第10章 有限状态机设计

有限状态机

有限状态机（Finite State Machine）又称有限状态自动机或简称状态机，是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。

在数字电路系统中，有限状态机是一种十分重要的时序逻辑电路模块，它对数字系统的设计具有十分重要的作用。

有限状态机是指输出取决于过去输入部分和当前输入部分的时序逻辑电路。一般来说，除了输入部分和输出部分外，有限状态机还含有一组具有“记忆”功能的寄存器，这些寄存器的功能是记忆有限状态机的内部状态，它们常被称为状态寄存器。在有限状态机中，状态寄存器的的下一个状态不仅与输入信号有关，而且还与该寄存器的当前状态有关，因此有限状态机又可以认为是组合逻辑和寄存器逻辑的一种组合。其中，寄存器逻辑的功能是存储有限状态机的内部状态；而组合逻辑有可以分为次态逻辑和输出逻辑两部分，次态逻辑的功能是确定有限状态机的下一个状态，输出逻辑的功能是确定有限状态机的输出。

有限状态机的优点

- 有限状态机结构模式相对简单，设计方案相对固定，有利于综合器优化。
- 容易构成性能良好的同步时序逻辑模块，有利于解决竞争冒险现象。
- 在高速运算和控制方面，状态机有其巨大优势。一个结构体可包含多个状态机，类似于并行运行的多CPU系统。
- 运行速度远远高于CPU。状态机状态变换周期只有一个时钟周期，而且每一状态可以完成许多并行运算和控制操作。
- 在可靠性方面，状态机优势明显。状态机可以使用容错技术；状态机进入非法状态并从中跳出所耗时间极短，通常只需2个时钟周期，约10个ns。CPU一般数十ms。

状态机分类

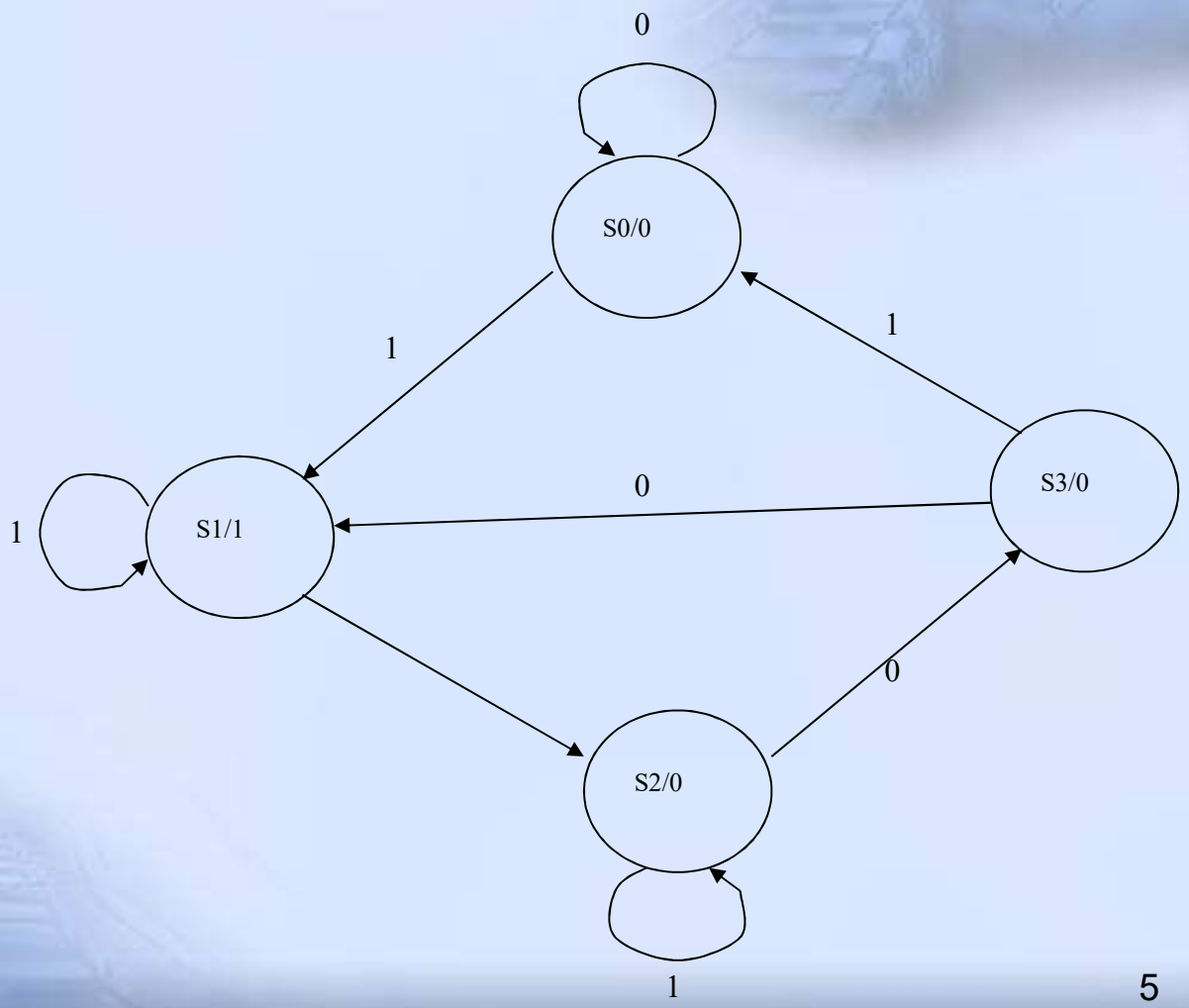
- Melay型：输出是当前状态和所有输入信号的函数。
- Moore型：输出仅是当前状态的函数。

