

模拟与数字电路

Analog and Digital Circuits



课程主页 扫一扫

第十七讲：状态机设计基础

Lecture 14: **Design of FSM(Finite State Machine)**

主 讲：陈迟晓

Instructor : Chixiao Chen

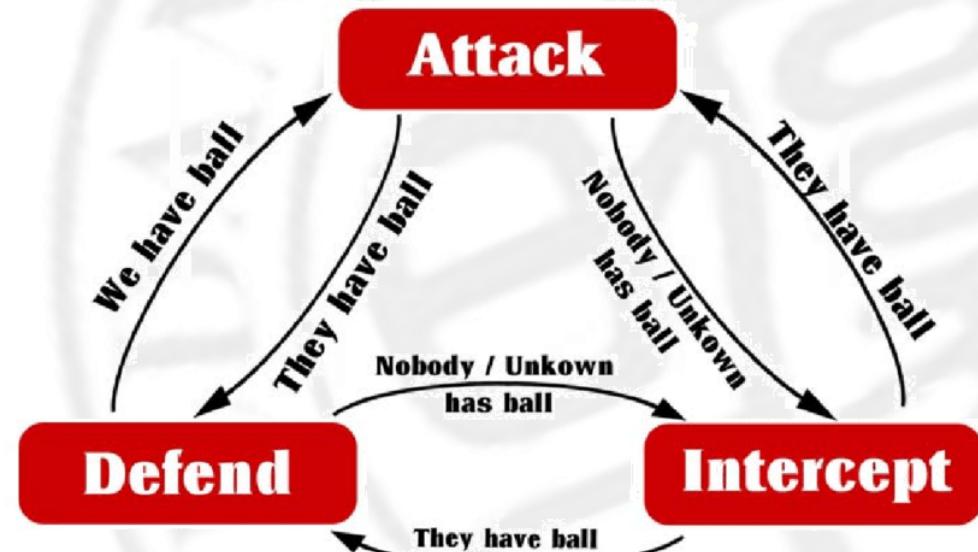
提纲

- 复习
 - DFF的Verilog 代码与组和逻辑代码的区别?
- 状态机基础
- 序列检测器
- 按键消抖动电路

状态机定义

有限状态机（Finite State Machine, FSM）简称状态机，是用来表示系统中的有限个状态及这些状态之间的转移和动作的模型。

这些转移和动作依赖于当前状态和外部输入，它下一步的状态逻辑通常是重新建立的，也称之为随机逻辑。



状态机优点

- **高效的顺序控制模型**

克服了纯硬件数字系统顺序方式控制不灵活的缺点，在其运行方式上类似于控制灵活和方便的CPU，是高速高效控制的首选。

- **容易利用现成的EDA工具进行优化设计**

1. 状态机构建简单，设计方案相对固定，使用HDL综合器可以发挥其强大的优化功能；
2. 性能良好的综合器都具备许多可控或自动优化状态机的功能

- **稳定性**

状态机容易构成良好的同步时序逻辑模块，可用于解决大规模逻辑电路设计中的竞争和冒险现象。

状态机优点(续)

● 高速性能

在高速通信和高速控制方面，状态机更具有其巨大的优势，一个状态机的功能类似于CPU的功能。

● 高可靠性

1. 状态机是由纯硬件电路构成，不存在CPU运行软件过程中许多固有的缺陷；
2. 状态机的设计中能使用各种容错技术
3. 当状态机进入非法状态并从中跳出，进入正常状态的时间短暂，对系统的危害不大。

