

# 模拟与数字电路

## Analog and Digital Circuits



课程主页 扫一扫

第十二讲：状态机的应用

Lecture 12: **Finite State Machine - II**

主 讲：陈 迟 晓

Instructor : Chixiao Chen

# 提纲

- 复习
  - 状态机模型有哪两种，他们的区别是什么？
- 基于Verilog的状态机设计流程
- 序列检测电路
- 按键消抖动电路
- 状态化简



# 基于Verilog的状态机通用设计流程

- 设计步骤

- 确定状态的逻辑功能
- 画出状态转移图
- 符号化的状态转移表（真值表）
- 对于状态进行合并，简化和编码
- 利用Verilog描述转移逻辑关系

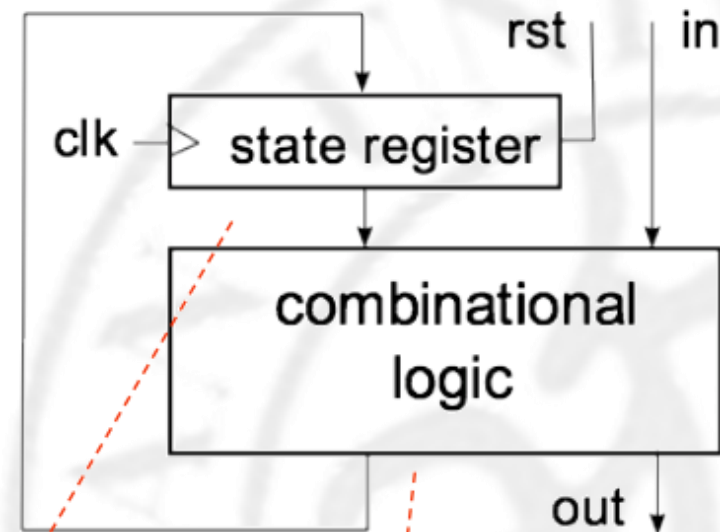
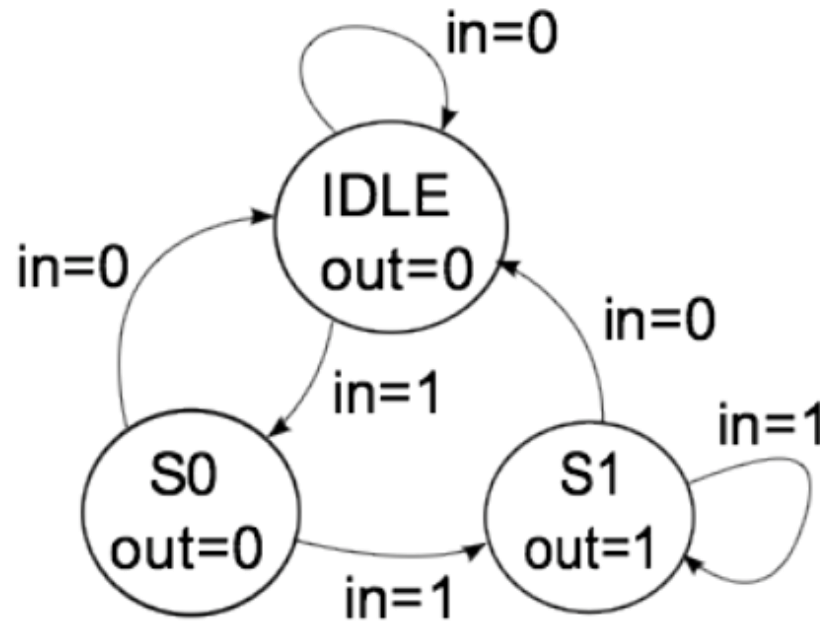
**风格良好**的状态机Verilog具有以下特点：

- ✓ 在Verilog中利用变量代表状态编码
- ✓ 在always块中使用阻塞赋值实现组合逻辑
- ✓ 利用case语句完成转移表与Verilog的直接转化

# 状态转移与预计硬件实现

## Implementation Circuit Diagram

### State Transition Diagram



*Holds a symbol to keep value and next state based on input track of which bubble the FSM is in.*

