基于FPGA的
时序逻辑电路设计和实现
EDA实验二

赵晓燕
电工电子实验中心

主要内容

一. 时序电路的HDL描述

二. 有限状态机的HDL描述

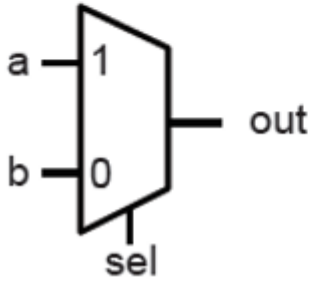
三. EDA实验二内容

组合与时序always 模块

Combinational

```
module combinational(a, b, sel,
                                out);
  input a, b;
  input sel;
  output out;
  reg out;

  always @ (a or b or sel)
  begin
    if (sel) out = a;
    else out = b;
  end

endmodule
```

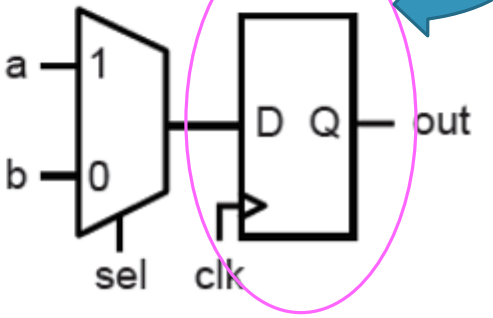
= 阻塞赋值



Sequential

```
module sequential(a, b, sel,
                        clk, out);
  input a, b;
  input sel, clk;
  output out;
  reg out;

  always @ (posedge clk)
  begin
    if (sel) out <= a;
    else out <= b;
  end

endmodule
```

<=非阻塞赋值



always 模块使用TIPS

- ✓ always模块中的变量必须声明为reg型；
- ✓ 设计时序逻电路时，要用**非阻塞赋值“<=”**；
- ✓ 模拟寄存器时，要用**非阻塞赋值“<=”**；
- ✓ 用always块设计组合电路时，要用**阻塞赋值“=”**；
- ✓ 在同一个always块中，既有组合逻辑又有时序逻辑时，要用**非阻塞赋值“<=”**；
- ✓ 在同一个always块中，不要混用阻塞和非阻塞赋值。
- ✓ **不要在多个always块中对同一个变量赋值**

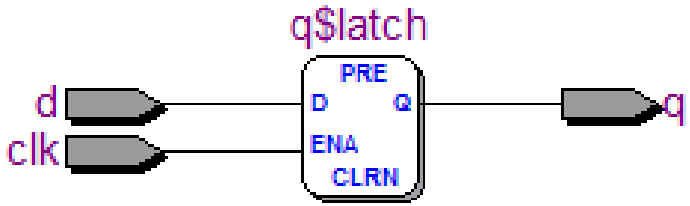
时序电路----D触发器(寄存器flipflop)

```
module latc1
 (input clk,
  input d,
  output reg q);
always @(clk or d )
begin
    if(clk)
     q<=d;
    end
endmodule
```

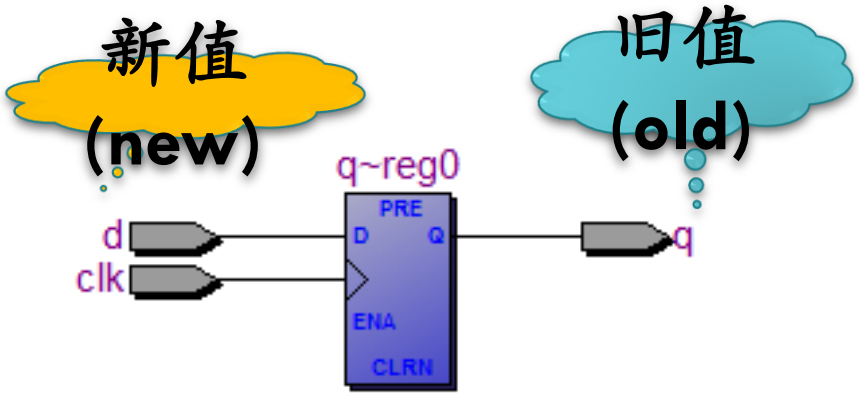
电平触发

```
module FF0
 (input clk,
  input d,
  output reg q);
always @(posedge clk )
begin
    q<=d;
    end
endmodule
```

边沿触发

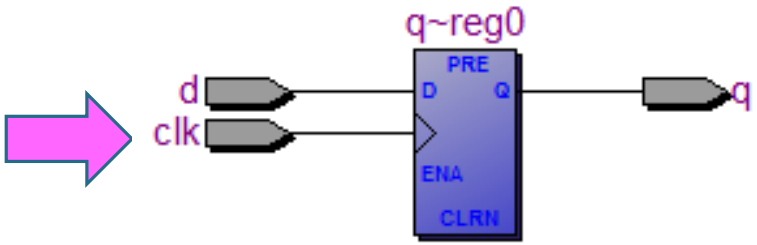


D型锁存器

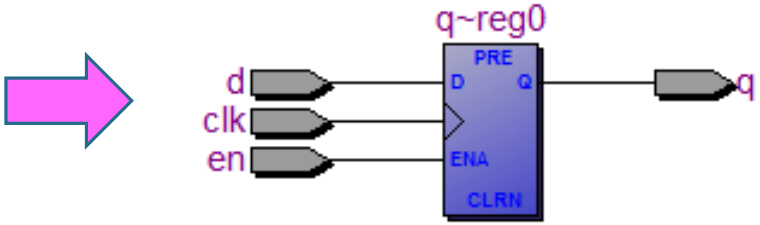


时序电路------D触发器

```
module FF0(input clk,input
d,output reg q);
always @( posedge clk )
begin
    q<=d;
end
endmodule
```




```
module FF0(input clk,input
d,output reg q);
always @( posedge clk )
begin
    if  (en)//使能端
    q<=d;
end
endmodule
```



时序电路------D触发器

```
always @( posedge clk )
begin
   if (~resetN)
         Q<=0;
   else if (enble)
         Q<=D;
end
```

同步复位

```
always @( posedge clk or negedgde resetN )
begin
   if (~resetN)
         Q<=0;
   else if (enble)
         Q<=D;
end
```

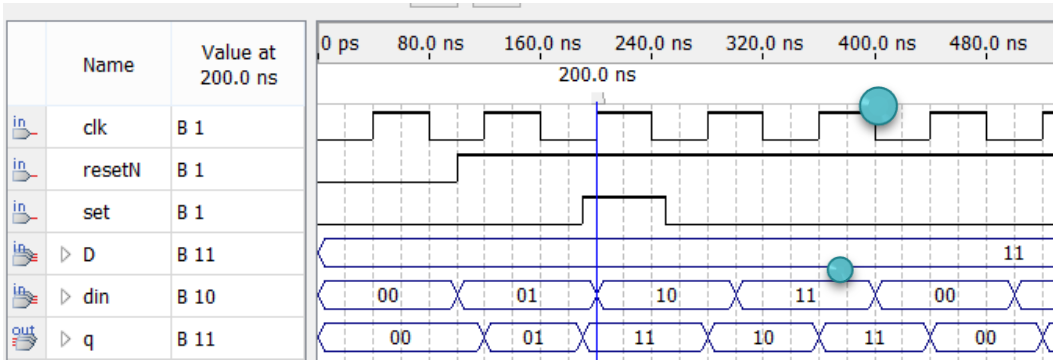
异步复位

敏感列表里包含复位信号

时序电路------ D触发器

敏感列表里包含置数信号

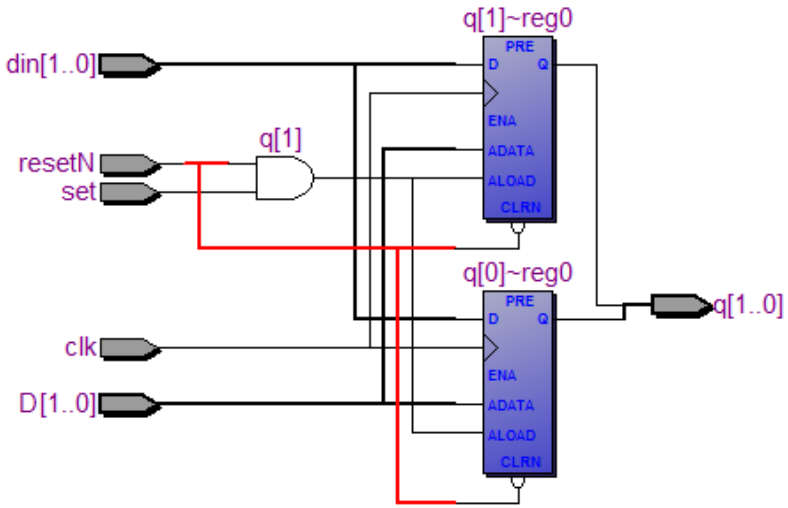
```
module flipflop
 (input clk,resetN,set,
  input [1:0]D,din,
  output reg [1:0]q);
```




```
always @(posedge clk or negedge resetN or posedge set)
begin
  if (~resetN)
          q<=0;

  else if (set)
          q<=D;//
```

