EDA Technology

Chapter 8

Design Technology of State Machine

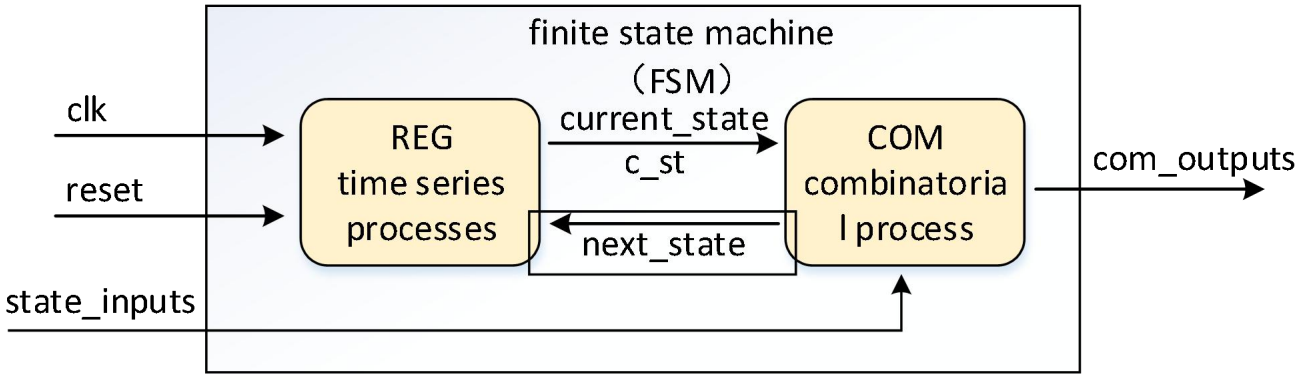
General Form of Verilog State Machine

- General Structure of State Machine
  - From the angle of signal output mode of state machine, there are two types of state machine, namely Mealy-type and Moore-type state machines.
  - From the description structure of state machine, there are single process state machine and multi process state machine.
  - In terms of state expression, there are symbolic state machines and state machines that determine state encoding.
  - From the state machine coding method, there are sequential encoding, one hot encoding or other coding state machine.

```
parameter[2:0]  s0=0, s1=1, s2=2,
s3=3, s4=4 ;
reg[2:0] current_state, next_state;
```