状态机分类

状态机主要分为两种类型：

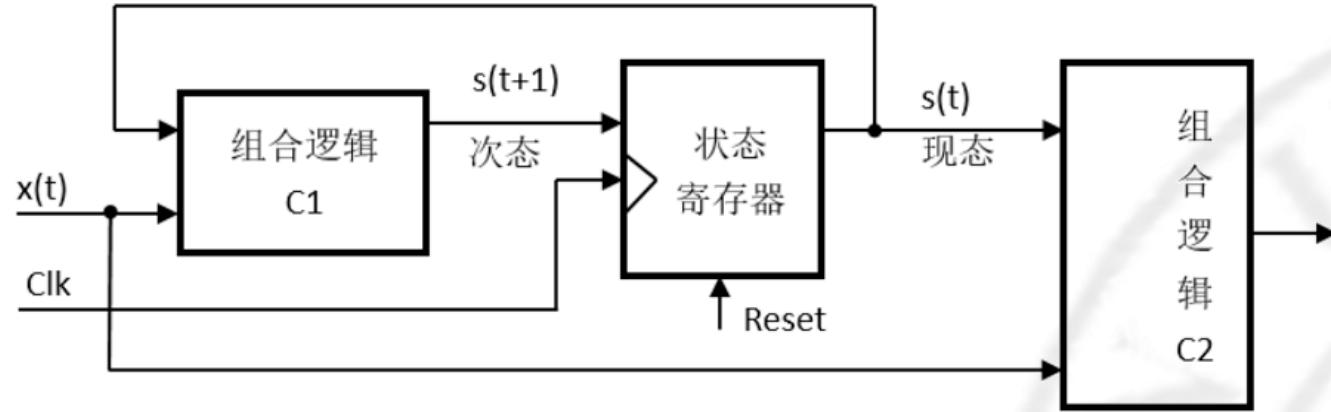
1. Moore型状态机：下一状态只由当前状态决定。即：

次态= f (现状, 输入), 输出= f (现状)；

2. Mealy型状态机：下一状态不但与当前状态有关, 还与当前输入值有关, 即：

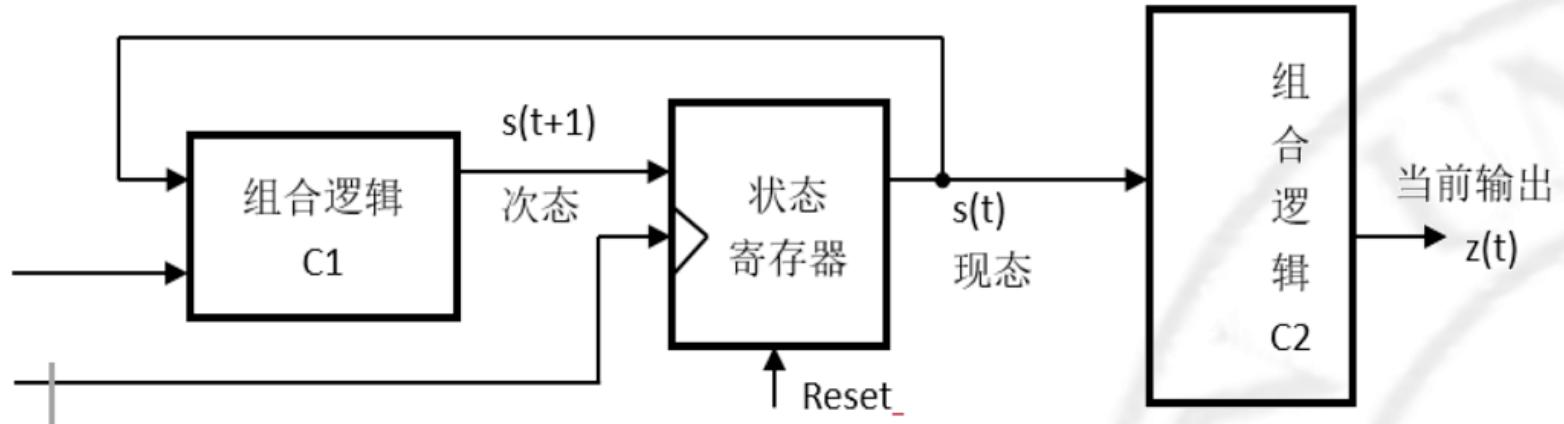
次态= f (现状, 输入), 输出= f (现状, 输入)；

Mealy型状态机



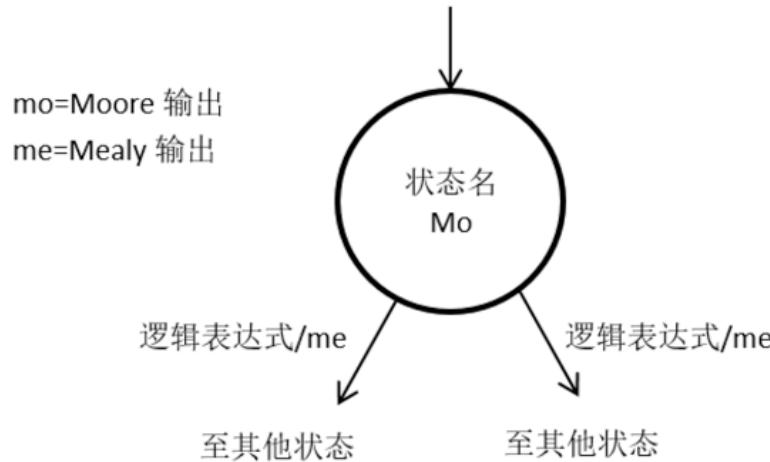
Mealy状态机的输出不仅与当前的状态有关，还与当前的输入有关，即当前的输入和当前的状态共同决定当前的输出。