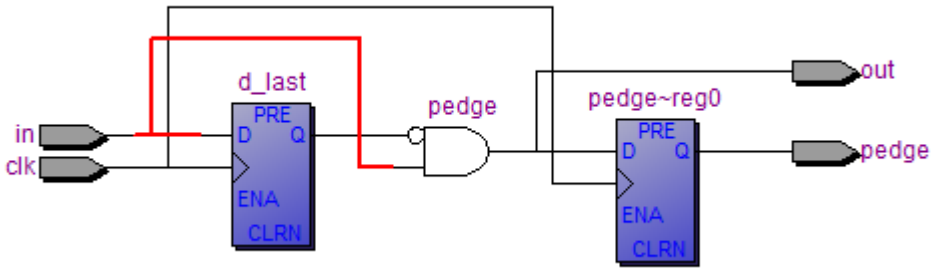
异步置数

```
  else q<=din;
end
endmodule
```

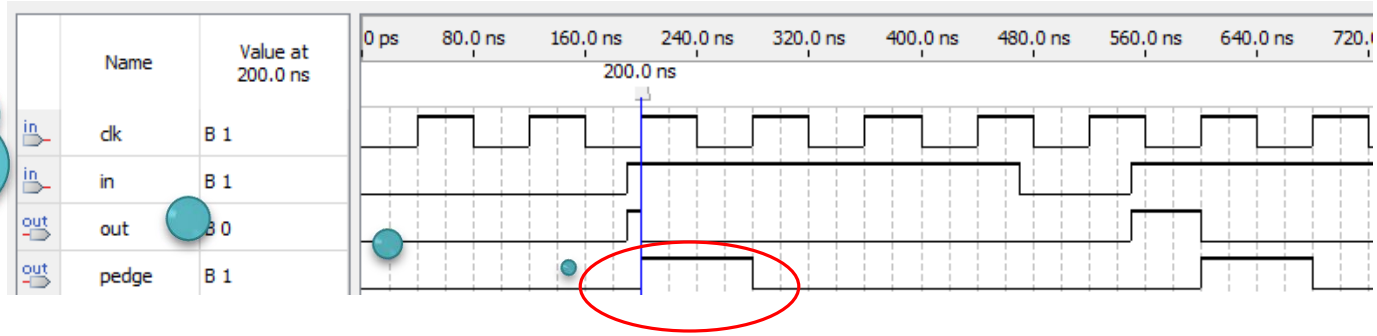


D触发器应用----边沿检测

```
module top_module(
        input clk,
        input in,
        output reg pedge,
        output out);

                reg  d_last
        always @(posedge clk) begin
                d_last <= in;  //存入in当前输入值
                pedge <= in & ~d_last; // in旧值为0，新值为1时
                                          表示检测到上升沿
                end
                assign  out=in & ~d_last;
endmodule
```



时序电路可输出稳定的电平信号



D触发器应用----计数器

```
module dff_asyn3(clk,d,rst,en,q);
input clk,rst,en;
input [1:0]d;
output [1:0]q;
reg [1:0]q;

always@(posedge clk or negedge
rst)
 begin
  if(!rst)
      q<=0;//  异步清零
  else  if(!en)

      q<=d;  // 同步置数
  else
       q<=q+1; //  计数

  end
endmodule
```

灵活使用if else语句描述电路的行为

- if else三种表达形式

1） if（表达式）
　　语句1;

2）if（表达式）
　　语句1;
　else
　　语句2;

3）if（表达式1）　语句1;
　else　if（表达式2）　语句2;
　else　if（表达式3）　语句3;
　　.....
　else　　语句n;

- if语句可以嵌套，注意else总是与它上面的最近的 if 进行配对。如果不希望else与最近的 if 配对，可以采用begin_end进行分割，如：

```
if ()
  begin
    if ()  语句1;
  end
else
  语句2;
```

计数器

利用使能端、进位、清零端构成任意进制计数器

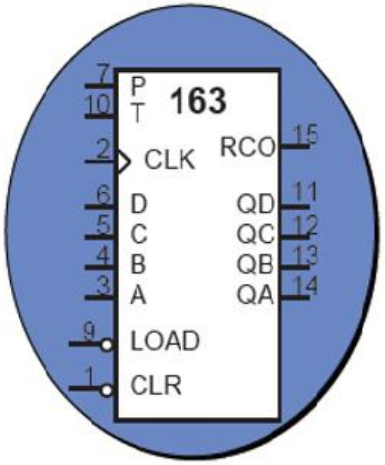
复位

置数

使能、累加

进位

```
module counter(LDbar, CLRbar, P, T, CLK, D,
                count, RCO);
  input LDbar, CLRbar, P, T, CLK;
  input [3:0] D;
  output [3:0] count;
  output RCO;
  reg [3:0] Q;

 always @ (posedge CLK) begin
   if (!CLRbar) Q <= 4'b0000;
   else if (!LDbar) Q <= D;
   else if (P && T) Q <= Q + 1;
 end

 assign count = Q;
 assign RCO = Q[3] & Q[2] & Q[1] & Q[0] & T;
endmodule
```



时序电路------分频器

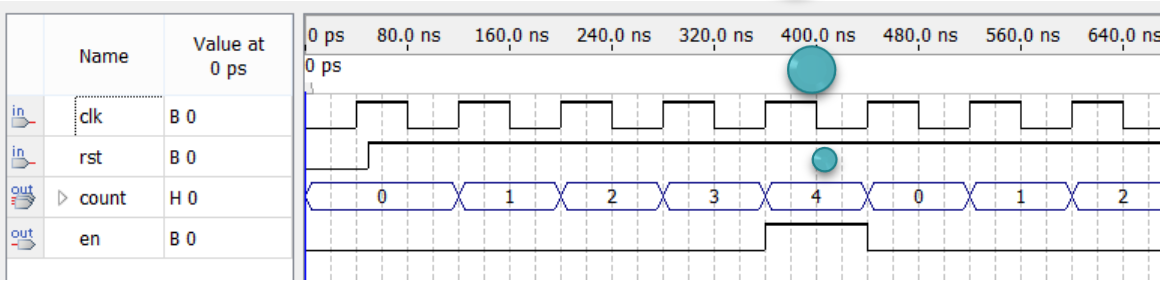
```
module fenp(clki,rst,clk1,clk2);
input clki,rst;
output clk1,clk2;

reg [30:0]q;

always@(posedge clki, negedge rst)

   if(!rst)  q<=0;
   else  q=q+1;

assign clk1=q[16];//输出250hz
assign clk2=q[24];//输出1HZ

endmodule
```

计数器输出
相邻位满足
2倍频关系

时序电路------分频器

```
module fenp(clk,rst,en,count);
input clk,rst;
output en;
output [3:0] count;
reg [3:0] count;

always@(posedge clk,negedge rst)
begin
   if (!rst) count<=0;
       else count<=(count==4)?0:count+1;
end

assign en=(count==4);

endmodule
```