Moore-type State Machine

- State Machine with Multiprocess Structure

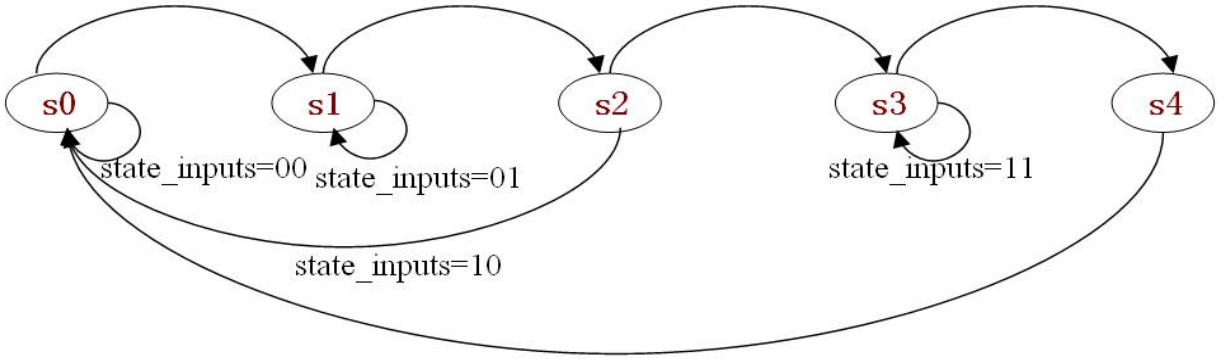


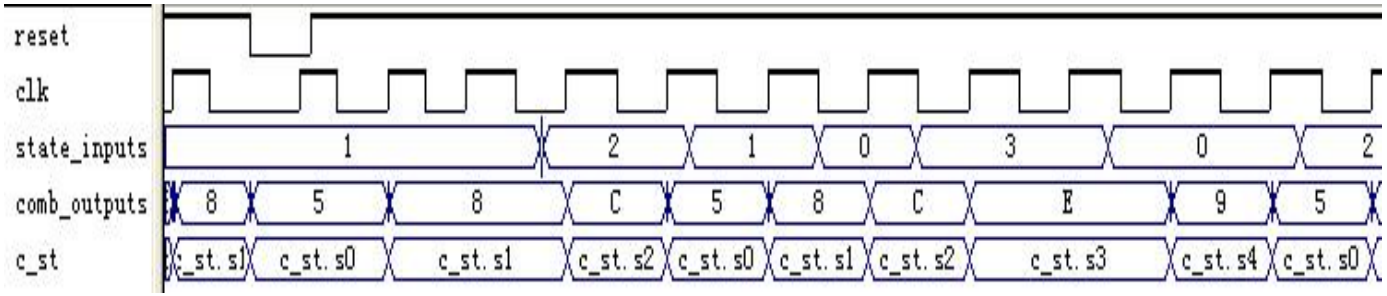
Moore-type State Machine

```
module FSM_EXP (clk, reset, state_inputs, comb_outputs);
input clk;                          //The working clock of state
input reset;                        //Reset control of state
input [0:1] state_inputs;                  // State machine co
output [3:0] comb_outputs;              //State machine output
reg [3:0] comb_outputs;
parameter  s0=0,s1=1,s2=2,s3=3,s4=4 ;      //Defining state
reg [4:0]  c_st, next_state;               //Defining the stat
always @(posedge clk or negedge reset) begin //Sequential
 if (!reset)  c_st<=s0;                    // When the reset i
  else      c_st<=next_state ;   end
always @(c_st or state_inputs) begin    //Combinational pro
  case (c_st)                  // In order to see clearly in the
   s0 : begin  comb_outputs<=5 ;     // When the state s0 i
     if (state_inputs==2'b00)  next_state<=s0;    // The co
      else  next_state<=s1;   end       // The condition
   s1 : begin  comb_outputs<=8 ;              // When it en
     if (state_inputs==2'b01)  next_state<=s1;
      else  next_state<=s2 ;  end
   s2 :  begin  comb_outputs<=12 ;
     if (state_inputs==2'b10)  next_state<=s0;
      else  next_state<=s3 ;   end
   s3 :  begin  comb_outputs<=14 ;
     if (state_inputs==2'b11)  next_state<=s3;
      else   next_state<=s4 ;  end
   s4 :   begin  comb_outputs<=9 ; next_state<=s0 ;   end
       default : next_state<=s0 ;       // If the current s
    endcase    end
endmodule
```

Moore-type State Machine

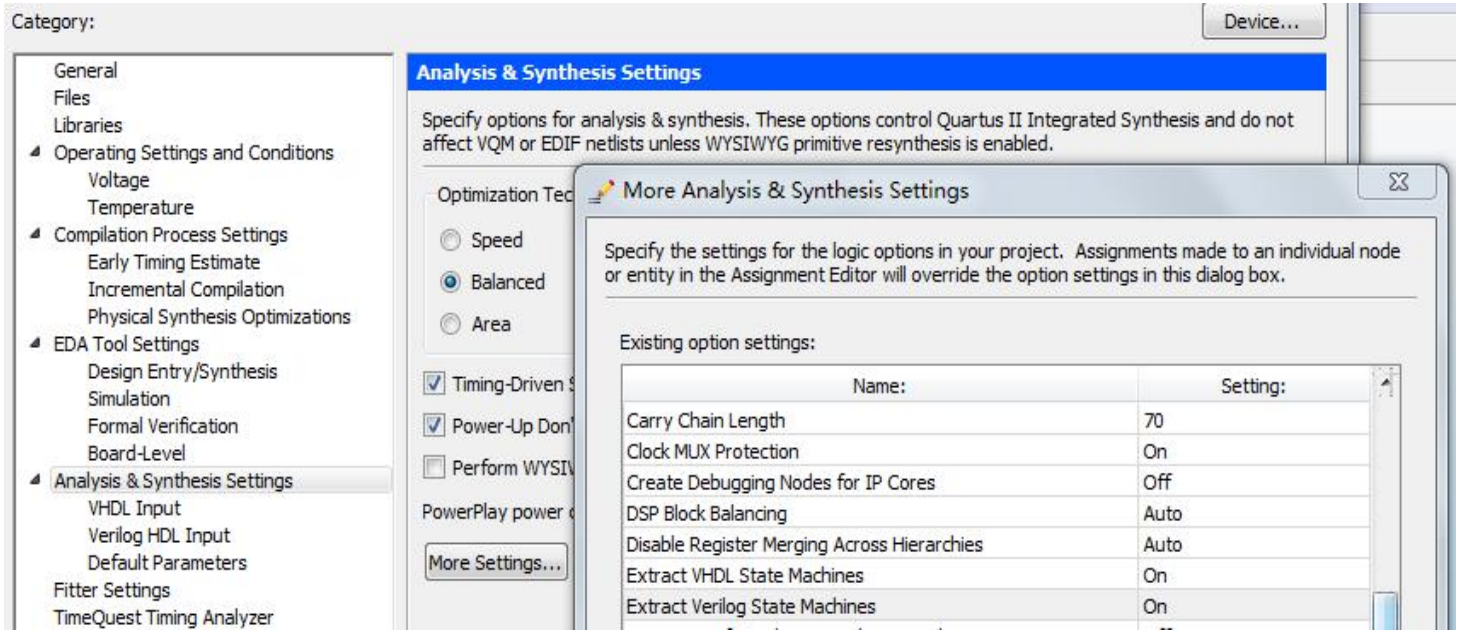
- Sequence Detector and Its State Machine Design