Moore型状态机



Moore状态机的输出只与当前的状态有关，也就是由当前的状态决定输出，而与此时的输入无关，输入只决定状态机的状态改变，不影响电路最终的输出。

状态机的图表示法：状态转移图



转移状态图组成：

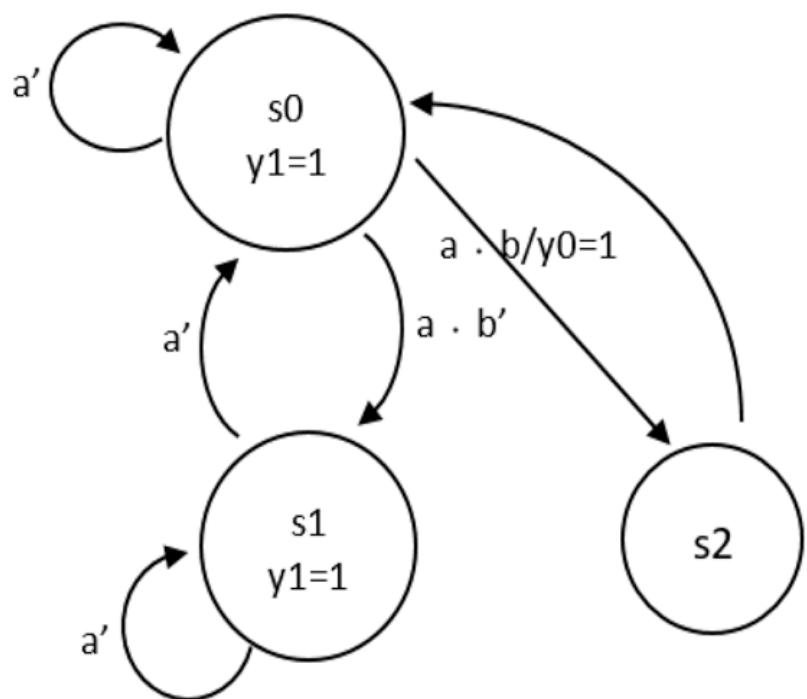
1. 带有标示状态的节点
2. 带有注释的有向弧线

上图中的逻辑表达式由输入信号决定，放在每个转移弧线上，表示状态转移的特定条件。

Moore 型输出值放置在节点内，只由当前状态决定；

Mealy 型输出放置在转换弧线上，由当前状态和外部输入决定。

状态转移图(续)



图中这个有限状态机包含三个状态、两个信号输入、一个Moore输出和一个Mealy输出。

当有限状态机在s0或s1状态时， y_1 输出高电平；

当状态机处于s0状态且a、b均为1的时候， y_0 也为高电平。

有限状态机代码实现

有限状态机和常规时序电路的代码编写对比：

相同点：均是先将状态寄存器拿出，然后将次态逻辑和输出逻辑结合起来并书写相应的代码。

不同点：次态逻辑的代码较为不同，对于有限状态机，次态逻辑单元的代码要遵循状态图或者ASM图的逻辑转移流向。

下面我们重点学习状态机的状态以及次态逻辑的代码编写

有限状态机代码实现(续)

```
// 次态逻辑
always @@
case ( state_reg )
    s0: if ( a )
        if ( b )
            state_next = s2 ;
        else
            state_next = s1 ;
    else
        state_next = s0;
    s1: if ( a )
        state_next = s0;
    else
        state_next = s1;
    s2: state_next = s0;
default: state_next = s0;
endcase
```

左侧是满足次态逻辑的代码，每个状态的代码都要遵循转移状态图的流程来描述。
次态逻辑单元是这个例程的关键。从例程中可以看出，次态（state_next signal）由当前状态(state_reg)和外部输入信号决定。

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

s0: 未检测到 ‘1’