Melay型状态机的输出是输入变化后立即发生变化；而Moore型状态机在输入发生变化后，还需等到时钟到来，时钟使状态发生变化才导致输出变化。因此要多等待一个时钟周期。

设计实例

例1　Moore状态机

状态s0时，输入为0维持s0，输入为1时，下一状态改为s1，不论输入是什么，输出均为0。



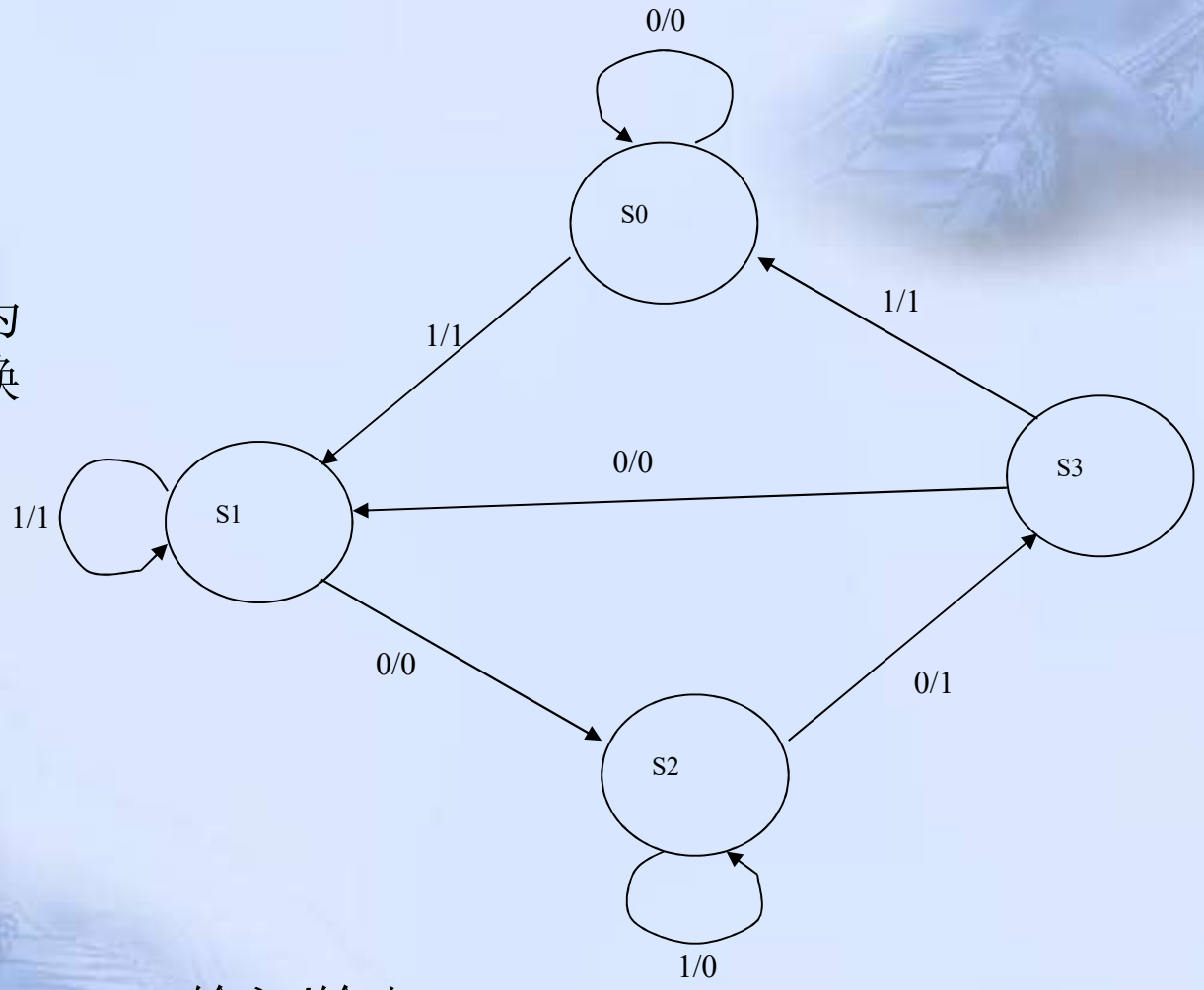
```
module moore_ex1(clk,din,op);
input clk,din;
output op;
reg op;

parameter s0=2'b00,
          s1=2'b01,
          s2=2'b10,
          s3=2'b11;

reg[1:0] presentstate ;
reg[1:0] nextstate ;

 always @(posedge clk)
    presentstate <= nextstate;
```

```
always @(din,presentstate)
 begin
   case (presentstate)
   s0:  begin
      if (din == 0)
        nextstate = s0;
      else
        nextstate = s1;
      op=0;
   end
   s1: begin
     if (din == 1)
       nextstate = s1;
     else
       nextstate = s2;
     op=1;
   end
   s2: begin
        if (din == 1)
          nextstate = s2;
        else
          nextstate= s3;
        op=0;
   end
   s3: begin
        if (din == 1)
          nextstate = s0;
        else
          nextstate = s1;
        op=0;
    end
   default:begin
          nextstate = s0;
          op = 0;
    end
  endcase
  end
  endmodule
```

```
module t_moore_ex1;
  reg clk,din;
  wire op;
  always   #20 clk=~clk;

  initial
  begin
    clk=0;
    din=0;
    #80 din=1;
    #160 din=0;
    #200 din=1;
    #160 din=0;
  end

  moore_ex1 u1(.clk(clk),.din(din),.op(op));
endmodule
```

