**《数字逻辑》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | |  | | **年级** | |  |
| **学号** | |  | | **专业、班级** | |  |
| **实验名称** | **项目三 密码锁设计** | | | | | |
| **实验时间** | **2019.12** | | **实验地点** | | **DS1410** | |
| **实验时间** |  | | **实验地点** | |  | |
| **实验成绩** |  | | **实验性质** | | **□验证性 □设计性 □综合性** | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  评语：  评价教师签名（电子签名）： | | | | | | |
| 一、实验目的  设计一个密码锁电路。 | | | | | | |
| 二、实验项目内容  设计一个密码锁电路，可以通过拨码开关设置初始密码，通过按钮来输入密码，密码位数至少为4位，判断输入密码与设置的密码是否一致。 | | | | | | |
| 三、实验设计  实验说明：   1. 本次实验使用了两种方式进行实现。第一种方式采用状态机，第二种方式基于模块化设计。 2. 第一种方式实现功能较为完整，通过良好的仿真测试验证。 3. 第二种方式在FPGA上实现。   方案一： 顶层模块 实现功能   * 实现四位长度密码。 * 修改密码需要身份验证，输入原密码和新密码，校验成功则设置新密码成功。 * 开锁有最大尝试次数，超过最大次数则密码锁自锁。 * 操作的各种结果提示信号将持续时间T，操作成功时间T后如果没有行动会自动返回初始状态。  IO数据读入  * 使用10个拨码开关来做一个数字，（0-9），密码长度为4位（4位0-9的数字） * 按键load来读入这10个数 * 按键change来设置密码 * 按键open来打开锁 * 按键sure来确认操作 * 拨码开关rst\_in来重置输入的数字 * 拨码开关rst\_code来重置密码 * 拨码开关rst\_open来重置信号灯的显示 * 拨码开关rst\_smg来重置数码管显示 * 拨码开关rst\_timer来重置计时器 * 拨码开关rst来重置状态  数据读出  * 使用4个七段数码管显示输入的4位数字（0-9） * 使用LED灯指示剩余可尝试的次数 * 使用LED灯代表密码锁自锁 * 使用LED灯代表修改密码成功 * 使用两个LED灯分别代表succ或者fail * 使用两个LED灯分别代表当前正处于开锁或修改密码过程   模块解释   1. **计数器**  * 用于读入数据，一共有8个临时存储 * 在开锁时，默认使用前四位。 * 在修改密码时，前四个用于保存身份验证时输入的原密码，后四个表示要设置的新密码 * 每次load加载进一个数字（0-9），计数器加一，直到4位（或者8位）  1. **状态机**  * 状态机共有6个状态，分别为初始状态，开锁状态，修改密码状态，开锁成功状态，自锁状态，密码成功修改状态 * 一开始为初始状态，当判定action行为为开锁时，跳转到开锁状态，当判定action行为为修改密码时，跳转到修改密码状态 * 在开锁状态，如果开锁的结果为成功，则跳转到成功开锁状态，如果结果为失败，则停留在此状态，如果达到操作次数上限，则跳转到自锁状态 * 在修改密码状态，如果身份验证的结果为成功，则跳转到成功修改密码状态，如果为失败，则停留在此状态 * 在开锁成功状态、修改密码成功状态和自锁状态，持续时间T后自动跳转到初始状态 * rst用于状态复位，直接回到初始状态。  1. **状态输出**  * 不同的状态对应不同的信号输出 * 指示当前操作阶段的信号：change\_now表示正在修改密码，open\_now 表示正在开锁 * 指示操作结果的信号：succ表示成功开锁，fail表示开锁失败，locked表示自锁，change\_succ表示修改密码成功 * 信号的持续时间和状态的持续时间一致，为时间T  1. **激励**  * 生成action行为激励 * 修改密码和开锁不能同时进行 * 开锁对应action 01 * 修改密码对应action 10   方案二：  **设计流程**   子模块debounce  * 按键消抖  smg  * 数码管显示  decode  * 10-4编码器，将10\_0000\_0000编码为0001，代表1，以此类推编码1234567890  compare  * 等值比较器 * 给定两个值，输出succ和fail的值  lock  * 目前仅仅支持设置密码，开锁，正确就亮正确的灯，不正确就亮不正确的 * 除了消抖没有使用时钟 * 使用按键来模拟时序 * 操作模式为：输入数据，load加载，输完4位，按change表示设置密码，按open表示开锁验证。   **原理图**  UZP$TS(W0@TU3QRXO0)TF9J | | | | | | |
| 四、实验过程或算法(关键步骤、核心代码注解等）  方案一：  module **Lock**(      clk, rst\_in, rst\_code, rst\_open, rst\_timer, rst, *// 系统时钟，输入复位，密码复位，开锁复位，计时器复位，状态复位*      load, sure, change, open,   *// 数字加载， 操作确认，修改密码，开锁操作*      din, *// 输入数字*      succ, fail, locked, change\_succ, cnt\_open,*// 成功开锁信号，开锁失败信号，自锁信号，修改密码成功信号，剩余开锁次数*      open\_now, change\_now *// 当前处于开锁过程，当前处于修改密码过程*  );  input clk, rst\_in, rst\_code, rst\_open, rst\_timer, rst;  input load, sure, change, open;  input [9:0] din;  output reg succ, fail, locked, change\_succ;  output reg open\_now, change\_now;  output reg [2:0] cnt\_open;  *// 临时存储*  reg [9:0] din\_1, din\_2, din\_3, din\_4;  *// 正确密码，输入的密码*  reg [39:0] right\_code, in\_code;  *// 状态机当前状态，下一状态*  reg [2:0] currentState, nextState;  parameter START\_STATE = 3'b000; *// 初始*  parameter OPEN\_STATE = 3'b001; *// 开锁*  parameter CHANGE\_STATE = 3'b010; *// 设置密码*  parameter PASS\_STATE = 3'b011; *// 成功开锁*  parameter LOCKED\_STATE = 3'b100; *// 锁定状态*  parameter CHANGE\_SUCC\_STATE = 3'b101; *// 成功修改密码*  *// 状态更新*  always @(posedge clk or posedge rst) begin      if (rst) currentState <= START\_STATE;      else currentState <= nextState;  end  */\*\**  *计数器，用于读入数据*  *一共有8个临时存储*  *在开锁时，默认使用前四位。*  *在修改密码时，前四个用于保存身份验证时输入的原密码，后四个表示要设置的新密码*  *每次load加载进一个数字（0-9），计数器加一，直到4位（或者8位）*  *\*/*  reg [3:0] cnt;  always @(rst\_in or din or load) begin      if(!rst\_in) begin          cnt <= 4'b1000;          din\_1 <= 0;          din\_2 <= 0;          din\_3 <= 0;          din\_4 <= 0;          din\_temp\_1 <= 0;          din\_temp\_2 <= 0;          din\_temp\_3 <= 0;          din\_temp\_4 <= 0;          in\_code = 0;      end      else if (load)begin          if(cnt == 4'b0000) begin              cnt <= 4'b0000;          end          else begin              if(cnt ==4'b1000) din\_4 <= din;              else if(cnt == 4'b0111) din\_3 <= din;              else if(cnt == 4'b0110) din\_2 <= din;              else if(cnt == 4'b0101) din\_1 <= din;              else if(cnt == 4'b0100) din\_temp\_4 <= din;              else if(cnt == 4'b0011) din\_temp\_3 <= din;              else if(cnt == 4'b0010) din\_temp\_2 <= din;              else if(cnt == 4'b0001) din\_temp\_1 <= din;              cnt = cnt - 1'b1;          end      end  end  */\**  *状态机共有6个状态，分别为初始状态，开锁状态，修改密码状态，开锁成功状态，自锁状态，密码成功修改状态*  *一开始为初始状态，当判定action行为为开锁时，跳转到开锁状态，当判定action行为为修改密码时，跳转到修改密码状态*  *在开锁状态，如果开锁的结果为成功，则跳转到成功开锁状态，如果结果为失败，则停留在此状态，如果达到操作次数上限，则跳转到自锁状态*  *在修改密码状态，如果身份验证的结果为成功，则跳转到成功修改密码状态，如果为失败，则停留在此状态*  *在开锁成功状态、修改密码成功状态和自锁状态，持续时间T后自动跳转到初始状态*  *rst用于状态复位，直接回到初始状态*  *\*/*  always @(currentState or action or open\_result or time\_stop or change\_result or rst) begin      if (rst) nextState <= START\_STATE;      else begin          case (currentState)              START\_STATE :                  if(action == 2'b01) nextState <= OPEN\_STATE;                  else if (action == 2'b10) nextState <= CHANGE\_STATE;                  else nextState <= START\_STATE;              OPEN\_STATE :                  if(open\_result == 2'b01) nextState <= PASS\_STATE;                  else if (open\_result == 2'b10) nextState <= OPEN\_STATE;                  else if (open\_result == 2'b11) nextState <= LOCKED\_STATE; *//  debug*                  else nextState <= START\_STATE;                CHANGE\_STATE :                  if(change\_result == 2'b01) nextState <= CHANGE\_SUCC\_STATE;                  