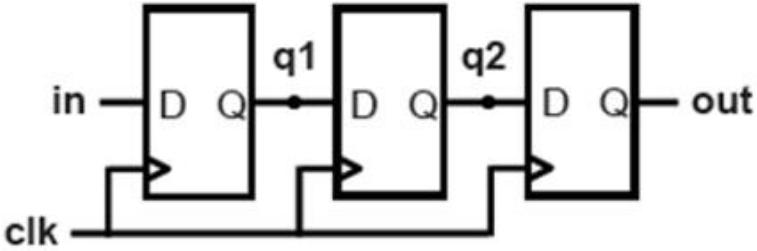
产生使能
控制信号



移位寄存器-----普通移位

```
always @ (posedge clk)
begin
   q1 <= in;
   q2 <= q1;
   out <= q2;
end
```

q1、q2的旧值

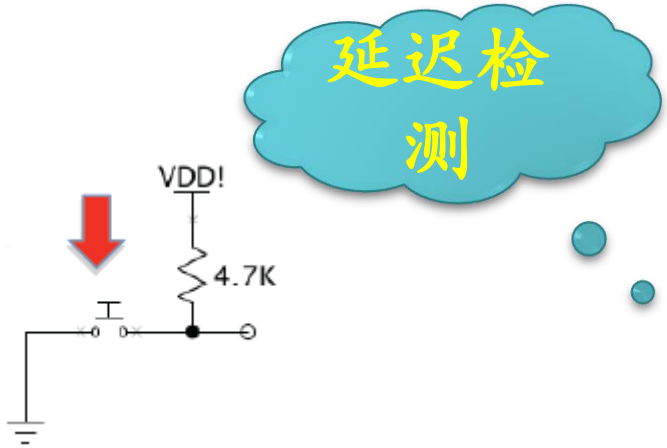


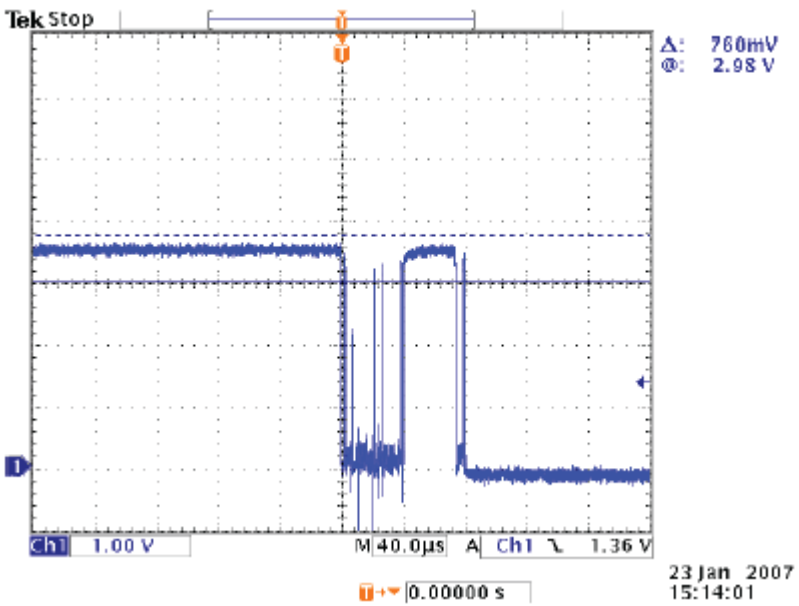

```
module top_module(
    input clk,
    input areset
    input load,
    input ena,
    input [3:0] data,
    output reg [3:0] q);

always @(posedge clk or posedge areset)
begin
      if(areset)
         q <= 4‘b0;//复位
      else if(load)
         q <= data;//置数
      else if(ena)
         q <= q >> 1; //右移，最低位移出
                        // q <= {1'b0,q[3:1]};
      else
         q <= q; //不移位
   end
endmodule
```

移位寄存器-----循环移位

```
module top_module(
    input clk,
    input load,
    input [1:0] ena,
    input [99:0] data,
    output reg [99:0] q);

  always @(posedge clk) begin
      if(load)
          q <= data;
      else begin
          case(ena)
              2'b01: q <= {q[0],q[99:1]};//循环右移
               2'b10: q <= {q[98:0],q[99]};// 循环左移
              default: q <= q;
          endcase
      end
  end

endmodule
```

消抖电路(D触发器、计数器)





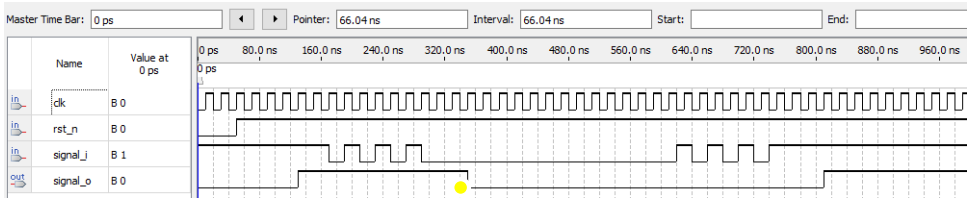

```
// Switch Debounce Module
// use your system clock for the clock input
// to produce a synchronous, debounced output
// DELAY = .01 sec with a 27Mhz clock
module debounce #(parameter DELAY=270000-1)
                 (input reset, clock, bouncey,
                  output reg steady);

   reg [18:0] count;
   reg old;

   always @(posedge clock)
     if (reset)     // return to known state
       begin
         count <= 0;
         old <= bouncey;
         steady <= bouncey;
       end
     else if (bouncey != old)  // input changed
       begin
         old <= bouncey;
         count <= 0;
       end
     else if (count == DELAY) // stable!
         steady <= old;
     else                     // waiting…
       count <= count+1;

endmodule
```

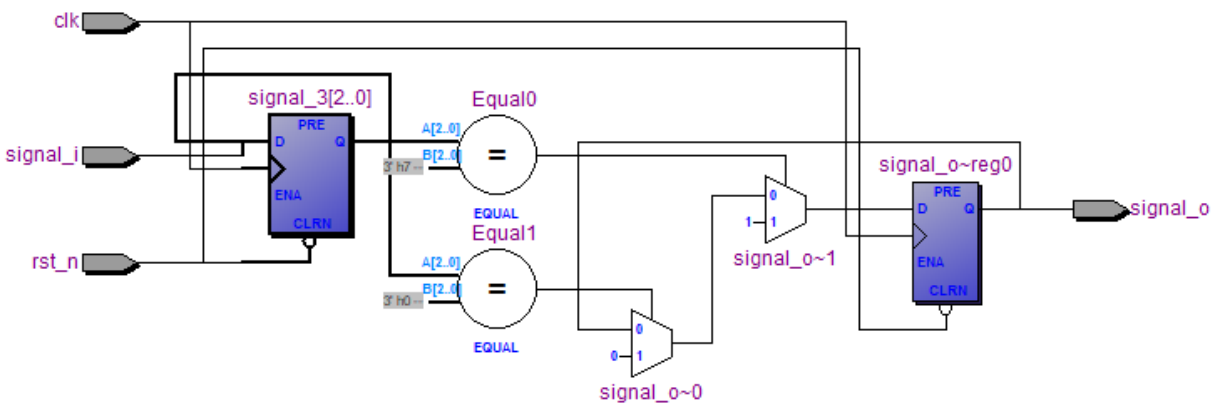
经验值，抖动时间一般不会大于20ms.