例2 Mealy状态机

若当前是S3，若输入是0，则输出0且下一个状态是s1，而输入为1则输出为1且下一个状态更换成s0



输入/输出

```
module mealy_ex1(clk,din,op);
input clk,din;
output op;
reg op;

parameter s0=2'b00,
          s1=2'b01,
          s2=2'b10,
          s3=2'b11;

reg[1:0] presentstate ;
reg[1:0] nextstate ;


  always @(posedge clk)
    presentstate <= nextstate;
```

```
always @(din,presentstate)
 begin
   case (presentstate)
   s0:  begin
        if (din == 0)
        begin
          nextstate = s0;
          op = 0;
        end
        else
        begin
          nextstate = s1;
          op =1 ;
        end
   end
```

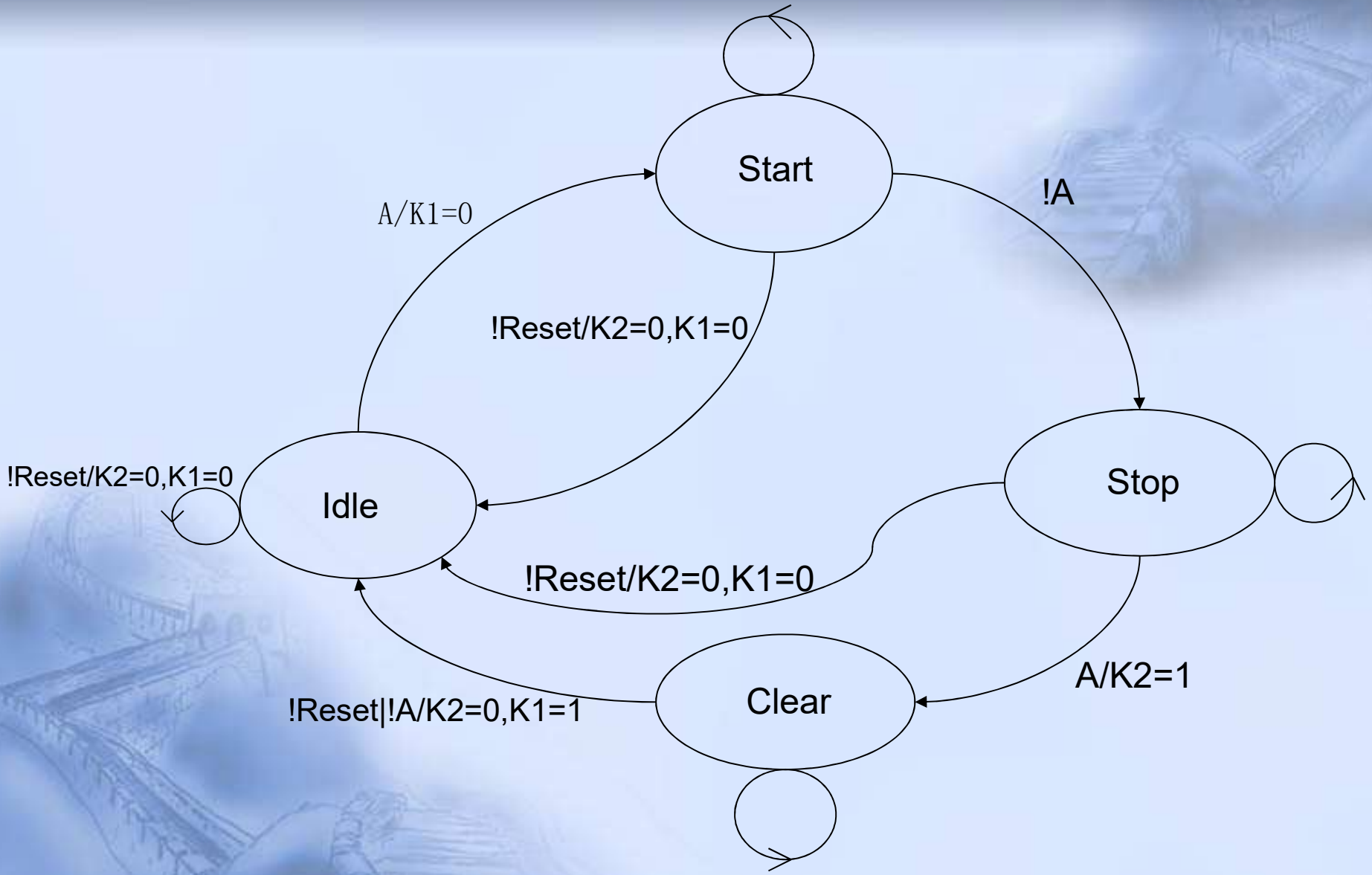
```
s1: begin
      if (din == 1)
      begin
        nextstate = s1;
        op <= 1;
      end
      else
      begin
        nextstate = s2;
        op = 0;
      end
    end
```

```
s2: begin
      if (din == 1)
      begin
        nextstate = s2;
        op = 0;
      end
      else
      begin
        nextstate= s3;
        op = 1;
      end
    end
```

```
s3: begin
      if (din == 1)
      begin
        nextstate = s0;
        op = 1;
      end
      else
      begin
        nextstate = s1;
        op= 0;
      end
    end
```

```
        default:begin
                nextstate = s0;
                op = 0;
            end
    endcase
end

endmodule
```

```
module t_mealy_ex1;
  reg clk,din;
  wire op;
  always   #20 clk=~clk;

  initial
  begin
    clk=0;
    din=0;
    #80 din=1;
    #160 din=0;
    #200 din=1;
    #160 din=0;
  end

  mealy_ex1 u1(.clk(clk),.din(din),.op(op));
endmodule
```


Start
!A
A/K1=0
!Reset/K2=0,K1=0
Idle
!Reset/K2=0,K1=0
Stop
!Reset/K2=0,K1=0
A/K2=1
!Reset|!A/K2=0,K1=1
Clear