输入

s1: 收到 ‘1’

s2: 收到 ’11 ‘

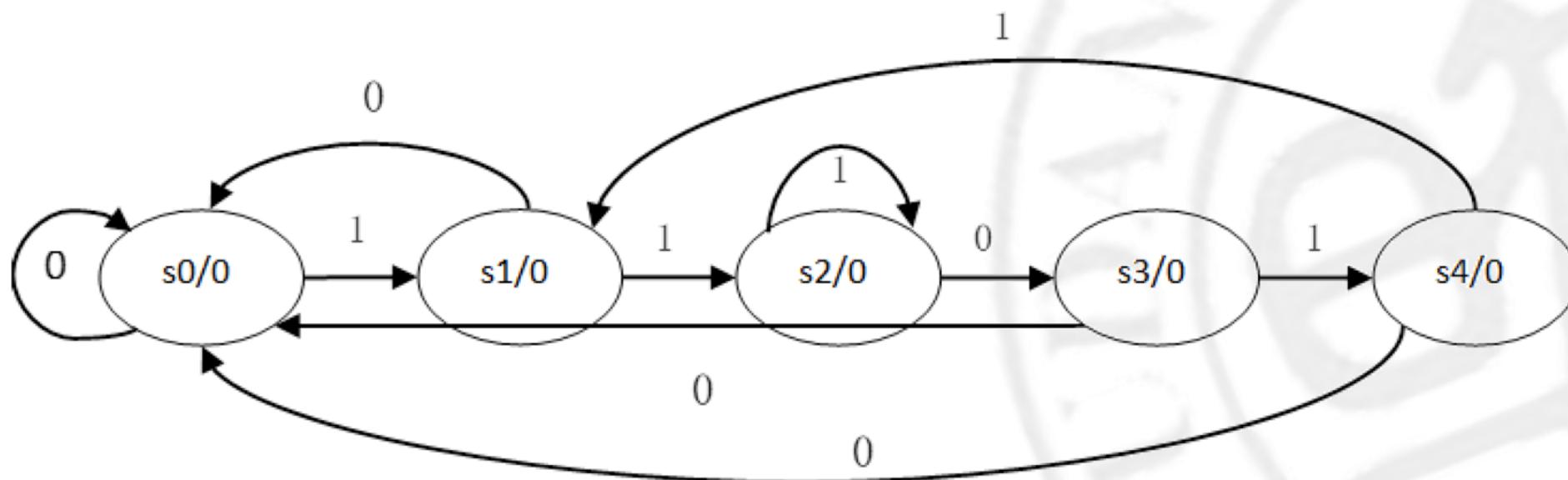
s3: 收到 ‘110’

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;
```

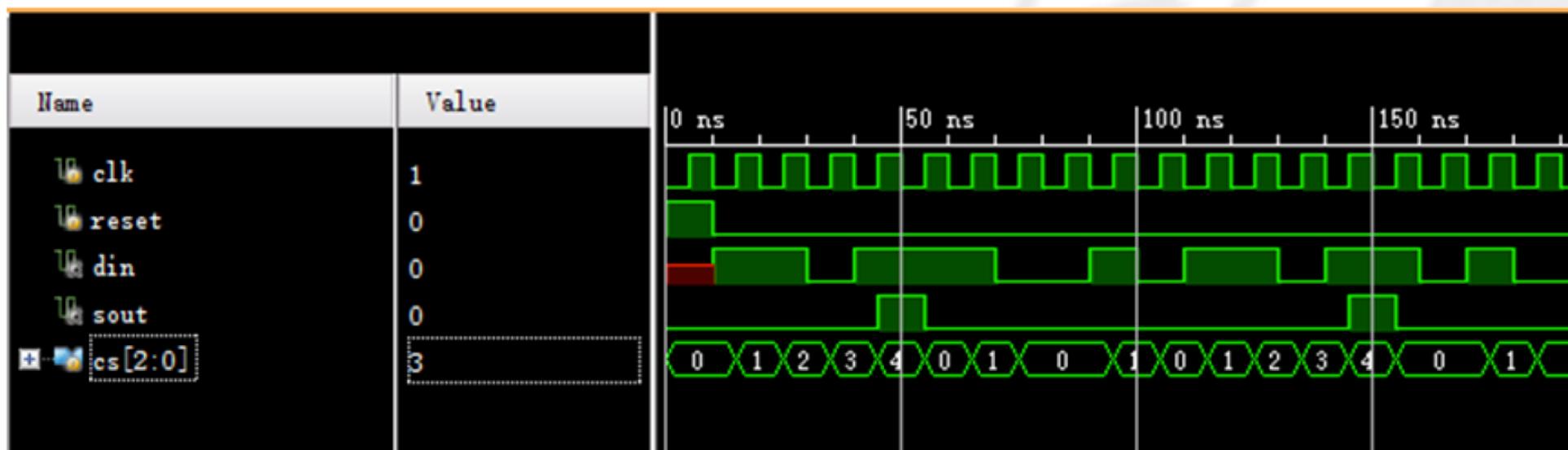
次级逻辑代码