若未给出抖动时间，如果数据稳定20ms，认为可以采集该数据。

消抖电路（D触发器、移位寄存器）

```
always@(posedge clk or negedge rst_n)
    begin
            if(!rst_n)   signal_3 <= 0;
            else         signal_3 <= { signal_3[1:0],signal_i };
    end

always@(posedge clk or negedge rst_n)
    begin
            if(!rst_n)   signal_o <= 0;

            else if( signal_3 == 3‘b111 )      signal_o <= 1'b1;

            else if( signal_3 == 3‘b000 )      signal_o <= 1'b0;
    end
```

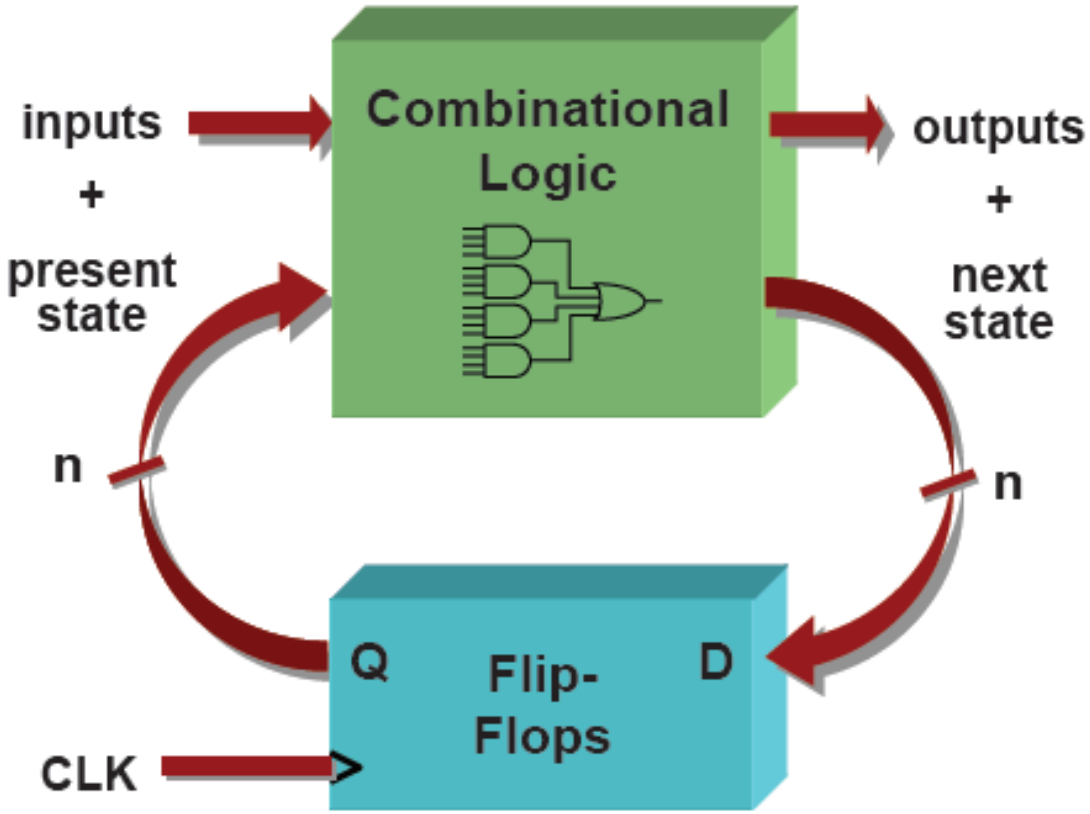




过滤2个时钟周期内的抖动。

时序电路------有限状态机FSM

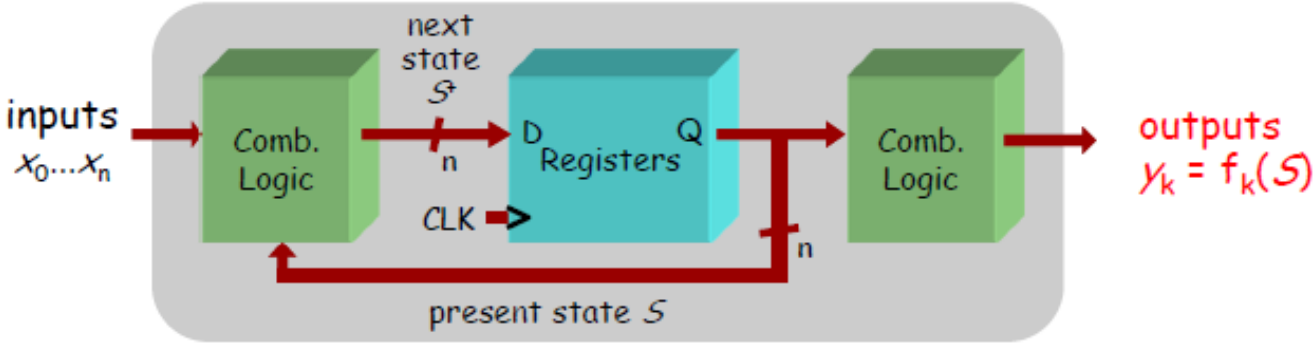
- Finite State Machines (FSMs) are a useful abstraction for sequential circuits with centralized "states" of operation
- At each clock edge, combinational logic computes *outputs* and *next state* as a function of *inputs* and *present state*



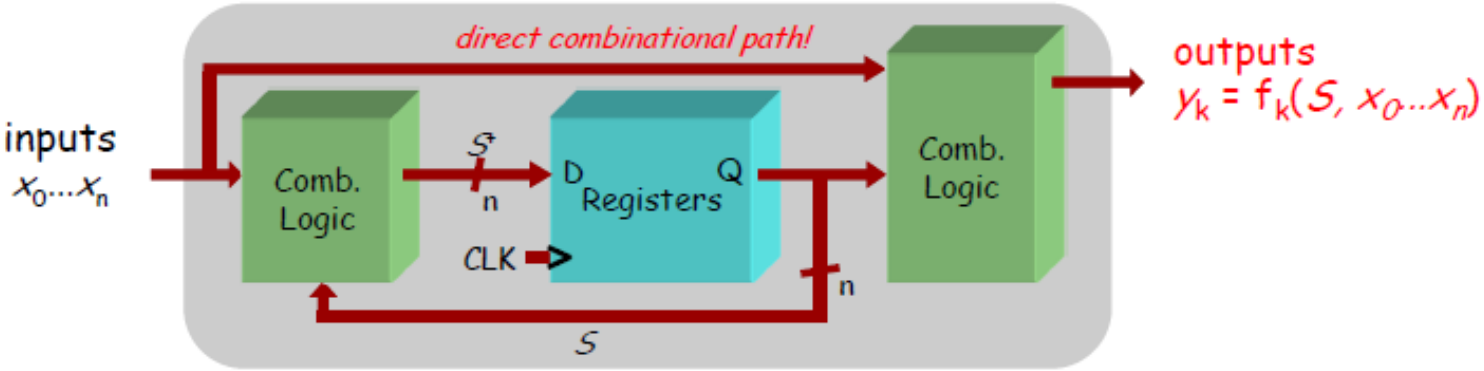
Two Types of FSMs

Moore and Mealy FSMs : different output generation

- Moore FSM:



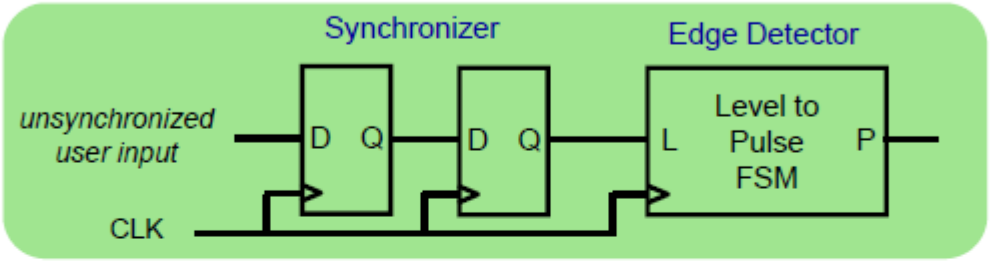
- Mealy FSM:



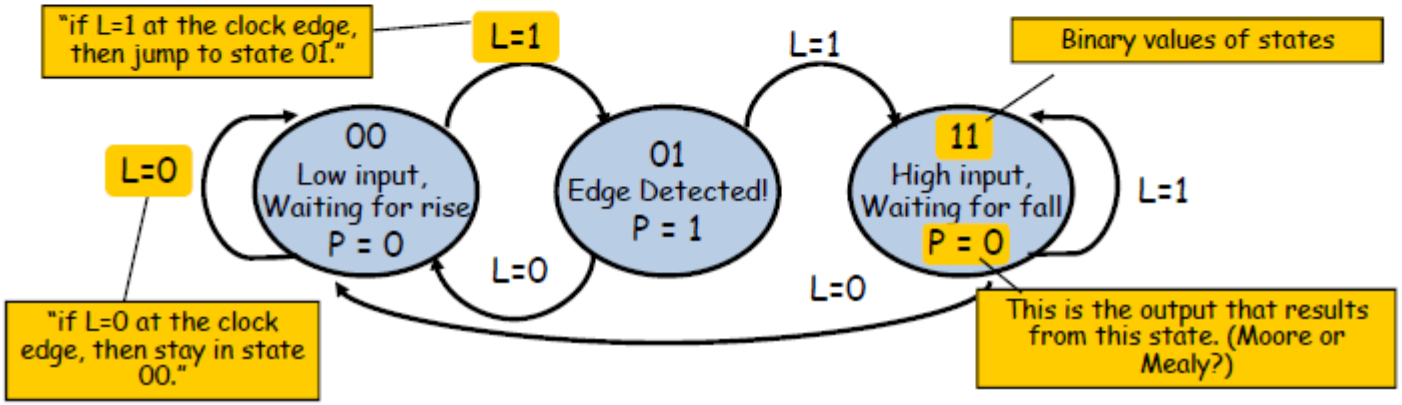
有限状态机FSM设计

Step 1: State Transition Diagram

- Block diagram of desired system:

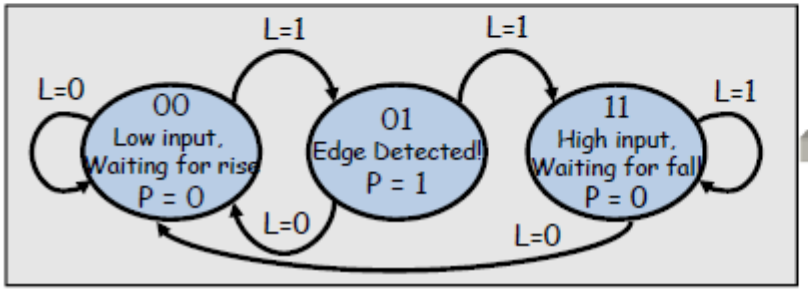


- State transition diagram is a useful FSM representation and design aid:



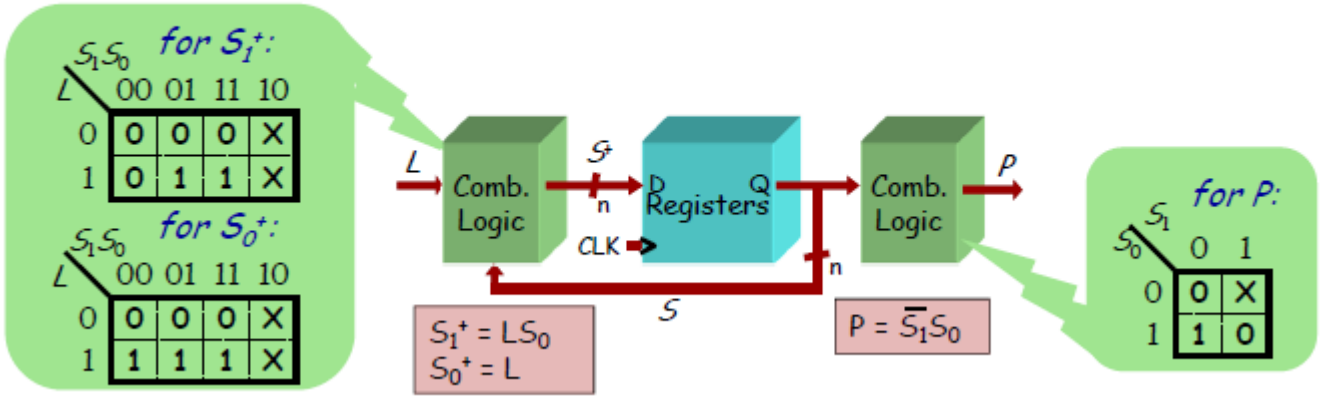
Step 2: Logic Derivation

Transition diagram is readily converted to a state transition table (just a truth table)

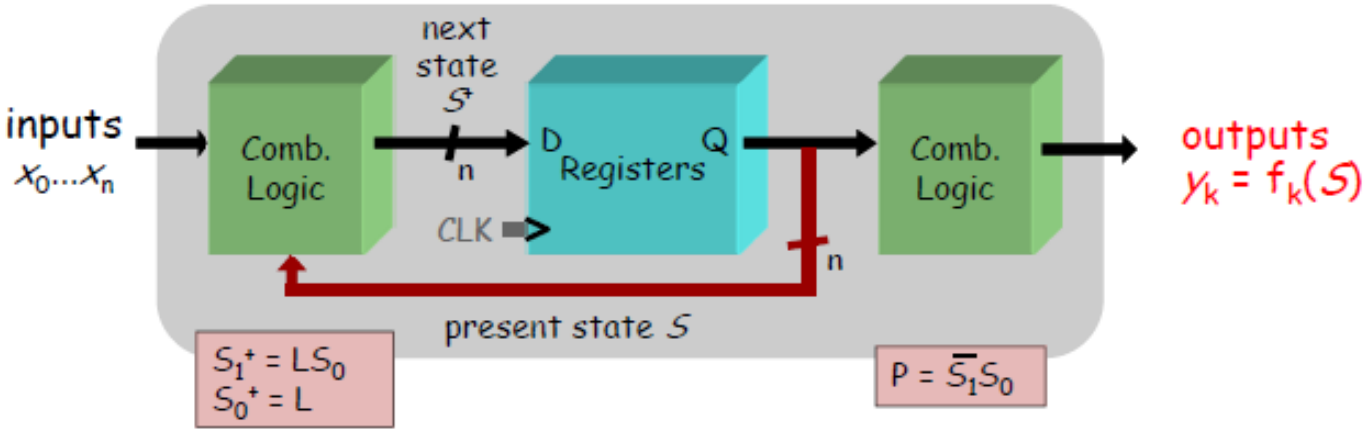


Current State		In	Next State		Out
S_1	S_0	L	S_1^+	S_0^+	P
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	1
1	1	0	0	0	0
1	1	1	1	1	0

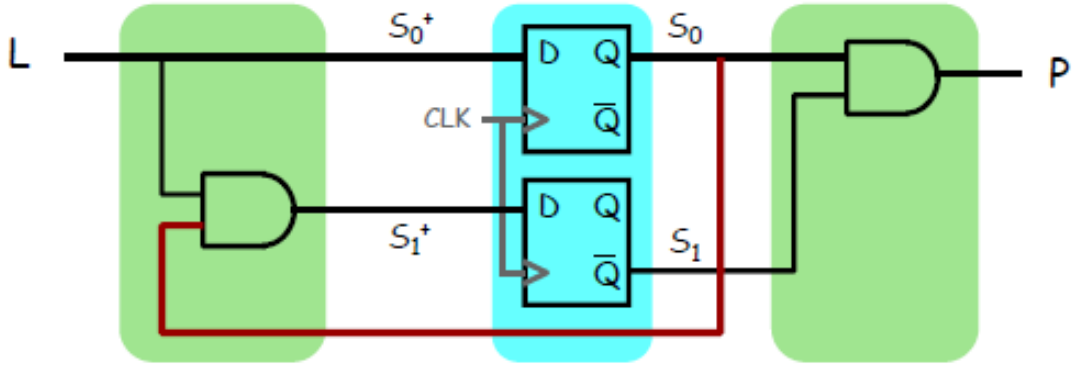
- Combinational logic may be derived using Karnaugh maps