状态转移图

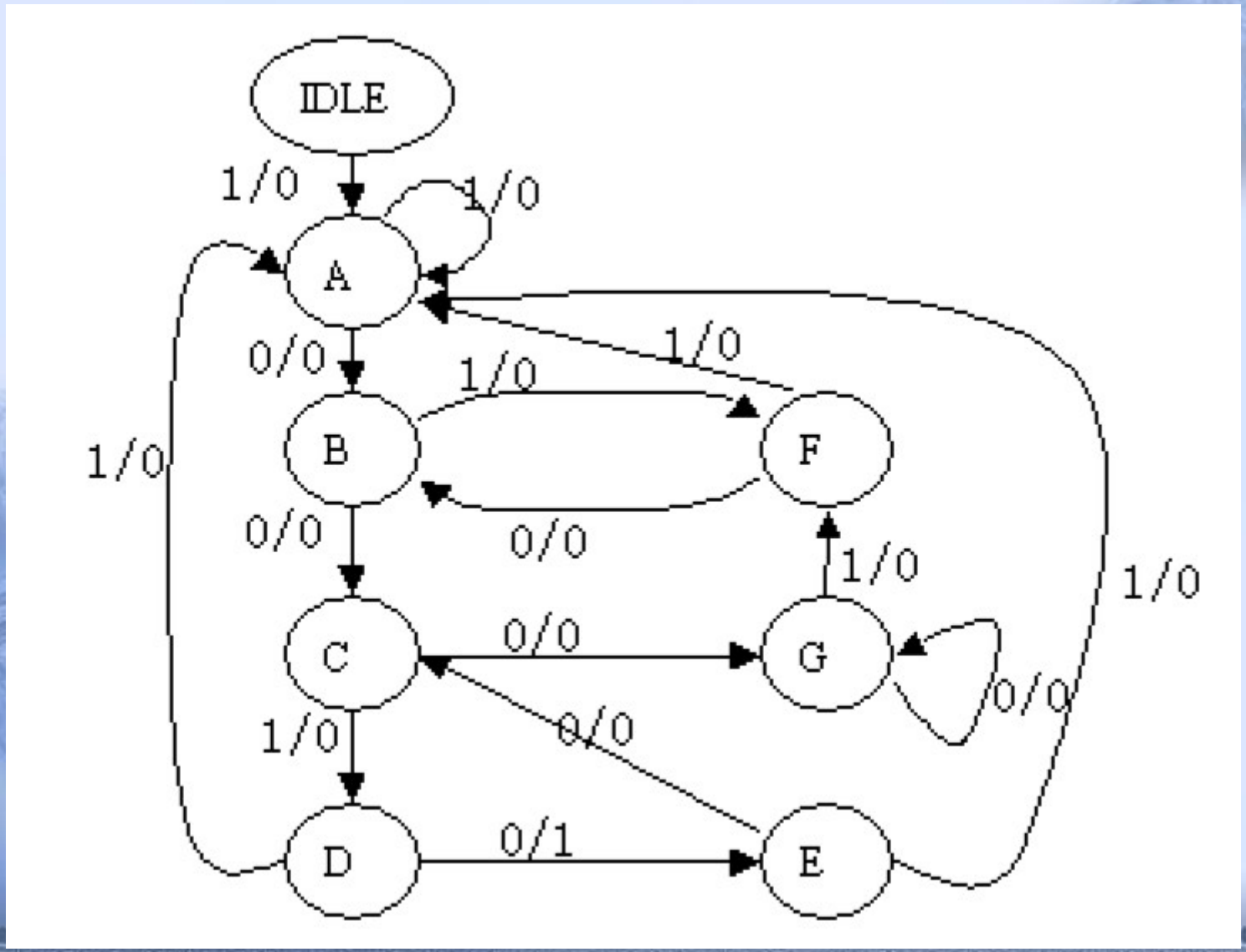
```
module  fsm (Clock, Reset, A,  K2, K1, state);
input Clock, Reset, A;
output K2, K1;
output [1:0] state;
reg K2, K1;
reg [1:0] state ;

parameter   Idle  = 2'b00,
            Start = 2'b01,
             Stop = 2'b10,
            Clear = 2'b11;
```

```
always @(posedge Clock)
  if (!Reset)
  begin
    state <= Idle;
    K2 <=0;
    K1 <=0;
  end
  else
    case (state)
      Idle:
        if (A)
        begin
          state <= Start;
          K1<=0;
        end
        else
        begin
          state <= Idle;
          K2<=0;
          K1<=0;
        end
```

```
Start:
  if (!A)  state <= Stop;
  else     state <= Start;
Stop:
  if (A)
  begin
    state <= Clear;
    K2<= 1;
  end
  else
  begin
    state <= Stop;
    K2<=0;
    K1<=0;
  end
```

```
        Clear:
          if (!A)
          begin
            state <=Idle;
            K2<=0;
            K1<=1;
          end
          else
          begin
            state <= Clear;
            K2<=0;
            K1<=0;
          end
        default: state<=Idle;
    endcase
endmodule
```