*CL functions to determine output and current state.*

*$out = f(in, \text{current state})$*

*$next\ state = f(in, \text{current state})$*

画出上述状态转移图对应的表

# 状态机的Verilog实现 - I

## 状态转移代码

```
reg [ 1:0 ] state_reg ,
state_next ;
always @ (posedge clk ,
posedge reset )
if ( reset )
    state_reg <= IDLE;
else
    state_reg <= state_next ;
```

```
module FSM1(clk, rst, in, out);
input clk, rst;
input in;
output out;
```

*Must use reset to force  
to initial state.*

*reset not always shown in STD -----*

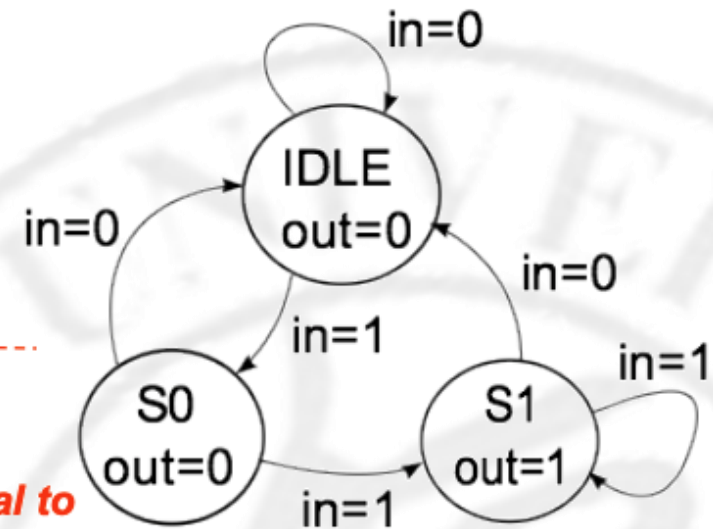
```
// Defined state encoding:
localparam IDLE = 2'b00;
localparam S0 = 2'b01;
localparam S1 = 2'b10;
```

*Constants local to  
this module.*

```
reg out;
reg [1:0] next_state;
wire [1:0] present_state;
```

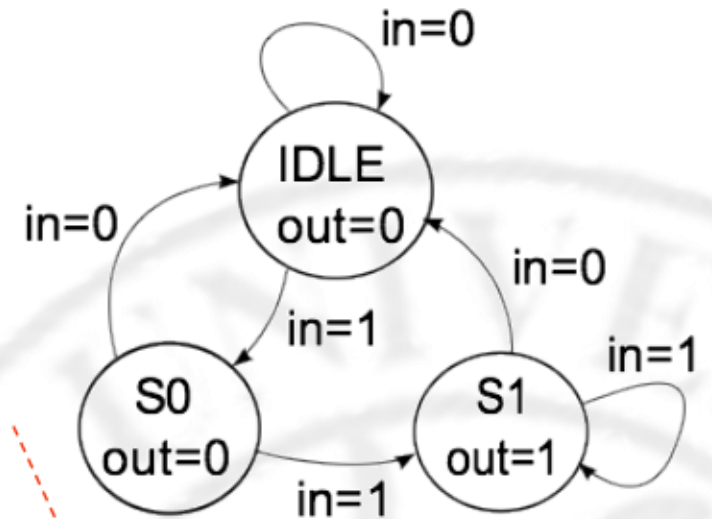
*out not a register, but assigned in always block  
Combinational logic  
signals for transition.*

```
// state register
REGISTER_R #(.N(2), .INIT(IDLE)) state
(.q(present_state), .d(next_state), .rst(rst));
```



# 状态机的Verilog实现 - II

```
// always block for combinational logic portion
always @(present_state or in)
case (present_state)
// For each state def output and next
  IDLE  : begin
    out = 1'b0;
    if (in == 1'b1) next_state = S0;
    else next_state = IDLE;
  end
  S0    : begin
    out = 1'b0;
    if (in == 1'b1) next_state = S1;
    else next_state = IDLE;
  end
  S1    : begin
    out = 1'b1;
    if (in == 1'b1) next_state = S1;
    else next_state = IDLE;
  end
  default: begin
    next_state = IDLE;
    out = 1'b0;
  end
endcase
endmodule
```