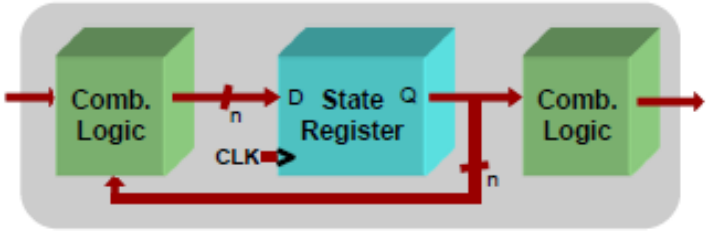
Moore Level-to-Pulse Converter



Moore FSM circuit implementation of level-to-pulse converter:



FSM（输出组合电路）



FSMs are easy in Verilog. Simply write one of each:

- **State register**
  (sequential always block)
- **Next-state combinational logic**
  (comb. always block with case)
- **Output combinational logic block**
  (comb. always block *or* assign statements)

```
module mooreVender (
  input N, D, Q, clk, reset,
  output DC, DN, DD,
  output reg [3:0] state);

  reg  next;
```

States defined with parameter keyword

```
parameter IDLE = 0;
parameter GOT_5c = 1;
parameter GOT_10c = 2;
parameter GOT_15c = 3;
parameter GOT_20c = 4;
parameter GOT_25c = 5;
parameter GOT_30c = 6;
parameter GOT_35c = 7;
parameter GOT_40c = 8;
parameter GOT_45c = 9;
parameter GOT_50c = 10;
parameter RETURN_20c = 11;
parameter RETURN_15c = 12;
parameter RETURN_10c = 13;
parameter RETURN_5c = 14;
```

State register defined with sequential always block

```
always @ (posedge clk or negedge reset)
  if (!reset)    state <= IDLE;
  else           state <= next;
```

FSM（输出组合电路）

Next-state logic within a combinational always block

```
always @ (state or N or D or Q) begin

   case (state)
     IDLE:       if (Q) next = GOT_25c;
           else if (D) next = GOT_10c;
             else if (N) next = GOT_5c;
             else next = IDLE;

     GOT_5c:     if (Q) next = GOT_30c;
           else if (D) next = GOT_15c;
             else if (N) next = GOT_10c;
             else next = GOT_5c;

     GOT_10c:    if (Q) next = GOT_35c;
           else if (D) next = GOT_20c;
             else if (N) next = GOT_15c;
             else next = GOT_10c;

     GOT_15c:    if (Q) next = GOT_40c;
           else if (D) next = GOT_25c;
             else if (N) next = GOT_20c;
             else next = GOT_15c;

     GOT_20c:    if (Q) next = GOT_45c;
           else if (D) next = GOT_30c;
             else if (N) next = GOT_25c;
             else next = GOT_20c;

     GOT_25c:    if (Q) next = GOT_50c;
           else if (D) next = GOT_35c;
             else if (N) next = GOT_30c;
             else next = GOT_25c;

     GOT_30c:  next = IDLE;
     GOT_35c:  next = RETURN_5c;
     GOT_40c:  next = RETURN_10c;
     GOT_45c:  next = RETURN_15c;
     GOT_50c:  next = RETURN_20c;

     RETURN_20c:  next = RETURN_10c;
     RETURN_15c:  next = RETURN_5c;
     RETURN_10c:  next = IDLE;
     RETURN_5c:   next = IDLE;

     default: next = IDLE;
   endcase
 end
```

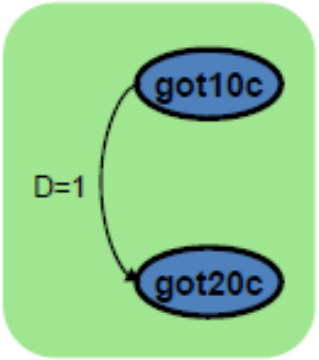
Combinational output assignment

```
assign DC = (state == GOT_30c || state == GOT_35c ||
             state == GOT_40c || state == GOT_45c ||
             state == GOT_50c);
assign DN = (state == RETURN_5c);
assign DD = (state == RETURN_20c || state == RETURN_15c ||
             state == RETURN_10c);
endmodule
```

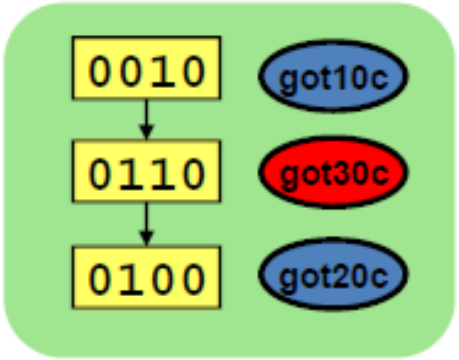
FSM Output Glitching

- FSM state bits may not transition at precisely the same time
- Combinational logic for outputs may contain hazards
- Result: your FSM outputs may glitch!

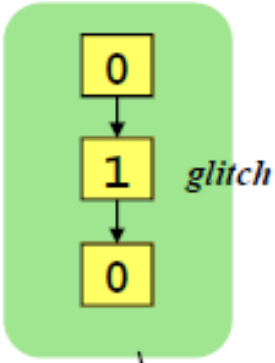
during this state transition...



...the state registers may transtion like this...

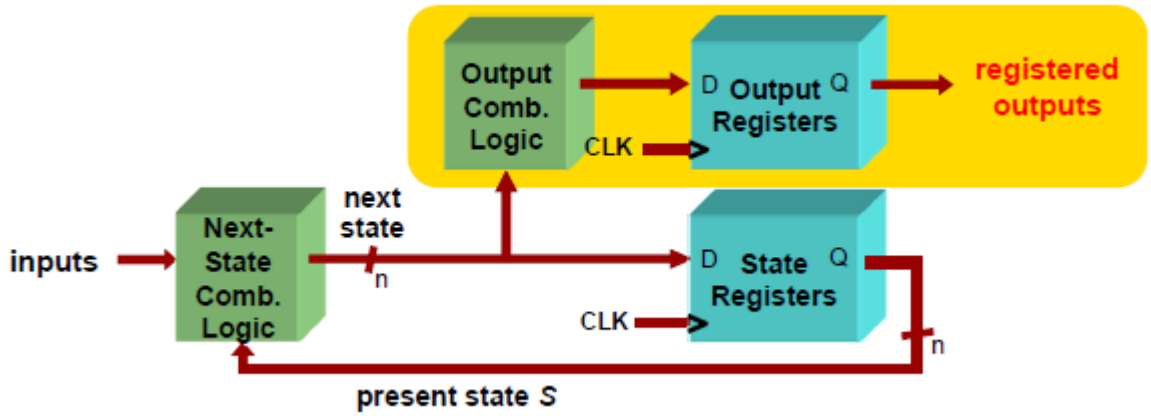


*...causing the DC output to **glitch** like this!*




```
assign DC = (state == GOT_30c || state == GOT_35c ||
             state == GOT_40c || state == GOT_45c ||
             state == GOT_50c);
```

FSM（输出寄存器电路）

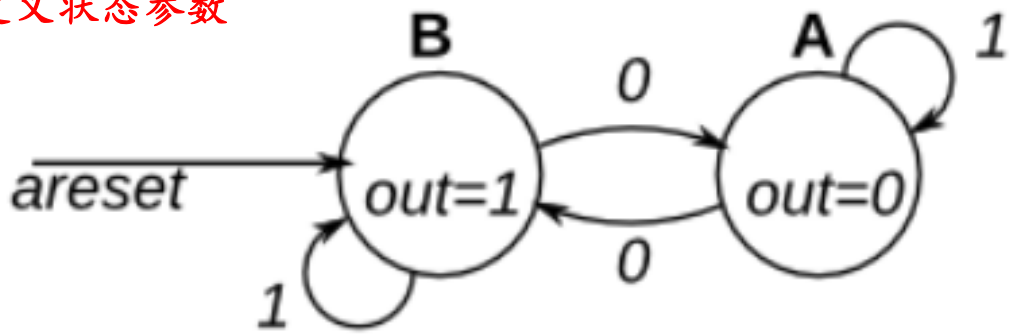


- Move output generation into the sequential always block
- Calculate outputs based on <u>next</u> state
- Delays outputs by one clock cycle. Problematic in some application.

```
reg DC,DN,DD;

 // Sequential always block for state assignment
 always @ (posedge clk or negedge reset) begin
   if (!reset)   state <= IDLE;
   else if (clk) state <= next;

   DC <= (next == GOT_30c || next == GOT_35c ||
          next == GOT_40c || next == GOT_45c ||
          next == GOT_50c);
   DN <= (next == RETURN_5c);
   DD <= (next == RETURN_20c || next == RETURN_15c ||
          next == RETURN_10c);
 end
```