对串行输入的数据流进行检测，只要发现10010码型会立即输出一个高位的电平



```
`timescale 1ns/1ns
module seqdet2( x, z, clk, rst);
input x,clk, rst;
output z;
reg [2:0] state;//状态寄存器
wire z;

parameter IDLE = 3 'd0,
          A = 3 'd1,
          B = 3 'd2,
          C = 3 'd3,
          D = 3 'd4,
          E = 3 'd5,
          F = 3 'd6,
          G = 3 'd7;

assign  z =(state==D && x==0) ? 1 :0;
```

```
always @(posedge clk or
negedge rst)
  if(!rst)
  begin
    state<=IDLE;
  end
  else
    casex( state)
      IDLE:
        if(x==1)
        state<=A;
      else  state<= IDLE;
      A:
        if (x==0)
        state<=B;
        else  state<= A;
      B:
        if (x==0)
        state<=C;
        else        state<=F;
C:
  if(x==1)
    state<=D;
  else  state<=G;
D:
  if(x==0)
    state<=E;
  else   state<=A;
E:
  if(x==0)
    state<=C;
  else      state<=A;
```

```
        F:
          if(x==1)
            state<=A;
          else
            state<=B;
        G:
          if(x==1)
          state<=F;
          else state <=G;
        default:
        state<=IDLE;
      endcase
endmodule
```

```
`timescale 1ns/1ns
`define halfperiod 20

module t_seqdet2;
reg clk, rst;
reg [23:0] data;
wire z,x;
assign x=data[23];

always #(`halfperiod) clk=~clk;
always @ (posedge clk)
            #2 data={data[22:0],data[23]};

seqdet2 m ( .x(x), .z(z), .clk(clk), .rst(rst));
```

```
initial
  begin
    clk =0;
    rst =1;
    #2 rst =0;
    #30 rst =1;
    data= 20 'b1100_1001_0000_1001_0100;
    //data= 20 'b1100_0010_0000_1001_0100;
    //data= 20 'b1100_1001_0000_1001_0100; //码流数据
    #(`halfperiod*1000)  $stop;
  end

endmodule
```

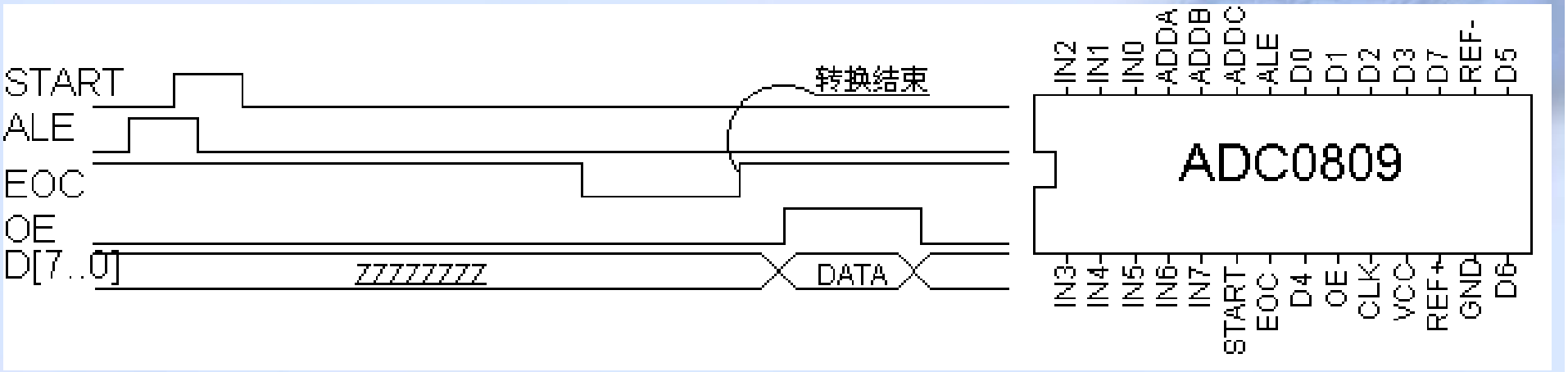
```
`timescale 1ns/1ns
module seqdet3( x, z, clk, rst);
input x,clk, rst;
output z;

reg [2:0] state;//状态寄存器
wire z;

parameter IDLE = 3 'd0,
          A = 3 'd1,
          B = 3 'd2,
        C = 3 'd3,
          D = 3 'd4,
          E = 3 'd5;

assign  z =(state==D && x==0) ? 1 :0;
```

```
always @(posedge clk or
negedge rst)
  if(!rst)
  begin
    state<=IDLE;
  end
  else
    casex( state)
      IDLE:
        if(x==1)
        state<=A;
      else  state<= IDLE;
      A:
        if (x==0)
        state<=B;
        else  state<= A;
      B:
        if (x==0)
          state<=C;
        else state<=A;
      C:
        if(x==1)
        state<=D;
        else  state<=IDLE;
      D:
        if(x==0)
          state<=E;
        else state<=A;
      E:
        if(x==0)
          state<=C;
        else state<=A;
      default:
      state<=IDLE;
    endcase
endmodule
```