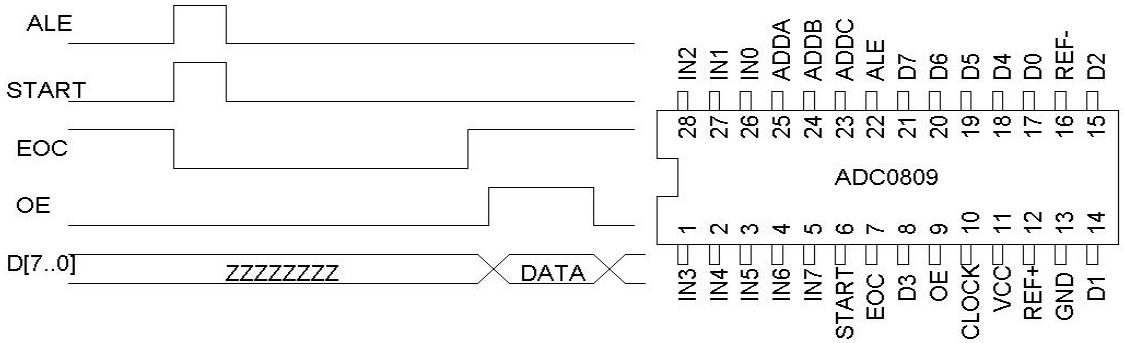
Moore-type State Machine

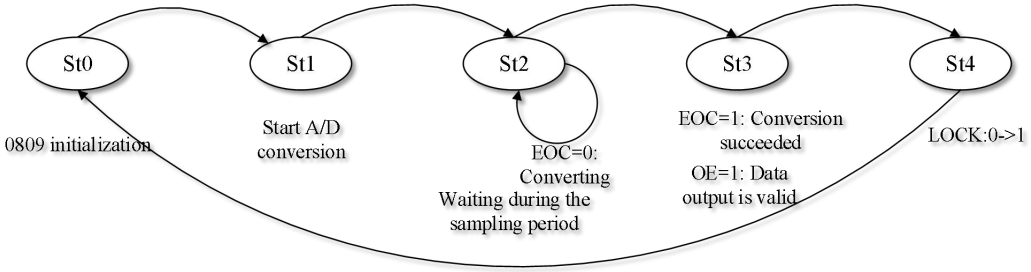
- Initial Control and Expression



Moore-type State Machine

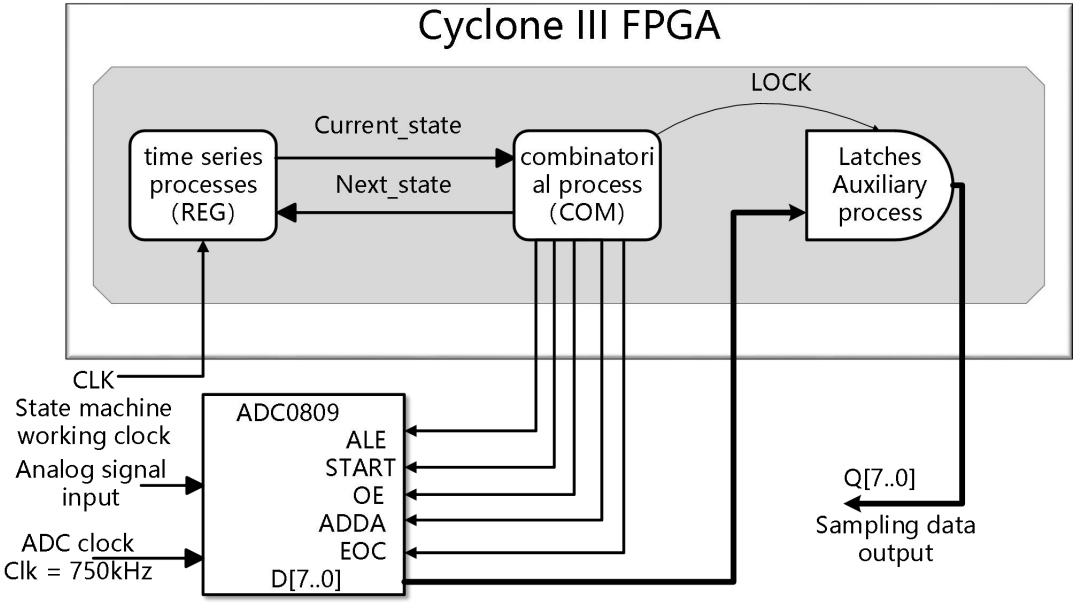
- State Machine with Multiprocess Structure