```
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

Moore状态机序列检测器设计_4

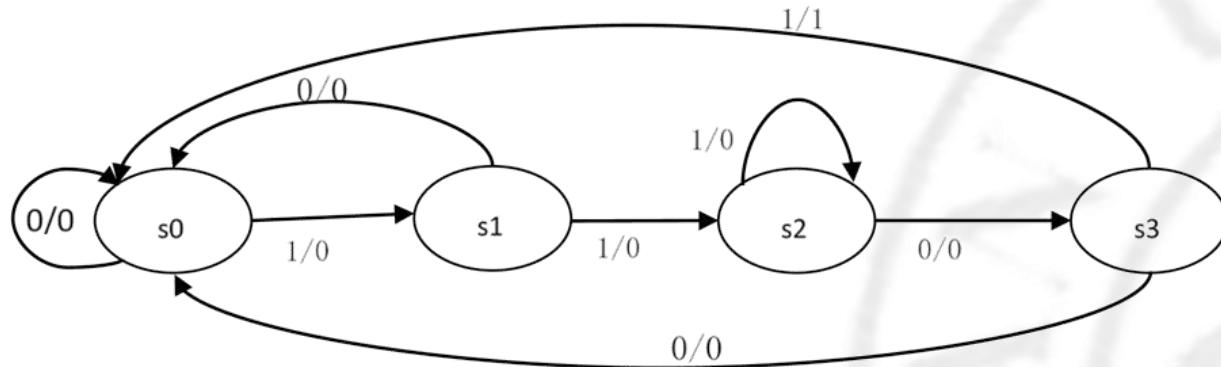
仿真结果：



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;
```

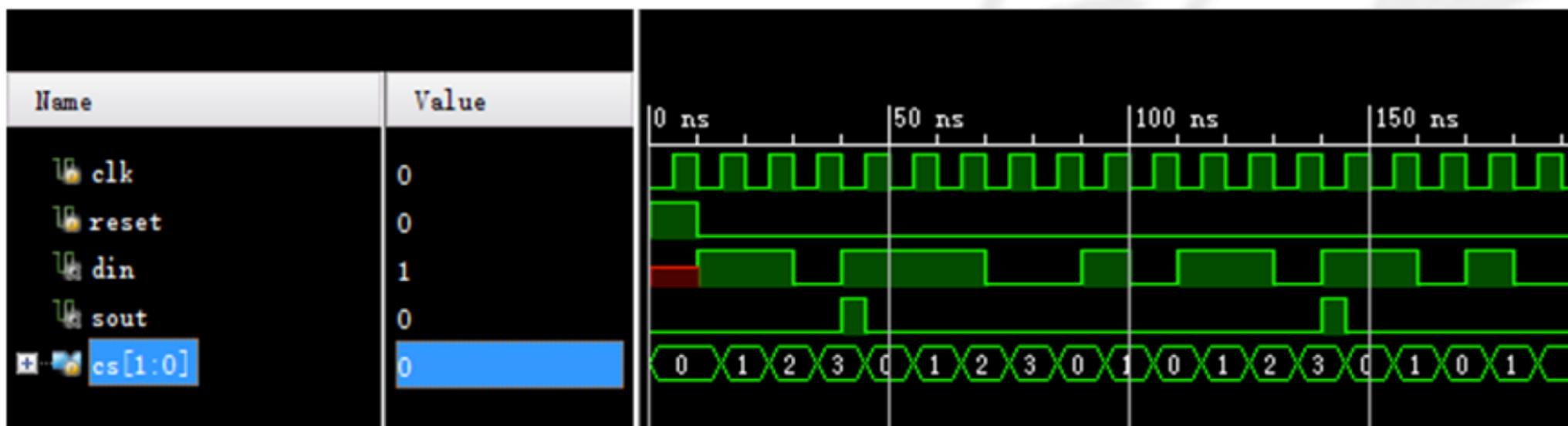
次级逻辑代码