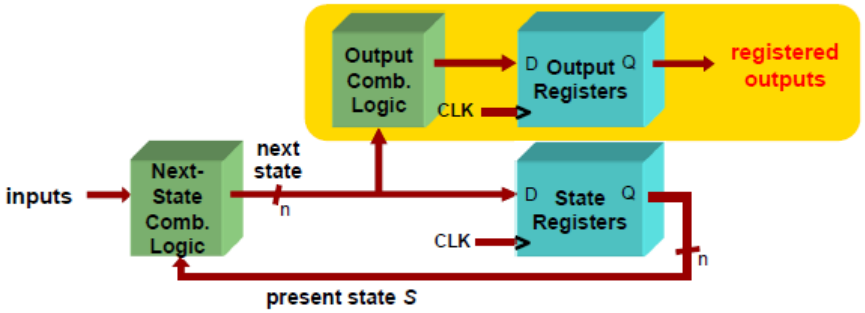
```
module top_module (
        input clk,
        input in,
        input areset,
        output out);

       parameter A=0, B=1;//定义状态参数
        reg state;
        reg next;

   always@(*)//状态转移逻辑
    begin
       case (state)
        A: next = in ? A : B;
        B: next = in ? B : A;
       endcase
     end

    always @(posedge clk, posedge areset) //状态更新
      begin
       if (areset) state <= B;
        else state <= next;
       end
     assign out = (state==B);//输出逻辑，简单输出逻辑用assign，
                                复杂用always组合块
endmodule
```

Moore型状态机，有两个状态，一个输入，一个输出。



FSM三段式代码模板（寄存器输出）

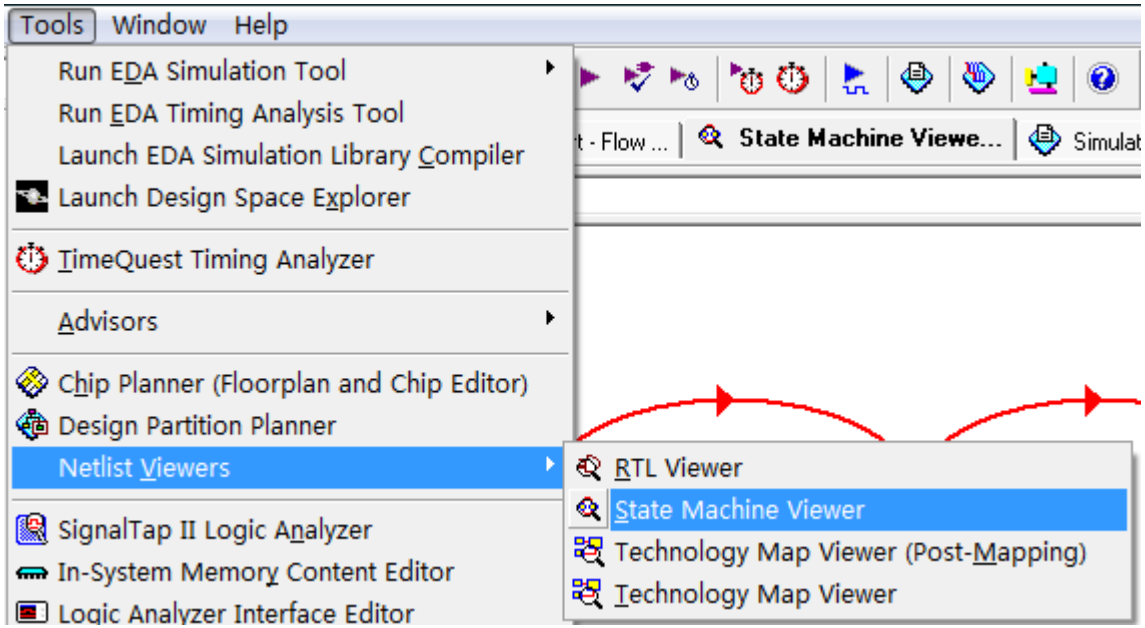
```
//组合逻辑always模块，描述状态转移
always @ (current_state) //电平触发，现态为敏感信号
begin
        next_state = x; //要初始化，使得系统复位后能进入正确的状态

        case(current_state)
        S1: if(...)

                next_state = S2; //阻塞赋值

        S2: if(...)

                next_state = S3; //阻塞赋值
            ...
          endcase
  end

//时序always模块，描述次寄存器更新
always @ (posedge clk or negedge rst_n)    //异步复位
if(!rst_n)
         current_state <= IDLE;    //注意，使用的是非阻塞赋值
else
         current_state <= next_state;

//第三个进程，同步时序always模块，描述次态寄存器输出
always @ (posedge clk or negedge rst_n)
begin

...//初始化

case(next_state)

       S1:    out1 <= 1'b1; //注意是非阻塞逻辑

        S2:     out2 <= 1'b1;

         default:... //避免综合工具综合出锁存器
endcase
end
```

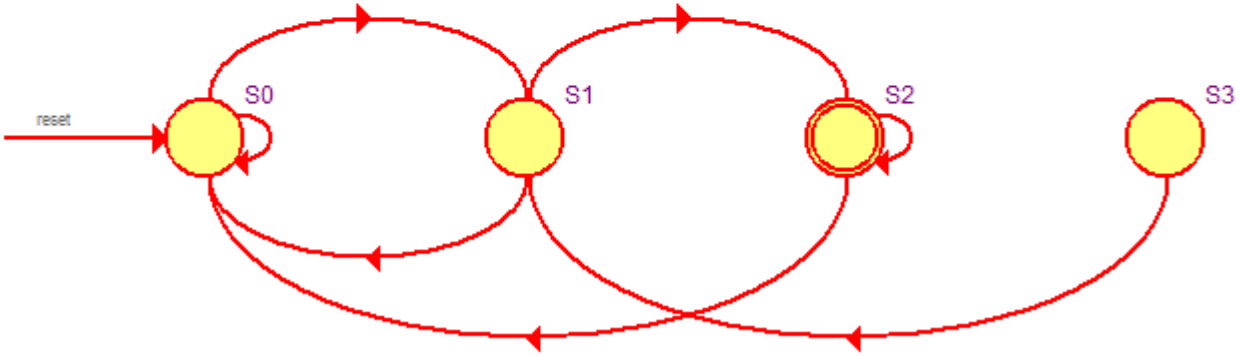


FSM设计及书写要求

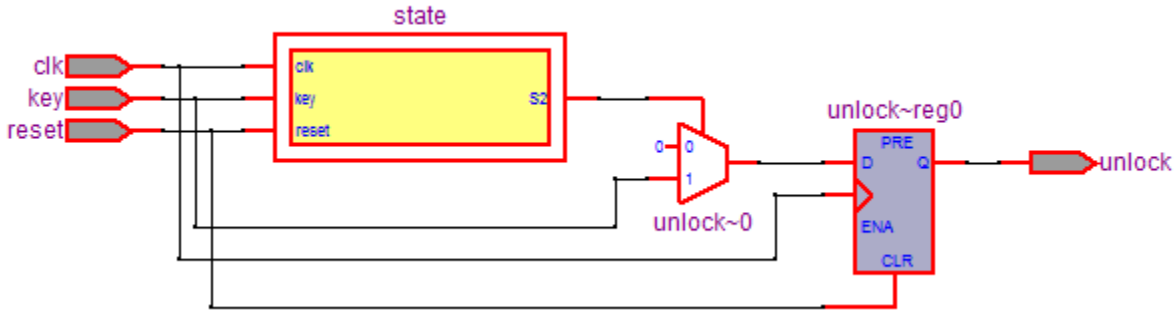
- FSM代码组成：状态转移逻辑、状态寄存器更新、输出逻辑（状态转移可通过输入或时钟或计数器控制）；
- FSM书写标准：两段式（输出逻辑由组合电路实现）或三段式（输出逻辑由实现电路实现）；
- FSM设计要安全
  - 设置初始态、异步复位键，防止死循环
  - 输出电路带寄存器，防止毛刺异常扰动