Moore-type State Machine

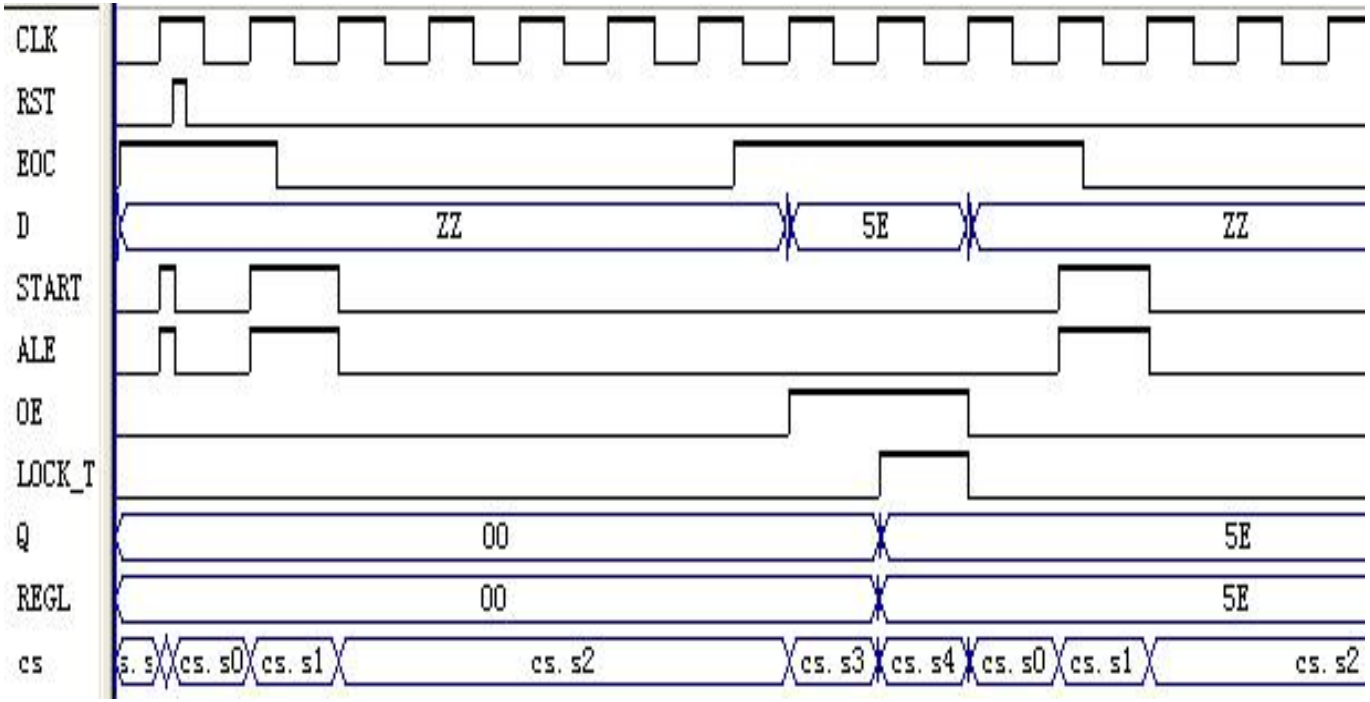
- State Machine with Multiprocess Structure



```
module ADC0809 (D, CLK, EOC, RST, ALE, START, OE, ADDA, Q,LOCK_T);
   input[7:0] D;                    //8-bit data converted from 0809
    input CLK,RST;                  // Working clock and system reset control of state 、、
                                    //machine
    input EOC;                      //The indication for state conversion and the low
                                    //level represents that it is being converted.
   output ALE;                      // Channel address latch signals of 8 analog signal
    output START,OE ;               // The start signal of conversion and three-state
                                    //control signal of data output
    output ADDA,LOCK_T ;            // Control signal of signal channel and latch test
                                    //signal
   output[7:0] Q;     reg ALE, START, OE;
   parameter s0=0,s1=1,s2=2,s3=3,s4=4;     // Defining the subtypes of each state
    reg[4:0] cs , next_state ;             // For the convenience of the simulation
                                           //display, the current state name is
                                           //simplified to be cs
    reg[7:0] REGL;  reg LOCK;              // Data output latch clock signal after
                                           //conversion
```

```
always @(cs or EOC)  begin                    // A combinational process that
                                               //specifies the conversion mode of each
                                               state
   case (cs)
    s0 : begin  ALE=0 ; START=0 ; OE=0 ; LOCK=0 ;
         next_state <= s1 ;   end                          //0809 initialization
    s1 : begin  ALE=1 ; START=1 ; OE=0 ; LOCK=0 ;
          next_state <= s2 ;    end                        // Starting sampling signal
                                                           //START
    s2 : begin  ALE=0 ; START=0 ; OE=0 ; LOCK=0 ;
         if (EOC==1'b1)  next_state = s3 ;                 //EOC=0 indicating the end of
                                                           //conversion
          else  next_state = s2 ;  end                     // The conversion does not
                                                           //end, and continues to wait
     s3 : begin ALE=0 ; START=0 ; OE=1;  LOCK=0;           // Starting OE and opening
                                                           //the AD data port
          next_state = s4 ;     end                        //The next state
                                                           //unconditionally turn to s4
     s4 : begin ALE=0 ;  START=0 ; OE=1;  LOCK=1;          //Turning on  the data latch
                                                           //signal
         next_state <= s0 ;     end
    default : begin  ALE=0 ; START=0 ; OE=0 ; LOCK=0 ;
        next_state = s0 ;     end
   endcase    end
```

```
always @(posedge CLK or  posedge RST)  begin          //Sequential process
   if (RST) cs <= s0 ;
     else  cs <= next_state ; end          // The current state value is taken out
                                           //of the process by the current state
                                           //variable cs
always @(posedge LOCK)                     //Register process
    if (LOCK)   REGL <= D ;                 //In this process, the converted data
                                            //is locked on the rising edge of LOCK.
     assign ADDA =0 ; assign Q = REGL ;    // Selecting the analog signal to enter
                                           //the channel IN0
    assign LOCK_T = LOCK ;                 //Outputting the testing signal
endmodule
```