**Each state becomes  
a case clause.**

**For each state define:  
Output value(s)  
State transition**

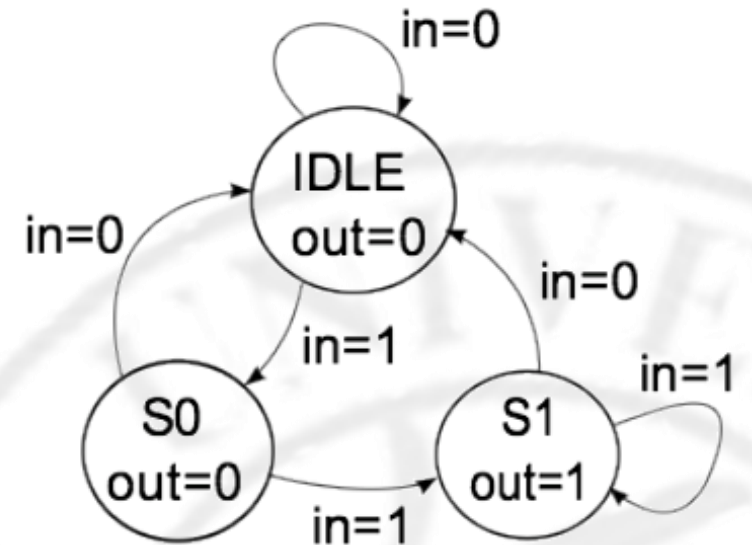
**Use "default" to cover unassigned state. Usually  
unconditionally transition to reset state.**

这是moore模型还是  
Mealy模型

# 状态机的Verilog实现 - III

## 阻塞赋值的Verilog代码优化

```
always @*          * for sensitivity list
begin
  next_state = IDLE;
  out = 1'b0;      Normal values: used unless
                    specified below.
  case (state)
    IDLE : if (in == 1'b1) next_state = S0;
    S0    : if (in == 1'b1) next_state = S1;
    S1    : begin
              out = 1'b1;
              if (in == 1'b1) next_state = S1;
            end
    default: ;
  endcase
end
Endmodule
```



**Within case only need to specify exceptions to the normal values.**

**Note: The use of “blocking assignments” allow signal values to be “rewritten”, simplifying the specification.**



# 状态机设计实例－序列检测器设计

序列检测器可用于检测一组或多组由二进制码组成的脉冲序列信号，当序列检测器连续收到一组串行二进制码后，如果这组码与检测器中预先设置的码相同，则输出1，否则输出0。

**关键步骤：**正确码的接收必须是连续的，要求检测器必须记住前一次的正确码及正确序列，直到在连续的检测中所收到的每一位码都与预置数的对应码相同。

我们利用Moore状态机和Mealy状态机来分别实现对输入序列数"1101"的检测



# 状态机设计实例 – 序列检测器设计

## Moore状态机序列检测器设计\_1

### 解题分析:

如果现态是s0，输入为0，那么下一状态还是停留在s0；如果输入1，则转移到状态s1。

在状态s1，如果输入为0，则回到状态s0；如果输入为1，那么就转移到s2。

在s2状态，如果输入为1，则停留在状态s2；如果输入为0，那么下一状态为s3。

在s3状态，如果输入为1，则转移到状态s4，输出1；如果输入为0，则返回状态s0。

在s4状态，如果输入为0，回到初始状态s0；如果输入为1，下一状态为s1。

### 定义以下状态

输入

s0: 未检测到 ‘1’

s1: 收到 ‘1’

s2: 收到 ‘11’