```
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

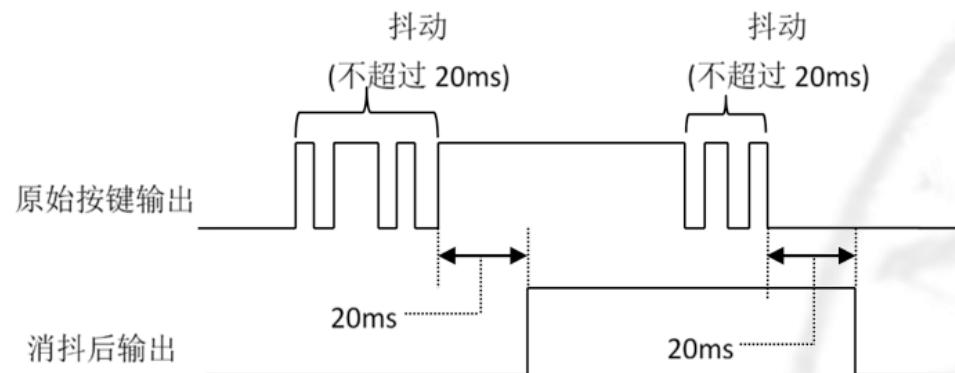
Mealy状态机序列检测器设计_3

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：

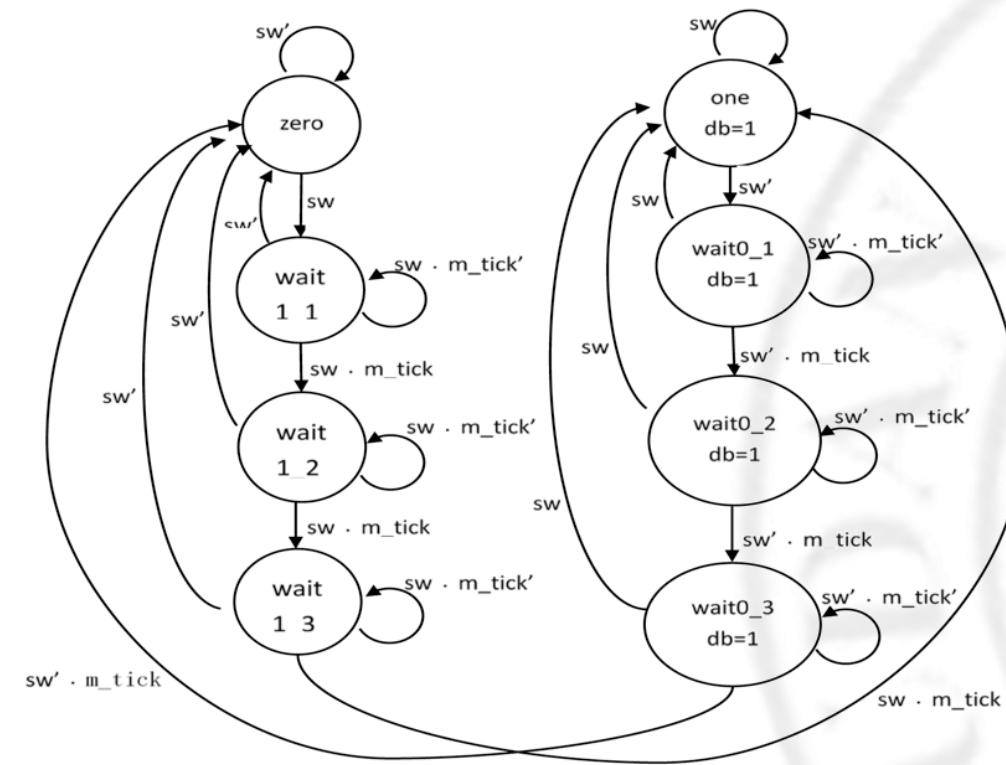
1. 假定系统的起始态是zero(one)态，当sw变为1(0)时，系统转换为wait1_1态。
2. 当处于wait1_1态时，有限状态机处于等待状态并将m_tick置为有效电平态。若sw变为0则表示1值所持续的时间过短有限状态机返回zero态。
3. 这个动作在wait1_2态和wait1_3态也将再重复2次。

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

状态转移代码

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 ;
```

```
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
```