CLK
RST
EOC
D
START
ALE
OE
LOCK_T
Q
REGL
cs
ZZ
5E
ZZ
00
5E
00
5E
s. s
cs. s0
cs. s1
cs. s2
cs. s3
cs. s4
cs. s0
cs. s1
cs. s2

```
always @(cs or EOC)  begin
case (cs)
   s0 : next_state <= s1 ;
   s1 : next_state <= s2 ;
   s2 : if (EOC==1'b1)  next_state=s3 ; else  next_state=s2 ;
   s3 : next_state = s4 ;
  s4 : next_state <= s0 ;
    default : next_state = s0 ;
 endcase   end
always @(cs)  begin
 case (cs)
   s0 : begin  ALE=0 ; START=0 ; OE=0 ; LOCK=0 ;  end
   s1 : begin  ALE=1 ; START=1 ; OE=0 ; LOCK=0 ;  end
   s2 : begin  ALE=0 ; START=0 ; OE=0 ; LOCK=0 ;  end
   s3 : begin  ALE=0 ; START=0 ; OE=1 ; LOCK=0 ;  end
   s4 : begin  ALE=0 ; START=0 ; OE=1 ; LOCK=1 ;  end
default : begin  ALE=0 ; START=0 ; OE=0 ; LOCK=0 ;  end
 endcase    end
```