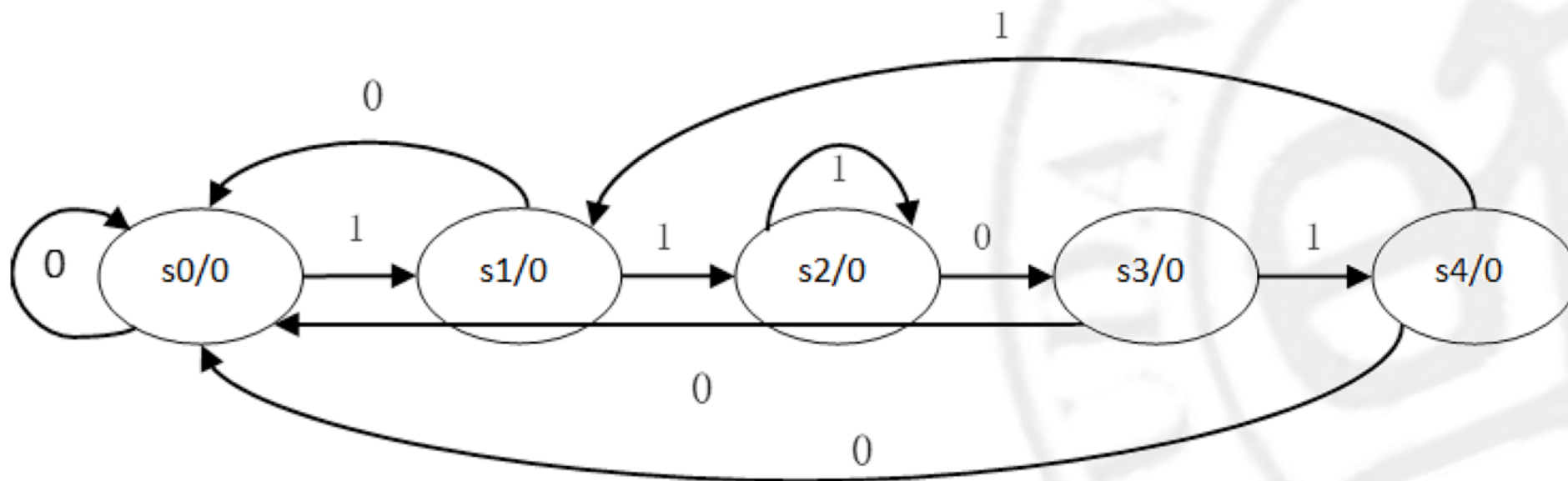
s3: 收到 ‘110’

s4: 收到 ‘1101’

# 状态机设计实例－序列检测器设计

## Moore状态机序列检测器设计\_2

状态转移图：



# 状态机设计实例 – 序列检测器设计

## Moore状态机序列检测器设计\_3

状态声明代码

```
localparam [ 2:0 ]  
    s0 = 3'b000 ,  
    s1 = 3'b001 ,  
    s2 = 3'b010 ,  
    s3 = 3'b011 ,  
    s4 = 3'b100;
```

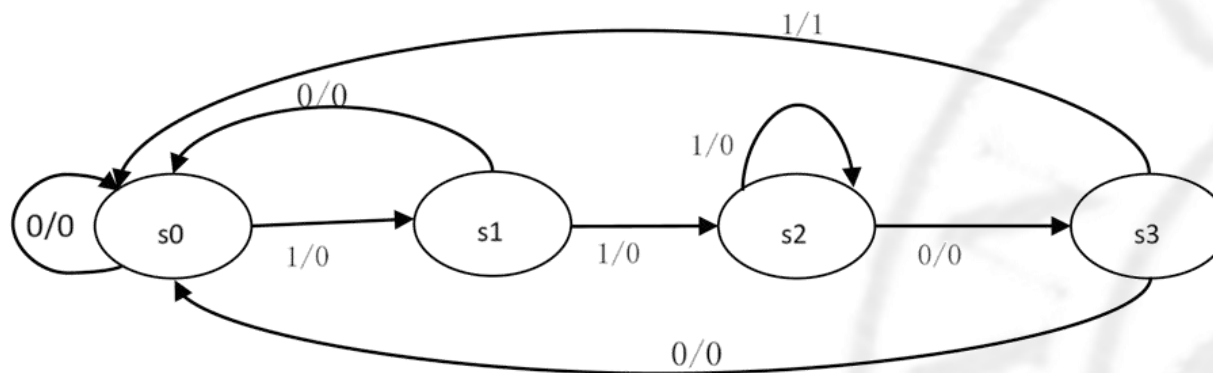
次级逻辑代码

```
case (cs )  
    s0:  
        if (din == 1'b1 ) nst = s1 ;  
    else nst = s0 ;  
    s1:  
        if (din == 1'b1 ) nst = s2 ;  
    else nst = s0 ;  
    s2:  
        if (din == 1'b0 ) nst = s3 ;  
    else nst = s2 ;  
    s3:  
        if (din == 1'b1 ) nst = s4 ;  
    else nst = s0 ;  
    s4:  
        if (din == 1'b0 ) nst = s1 ;  
    else nst = s0 ;  
    default : nst = s0 ;  
endcase
```