控制A/D变换器的几种方法

- 使用单片机
- 使用高性能嵌入式CPU（如ARM）
- 使用DSP
- 使用FPGA
- 使用ASIC

ADC0809时序



START是转换启动信号，高电平有效；ALE是3位通道选择地址（ADDC、ADDB、ADDA）信号的锁存信号。当模拟量送至某一输入端（如IN1或IN2等），由3位地址信号选择，而地址信号由ALE锁存；EOC是转换情况状态信号（类似于AD574的STATUS），当启动转换约100us后，EOC产生一个负脉冲，以示转换结束；在EOC的上升沿后，若使输出使能信号OE为高电平，则控制打开三态缓冲器，把转换好的8位数据结果输至数据总线。至此ADC0809的一次转换结束。

```
module adcint(reset,d,clk,eoc,lock1,ale,start,oe,adda,q);
input[7:0] d;
input reset,clk,eoc;
output lock1,ale,start,oe,adda;
output[7:0] q;
reg ale,start,oe;

parameter st0=0,st1=1,st2=2,st3=3,st4=4,st5=5,st6=6;
reg[2:0] current_state=st0;
reg[2:0] next_state=st0;
reg[7:0] reg1;
reg lock;

assign adda=1'b1;
assign lock1=lock;
```

```
always@(current_state or eoc or reset)
begin
if (reset)
begin
ale<=1'b0;start<=1'b0;oe<=1'b0;lock<=1'b0;next_state<=st0;
end
else
begin
  case (current_state)
    st0:
    begin
      ale<=1'b0;
      start<=1'b0;
      oe<=1'b0;
      lock<=1'b0;
      next_state<=st1;
    end
```

```
st1:
begin
  ale<=1'b1;start<=1'b0;oe<=1'b0;
  lock<=1'b0;next_state<=st2;
end
st2:
begin
  ale<=1'b0;start<=1'b1;oe<=1'b0;
  lock<=1'b0;next_state<=st3;
end
st3:
begin
  ale<=1'b0;start<=1'b0;oe<=1'b0;lock<=1'b0;
  if(eoc==1'b1)
    next_state<=st3;
  else
    next_state<=st4;
end
```

```
st4:
begin
  ale<=1'b0;start<=1'b0;oe<=1'b0;lock<=1'b0;
  if(eoc==1'b0)
    next_state<=st4;
  else
    next_state<=st5;
end
st5:
begin
  ale<=1'b0;start<=1'b0;oe<=1'b1;
  lock<=1'b0;next_state<=st6;
end
st6:
begin
  ale<=1'b0;start<=1'b0;oe<=1'b1;
  lock<=1'b1;next_state<=st0;
end
```

```
      default:
      begin
        ale<=1'b0;start<=1'b0;oe<=1'b0;
        lock<=1'b0;next_state<=st0;
      end
    endcase
end
end

always @(posedge clk or posedge reset)
begin
    if (reset)
      current_state<=st0;
    else
      current_state<=next_state;
end
```

```
always @(posedge lock)
begin
  regl<=d;
end

assign q=regl;

endmodule
```