State Machine with Different Coding Types

⌘ Direct Output Coding

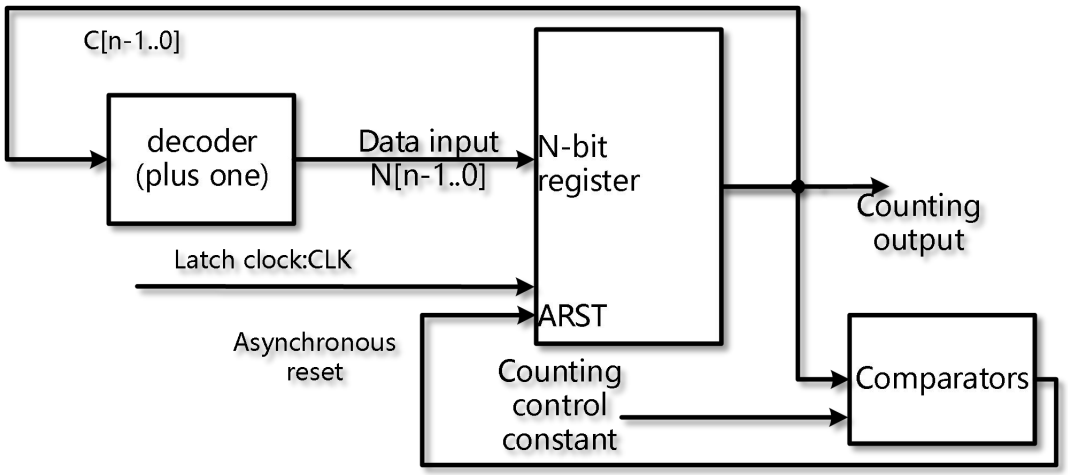


Figure: The general model of addition counter

The output of this counter is the output of the state coding.

output=state

State coding table of controlling signal

state	State coding					
	START	ALE	OE	LOCK	B	Function specification
S0	0	0	0	0	0	Initial state
S1	1	1	0	0	0	Start transformation
S2	0	0	0	0	1	If EOC=1, transfer to next state ST3
S3	0	0	1	0	0	Output the transformed data
S4	0	0	1	1	0	Latch the transformed data using the rising edge of LOCK

S0~S4：00000、11000、00001、00100、00110.

[Example]

```
module ADC0809  (D, CLK, EOC, RST, ALE, START, OE, ADDA, Q,LOCK_T);
  input[7:0] D;  input CLK,RST,EOC;
  output START,OE , ALE, ADDA,LOCK_T ;   output[7:0] Q;
 parameter s0=5'B00000,s1=5'B11000,s2=5'B00001,s3=5'B00100,s4=5'B00110;
  reg[4:0] cs ,SOUT, next_state ;  reg[7:0] REGL;  reg LOCK;
  always @ (cs or EOC)  begin
   case (cs)
     s0 : begin  next_state<=s1 ;  SOUT=s0 ;   end
     s1 : begin  next_state<=s2 ;  SOUT=s1 ;   end
     s2 : begin  SOUT=s2 ;
          if (EOC==1'b1)  next_state=s3 ;  else  next_state=s2;   end
     s3 : begin  SOUT=s3 ;  next_state = s4 ;   end
     s4 : begin  SOUT=s4 ;  next_state = s0 ;   end
     default : begin  next_state=s0 ;  SOUT=s0; end
   endcase
   end
 always @ (posedge CLK or  posedge RST)  begin
    if (RST) cs <= s0 ;  else  cs<=next_state ; end
 always @ (posedge SOUT[1] )
  if (SOUT[1])   REGL <= D ;
 assign ADDA=0;    assign Q=REGL;   assign LOCK_T=SOUT[1];
 assign OE=SOUT[2] ; assign ALE=SOUT[3];  assign  START=SOUT[4];
endmodule
```