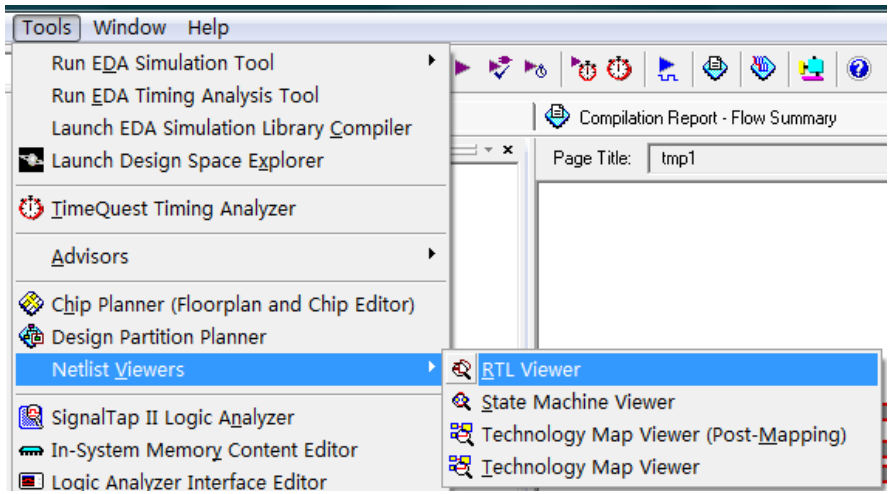
调试—查看状态图


Tools
Window
Help
Run EDA Simulation Tool
Run EDA Timing Analysis Tool
Launch EDA Simulation Library Compiler
Launch Design Space Explorer
TimeQuest Timing Analyzer
Advisors
Chip Planner (Floorplan and Chip Editor)
Design Partition Planner
Netlist Viewers
SignalTap II Logic Analyzer
In-System Memory Content Editor
Logic Analyzer Interface Editor
RTL Viewer
State Machine Viewer
Technology Map Viewer (Post-Mapping)
Technology Map Viewer
State Machine Viewe...


reset
S0
S1
S2
S3

调试—查看电路结构图





- ✓ 功能仿真
- ✓ 修改警告错误
- ✓ 查看RTL图
- ✓ 硬件调试将中间变量输出（状态变量）

EDA实验二内容

Vending machine

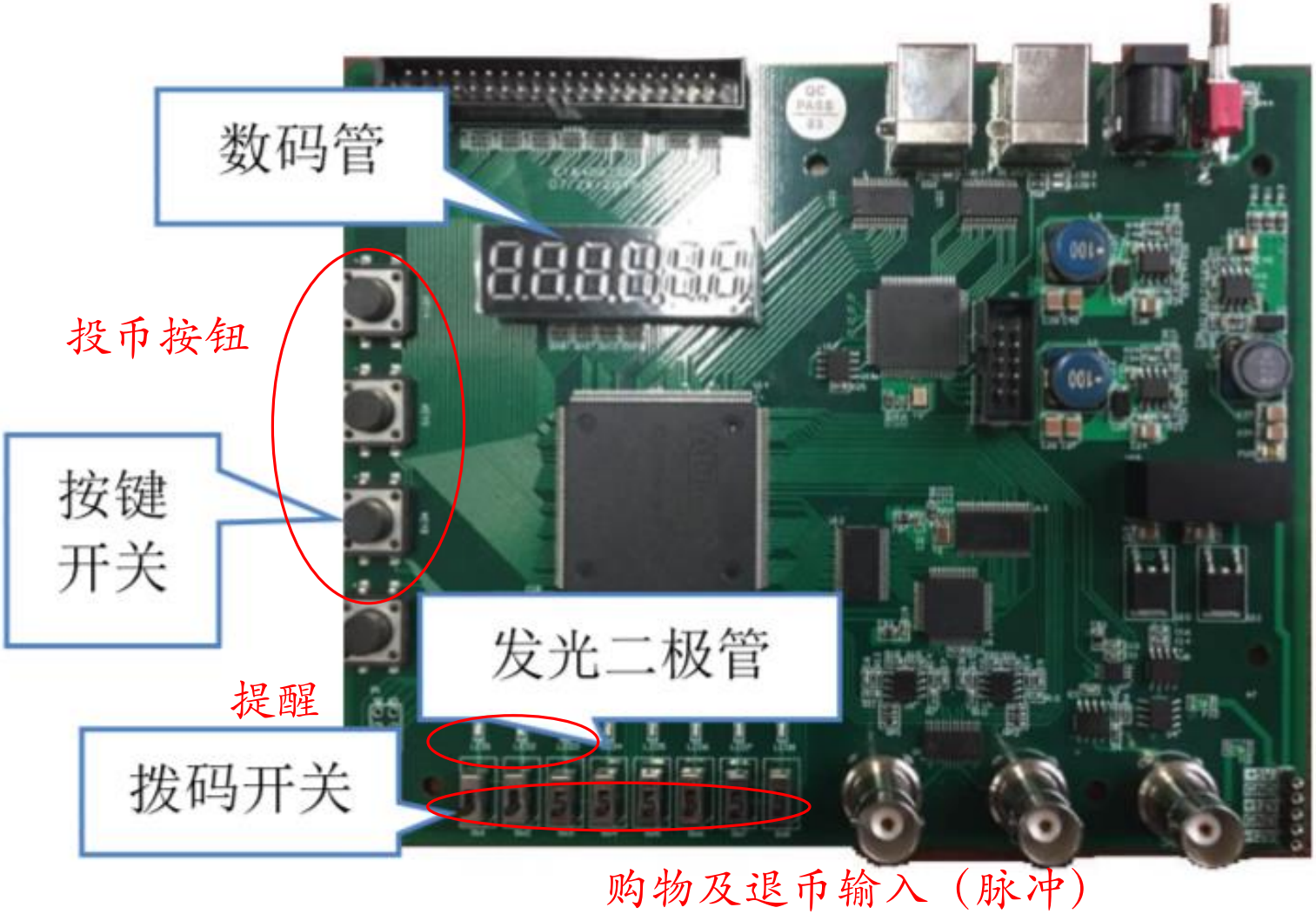
利用实验板上的拨码开关和按键开关模拟投币、购物和退币输入，用发光二极管模拟各种提示信息，用数码管显示余额，实现一个自动售货机内部控制电路。要求满足如下规格:

1）可接受5角、1元和5元的投币，每次购买允许投入多种不同币值的钱币；用3只数码管显示当前投币金额，如055表示已投币5.5元；

2）可售出价格分别为1.5元和2.5元的商品，假设用户每次购买时只选择单件、一种商品；允许用户多次购买商品，每次购买后，可以进行补充投币；

3）选择购买商品后，如果投币金额不足，则提醒；否则，售出相应的商品，并提醒用户取走商品；

4）若用户选择退币，则退回余下的钱，并提醒用户取钱。



实验板上有40MHz的时钟信号，对应FPGA引脚号为PIN_152，自动售货机的工作时钟及数码管循环扫描显示的时钟可由该40MHz分频得到。

自动售货机电路框图

- ✓ 3个按键开关模拟投币、2个拨码开关模拟购物、1个按键或拨码开关模拟退币；或者仅用4个按键开关模拟投币、购物和退币输入；
- ✓ 核心控制电路请用有限状态机实现；
- ✓ 电路应具有初态及复位功能。