波形在clk的上升沿才发生跳变，是与例8.6的最大不同之处。
- 设计方法：

1、FSM方法；

2、传统的方法：周期计数。

采用FSM方法进行设计：



设计思路：

①使用8个状态的FSM，并使用模8计数器的计数值代表这8个状态。当count为0时，进入第一个状态，将输出电平置为低电平；当count为1时，进入第二个状态，将输出电平置为高电平；以此类推，即可得图中输出波形。

②另外，由于信号发生器中不允许出现毛刺，须使用FSM设计风格#2，将输出存储到寄存器中。

实现代码：

```
library ieee;
use ieee.std_logic_1164.all;
entity signal_gen is
    port (clk: in std_logic;
          wave: out std_logic);
end signal_gen;
architecture fsm of signal_gen is
  type states is (zero, one, two, three, four,
                  five, six, seven );
  signal present_state, next_state: states;
  signal temp: std_logic;
begin
-----------lower section--------------
process (clk)
begin
    if (clk'event and clk='1') then
     wave<=temp;
     present_state <= next_state;
    end if;
end process;
```

```
-----------upper section of out1-----------
process (present_state)
begin
      case present_state is
      when zero=>temp<='0'; next_state<=one;
      when one=>temp <= '1';next_state<=two;
      when  two=>temp<='0';next_state<=three;;
      when three=>temp<='1';next_state<=four;
      when four=>temp<='1';next_state<=five;
      when five=>temp<='1';next_state<=six;
      when six=>temp<='0';next_state<=seven;
      when seven=>temp<='0';next_state<=zero;
      end case;
end process;
end fsm;
```

寄存器数量：$\log_2 8+1=4$.

（3个用于计数器，1个用于存储输出电平值。）

采用传统方法进行设计

将计数器和输出波形都在时钟信号上升沿时进行赋值。

```
library ieee;
use ieee.std_logic_1164.all;
entity signal_gen is
   port (clk: in BIT;
         wave: out BIT);
end signal_gen;
architecture arch1 of signal_gen is
  begin
process (clk)
    variable count: integer range 0 to 7;
begin
    wait until (clk'event AND clk='1');
    case count is
        when 0=>wave<='0';
        when 1=>wave<='1';
        when 2=>wave<='0';
        when 3=>wave<='1';
        when 4=>wave<='1';
        when 5=>wave<='1';
        when 6=>wave<='0';
        when 7=>wave<='0';
   end case;
      count:=count+1;
   end process;
end arch1;
```

第8章　思考题

1、按输入的相关性，状态机的种类有哪些？

2、标准的FSM设计模板包括几个部分？

3、FSM的设计风格#1和设计风格#2有何区别？

4、FSM有哪些主要的编码方式？

作业：

1、改写例8.4为检测连续4个‘1’的状态机，画出状态转移图，并写出VHD代码，写在纸上上交。代码的正确性及其激励代码在下节上机课上要检查；

2、预习第9章，准备上机。