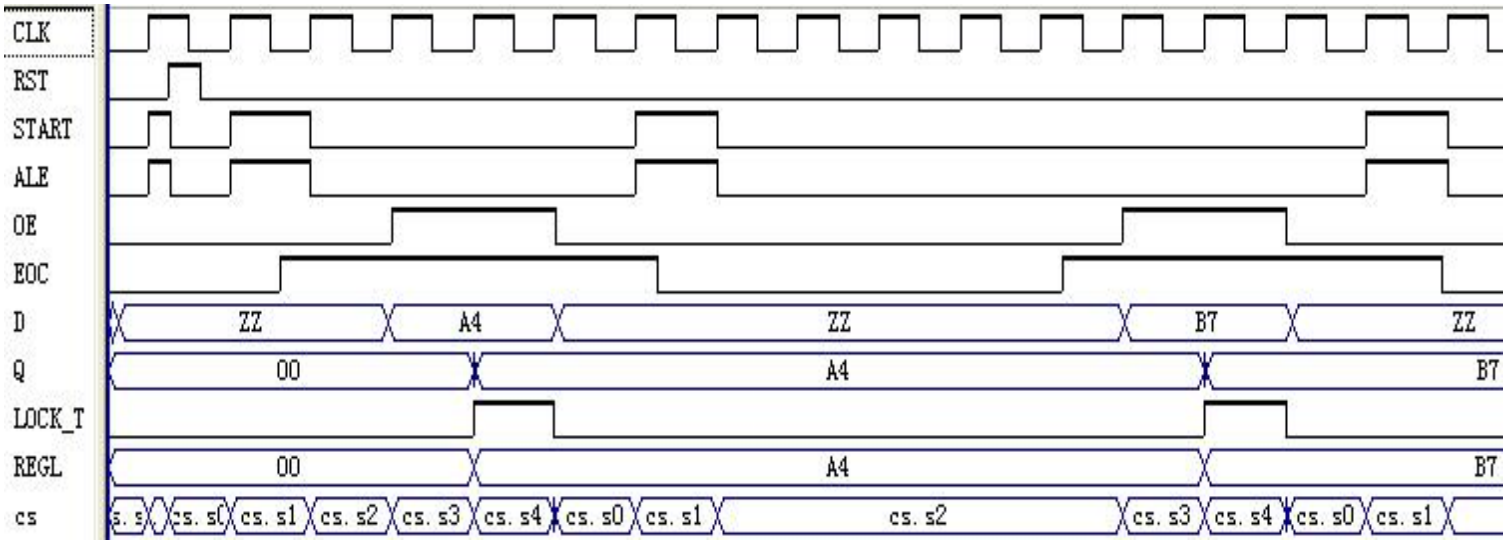


Figure: The working timing diagram of the state machine in Example 8-8

State Machine with Different Coding Types

- Defining the State Coding with the Use of Macro Definition Statement

```
`define s0  5'B00000
`define s1  5'B11000
`define s2  5'B00001
`define s3  5'B00100
`define s4  5'B00110
module ADC0809 (D, CLK, EOC, RST, ALE, START, OE, ADDA, Q,LOCK_T);
   input[7:0] D;    input CLK,RST,EOC;
   output START,OE, ALE, ADDA,LOCK_T ;   output[7:0] Q;
   reg[4:0] cs;   reg[4:0] SOUT, next_state; reg[7:0] REGL; reg LOCK;
always @ (cs or EOC)  begin
    case (cs)
      `s0 : begin  next_state<=`s1 ;  SOUT=`s0 ;   end
      `s1 : begin  next_state<=`s2 ;  SOUT=`s1 ;   end
      `s2 : begin  SOUT=`s2 ;
         if (EOC==1'b1)  next_state=`s3;  else  next_state=`s2;  end
      `s3 : begin  SOUT=`s3 ;  next_state = `s4 ;   end
      `s4 : begin  SOUT=`s4 ;  next_state = `s0 ;   end
     default : begin  next_state=`s0 ;  SOUT=`s0; end
    endcase    end
```

```
  always @ (posedge CLK or  posedge RST)  begin
    if (RST) cs <= `s0 ;  else  cs<=next_state ; end
  always @ (posedge SOUT[1] )
   if (SOUT[1])   REGL <= D ;
  assign ADDA =0 ;   assign Q = REGL ;
  assign LOCK_T = SOUT[1] ;    assign  START = SOUT[4] ;
  assign    ALE = SOUT[3] ;    assign    OE = SOUT[2] ;
endmodule
```

- Difference between `define and parameter:
- `define: it aims at the overall design and can be located outside the module;
- parameter: it has the local feature and is located in the module. (Of course, `define can also be located in the module)

One advantage of macro definition is that the code of each state can be seen in the simulation waveform.

Command statement of macro definition

`define *macro name* *macro content*

```
`define  s   A+B+C+D

assign  DOUT = A+B+C+D+E;
               ---------
                  `s
```

State Machine with Different Coding Types

Sequential Coding

Table: The coding methods of state machine

States	Sequential-Encoded	One-Hot-Encoded	Johnson-Encoded
State0	000	100000	0000
State1	001	010000	1000
State2	010	001000	1100
State3	011	000100	1110
State4	100	000010	1111
State5	101	000001	0111

Sequential coding: it uses the least flip-flops and has the least remaining illegal states. But sometimes, adjacent state or nonadjacent state refers to the simultaneous state transformations of multiple flip-flops, and thus consumes more transformation time. Also, it will cause burr.

State Machine with Different Coding Types

- One-hot Coding

One-hot coding: use *n* flip-flops to realize the state machine with *n* state. When the state machine is at one state, the corresponding flip-flops is 1 and others are 0.

Simple decoding, high transformation speed and good operation stability are the advantages of the one-hot coding.