# 状态机设计实例 – 序列检测器设计

## Mealy状态机序列检测器设计\_1

状态转移图：



对比Mealy状态机与Moore状态机的状态图可知：

Moore状态机的检测结果输出是与时钟同步的；而Mealy状态机的检测结果输出是异步的，当输入发生变化时，输出就立即变化。因此Mealy状态机的输出比Moore状态机状态的输出提前一个周期。

# 状态机设计实例－序列检测器设计

## Mealy状态机序列检测器设计\_2

状态声明代码

```
localparam [ 1:0 ]  
    s0 = 2'b00 ,  
    s1 = 2'b01 ,  
    s2 = 2'b10 ,  
    s3 = 2'b11;
```

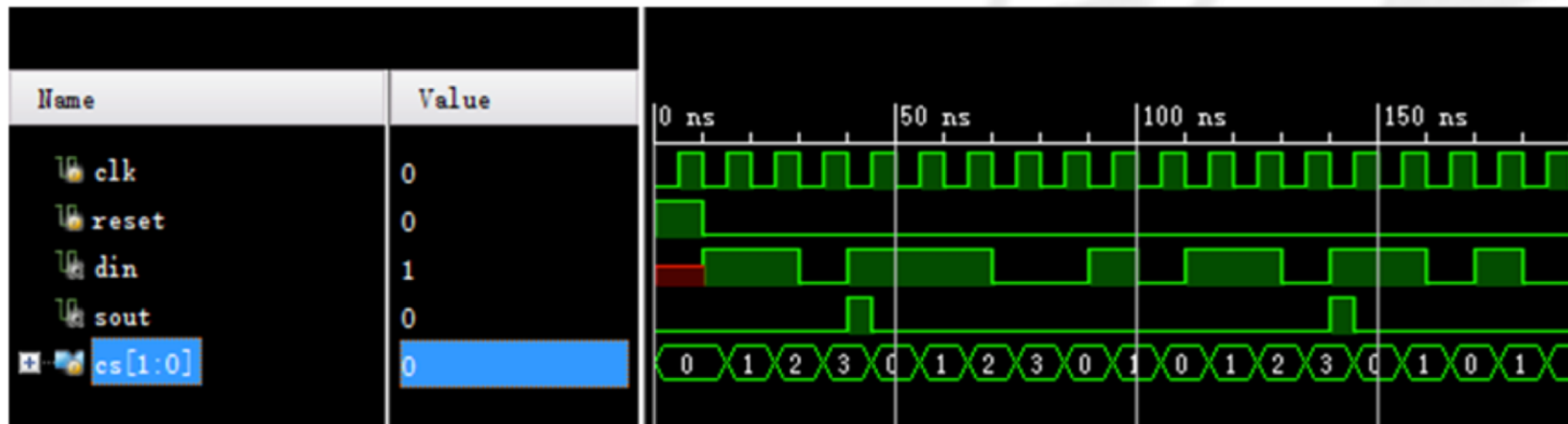
次级逻辑代码

```
case (cs )  
    s0:  
        if (din == 1'b1 ) nst = s1 ;  
    else nst = s0 ;  
    s1:  
        if (din == 1'b1 ) nst = s2 ;  
    else nst = s0 ;  
    s2:  
        if (din == 1'b0 ) nst = s3 ;  
    else nst = s2 ;  
    s3: nst = s0 ;  
    default : nst = s0 ;  
endcase
```