State Machine with Different Coding Types

There are several ways to set up the coding of states.

1. User-defined way

Just write the code of state in the program and does not need the use of EDA software tool. For example, 8-10.

2. Set up using property definition statement

[Example 8-10]

```
module SCHK  (input  CLK, DIN, RST, output reg SOUT);
parameter s0=0, s1=1, s2=2, s3=3, s4=4, s5=5, s6=6, s7=7,s8=8 ;
 (*syn_encoding="one-hot"*) reg[8:0] ST ;
always @(posedge CLK)   begin

    (* syn_encoding = "one-hot" *)
```

State Machine with Different Coding Types

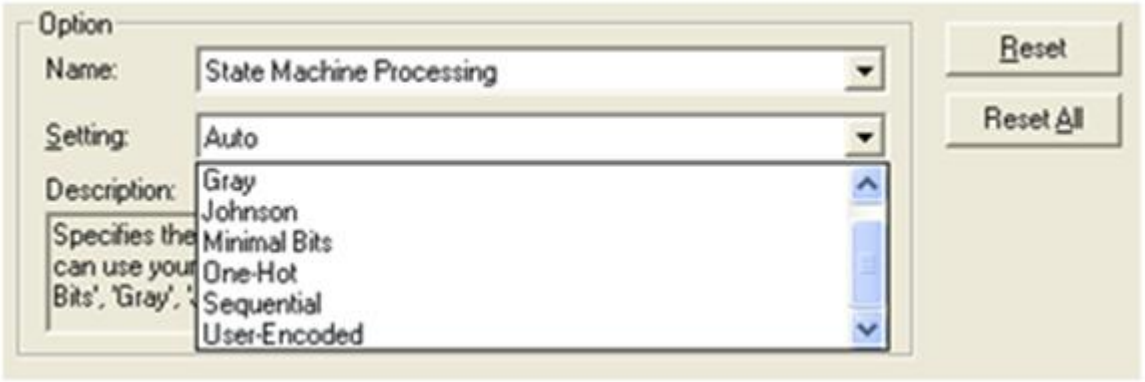
Set up using property definition statement

Table: Attribute definitions of coding ways and references of resource consumption

Coding way	Property definition of coding ways	The number of LCs	The number of REGs
One-hot coding	(*syn_encoding="one-hot"*)	13	10
User-defined coding	(*syn_encoding="user"*)	12	5
Gray coding	(*syn_encoding="gray"*)	8	5
Sequential coding	(*syn_encoding="sequential"*)	10	5
Johnson coding	(*syn_encoding="johnson"*)	23	6
Default coding	(*syn_encoding="default"*)	13	10
Compact coding	(*syn_encoding="compact"*)	9	5
Safe one-hot coding	(*syn_encoding="safe, one-hot"*)	21	10

Direct setting up

Assignments-> Settings- >Category-> Analysis & Synthesis Setting ->More Settings- >Option->Name- >State Machine Processing



Design of Safe State Machine

Table: The remaining states

State	Sequential code
s0	000
s1	001
s2	010
s3	011
s4	100
s5	101
s6	110
s7	111

There is no remaining state in Table 8-4. But in most of time, there exist remaining states, such as example 8-1. These states do not be needed in the normal operation, and are commonly called illegal states.

If the system enters into these illegal states, it is required to go back into the normal states.

In example of 8-1, the program defines 5 legal states (effective states), s0,s1,s2,s3,s4. If the sequential code is used to denote the state, at least 3 flip-flops are needed. Then it generates at most 8 possible states. The coding way is shown in Table 8-4. Therefore, the last 3 states "s5,s6,s7" are illegal states.

State guiding method

An example :

```
parameter s0=0,s1=1,s2=2,s3=3,s4=4, s5=5, s6=6,s7=7;
   ...
S5 :   next_state = s0 ;
S6 :   next_state = s0 ;
S7 :   next_state = s0 ;
default : begin  next_state=s0 ;
```

The advantage of this method is direct and reliable. The disadvantage is that if the number of illegal states are large, it consumes a lot of logic resources. Therefore, it is only suitable to the state machine with sequential coding.

Monitoring Method of State Coding

For one-hot code, the remaining states are so many that the state guiding method becomes complicated.

Therefore, logic detection module can be designed: check whether the number of "1" in the state code is bigger than 1. If sum of bits of the state code is bigger than 1, it must be the illegal state.

Auto-generation of Safe State Machine with the Use of EDA Tool

In addition, the property statements can also be used, like （ *syn_encoding="safe, one-hot"*）.