# 状态机设计实例－序列检测器设计

## Mealy状态机序列检测器设计\_3

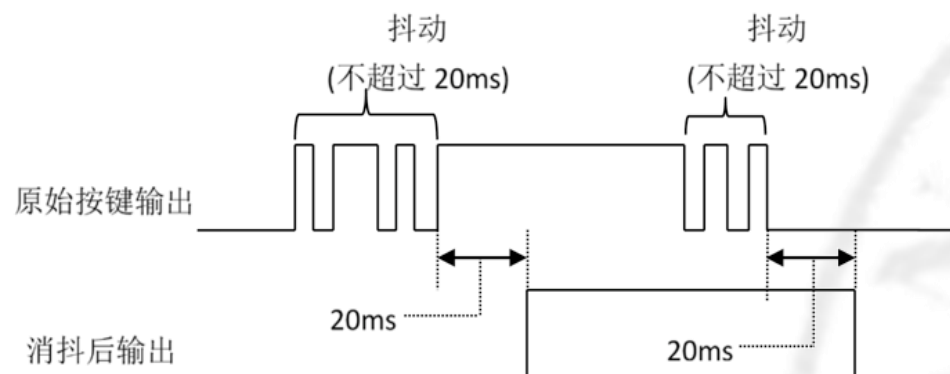
仿真结果：





# 状态机设计实例 – 按键消抖电路设计

## 按键消抖电路设计\_1



基于FSM的设计消抖电路，利用一个10ms的非同步定时器和有限状态机，计时器每10ms产生一个滴答使能周期信号，有限状态机利用此信号来确定输入信号是否稳定。

有限状态机将消除时间较短的抖动，当输入信号稳定20ms以后才改变去抖动以后的输出值。

# 状态机设计实例 – 按键消抖电路设计

## 按键消抖电路设计\_2

解题分析：

1. 假定系统的起始态是zero(one) 态，当sw变为1(0) 时，系统转换为wait1\_1态。
2. 当处于wait1\_1态时，有限状态机处于等待状态并将m\_tick置为有效电平态。若sw变为0则表示1值所持续的时间过短有限状态机返回zero态。
3. 这个动作在wait1\_2态和wait1\_3态也将再重复2次。

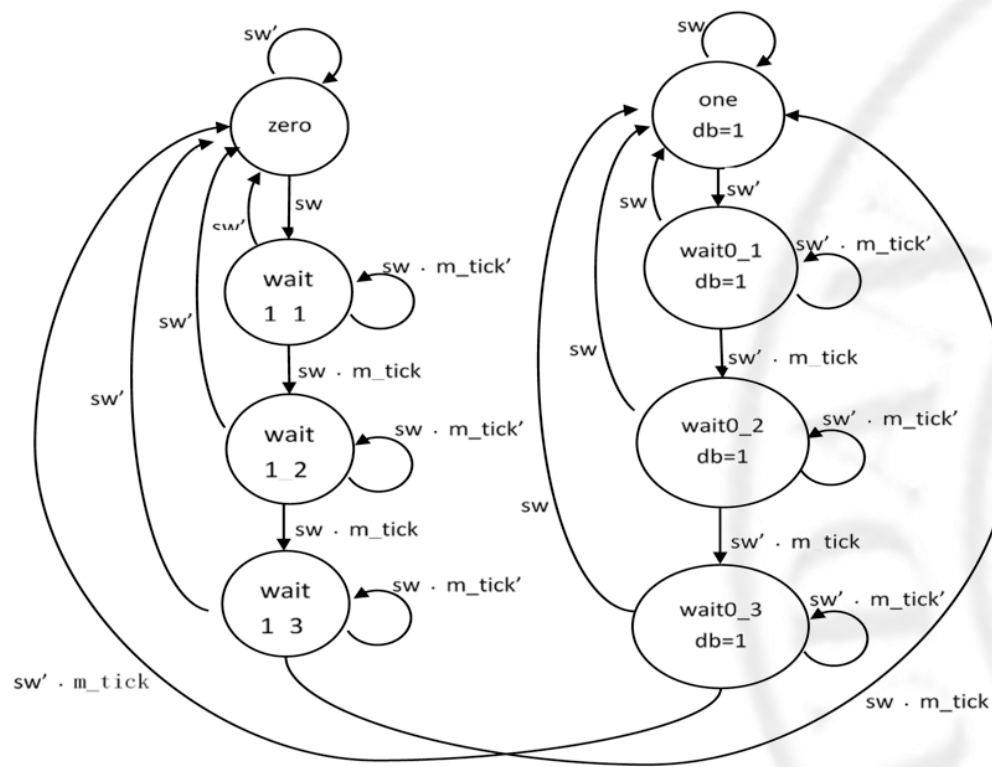
zero态： sw稳定在0值

one态： sw稳定在1值。

# 状态机设计实例 – 按键消抖电路设计

## 按键消抖电路设计\_2

状态转移图：



# 状态机设计实例 – 按键消抖电路设计

## 按键消抖电路设计\_3

### 状态声明代码

```
localparam [ 2:0 ]  
zero   = 3'b000 ,  
wait1_1 = 3'b001 ,  
wait1_2 = 3'b010 ,  
wait1_3 = 3'b011 ,  
one    = 3'b100 ,  
wait0_1 = 3'b101 ,  
wait0_2 = 3'b110 ,  
wait0_3 = 3'b111 ;
```

### 状态转移代码

```
case (state_reg )  
  zero :  
    if (sw )  
      state_next = wait1_1 ;  
  wait1_1 :  
    if ( ~sw )  
      state_next = zero ;  
    else if (m_tick )  
      state_next = wait1_2;  
  wait1_2 :  
    if ( ~sw )  
      state_next = zero ;  
    else if (m_tick )  
      state_next = wait1_3 ;  
  wait1_3 :  
    if ( ~sw )  
      state_next = zero ;  
    else if (m_tick )  
      state_next = one ;  
  one :  
    begin  
      db = 1'b1 ;  
      if ( ~sw )  
        state_next = wait0_1 ;  
    end  
  wait0_1 :
```

```
begin  
  db = 1'b1 ;  
  if(sw)  
    state_next = one ;  
  else if(m_tick )  
    state_next = wait0_2;  
end  
wait0_2 :  
  begin  
    db = 1'b1 ;  
    if(sw)  
      state_next = one ;  
    else if(m_tick )  
      state_next = wait0_3 ;  
  end  
wait0_3 :  
  begin  
    db = 1'b1 ;  
    if(sw)  
      state_next = one ;  
    else if(m_tick )  
      state_next = zero ;  
  end  
  default :state_next = zero ;  
endcase
```

# 状态的化简 – 等价状态

- 状态越少 数字逻辑电路的成本越低
- 等价状态：是指能满足以下条件的两个状态 $S_i$ 和 $S_j$ ，记为 $\{S_i, S_j\}$ 。
  - ①在各种输入取值下，输出完全相同。
  - ②在各种输入取值下，次态满足下列条件之一：
    - a.两个次态完全相同；
    - b.两个次态为其现态本身或交错；
    - c.两个次态为状态对循环中的一个状态对；
    - d.两个次态的某一后续状态对可以合并。

# 状态的化简－隐含表

- 构造隐含表：纵向“缺头”，横向“少尾”
  - ①状态对肯定不等价的，在隐含表相应方格中标注“×”；
  - ②状态对肯定等价的，在隐含表相应方格中标注“√”；
  - ③状态对条件等价的，在隐含表相应方格中标注等价条件。
- 顺序比较。先将隐含表中所有的状态按照一定顺序对照原始状态表逐一进行比较，并将比较结果按上面的约定标注在隐含表中每一个小方格内。
- 确定原始状态表的最大等价类，并画出新表

B				
C				
D				
E				
	A	B	C	D



# 状态化简 – 例题

- 下图是原始状态表

表 5-14 例 5-8 的原始状态表

$S^{n+1} / Y$ $S$	$x$		
		0	1
A		C/0	B/1
B		F/0	A/1
C		D/0	G/0
D		D/1	E/0
E		C/0	E/1
F		D/0	G/0
G		C/1	D/0

B	CF					
C	x	x				
D	x	x	x			
E	BE	AE CF	x	x		
F	x	x	√	x	x	
G	x	x	x	CD <sup>x</sup> DE	x	x
	A	B	C	D	E	F

隐含表

# 状态化简 – 例题

- 最大等价类：隐含表中未打“×”的方格都代表一个等价状态对。全部等价对：{A, B}、{A, E}、{B, E}、{C, F}。
- 因此可得到最大等价类 {A, B, E}、{C, F}、{D}、{G}

- 令  $a=\{G\}$ ,  $b=\{C, F\}$ ,  
 $c=\{A, B, E\}$ ,  $d=\{D\}$   
右图为化简后的状态表

$S^{n+1}/Y$ $x$		
	0	1
$a$	$b/1$	$d/0$
$b$	$d/0$	$a/0$
$c$	$b/0$	$c/1$
$d$	$d/1$	$c/0$