else if(change\_result == 2'b10) nextState <= CHANGE\_STATE;                  else nextState <= START\_STATE;                PASS\_STATE :                  if(time\_stop == 0) nextState <= PASS\_STATE;                  else nextState <= START\_STATE;              LOCKED\_STATE :                  if(time\_stop == 0) nextState <= LOCKED\_STATE;                  else nextState <= START\_STATE;              CHANGE\_SUCC\_STATE :                  if(time\_stop == 0) nextState <= CHANGE\_SUCC\_STATE;                  else nextState <= START\_STATE;              default: nextState <= START\_STATE;          endcase      end  end  */\**  *不同的状态对应不同的信号输出*  *指示当前操作阶段的信号：change\_now表示正在修改密码，open\_now 表示正在开锁*  *指示操作结果的信号：succ表示成功开锁，fail表示开锁失败，locked表示自锁，change\_succ表示修改密码成功*  *信号的持续时间和状态的持续时间一致，为时间T*  *\*/*  always @(rst or currentState) begin      if (rst) begin          open\_now <= 0;          change\_now <= 0;          succ <= 0;          fail <= 0;          locked <= 0;      end      else begin          case (currentState)              START\_STATE :begin              open\_now <= 0;              change\_now <= 0;              succ <= 0;              fail <= 0;              locked <= 0;              change\_succ = 0;              end              OPEN\_STATE :begin                  open\_now <= 1;                  fail <= 1;                  change\_now <= 0;                  succ <= 0;                  locked <= 0;                  change\_succ = 0;              end              CHANGE\_STATE:begin                  change\_now <= 1;                  open\_now <= 0;                  succ <= 0;                  fail <= 0;                  locked <= 0;                  change\_succ = 0;              end              PASS\_STATE:begin                  succ <= 1;                  open\_now <= 0;                  change\_now <= 0;                  fail <= 0;                  locked <= 0;                  change\_succ = 0;              end              LOCKED\_STATE:begin                  locked <= 1;                  fail <= 1;                  open\_now <= 0;                  change\_now <= 0;                  succ <= 0;                  change\_succ = 0;              end              CHANGE\_SUCC\_STATE :begin                  locked <= 0;                  fail <= 0;                  open\_now <= 0;                  change\_now <= 0;                  succ <= 0;                  change\_succ = 1;              end          endcase      end  end  */\**  *生成action行为激励*  *修改密码和开锁不能同时进行*  *开锁对应action 01*  *修改密码对应action 10 \*/*  reg [1:0] action; *// 01 - change 10 - open*  always@(change or open) begin      if(change && !open) action = 2'b10;      else if(!change && open) action = 2'b01;      else action = 2'b00;  end  */\**  *激励信号*  *开锁和修改密码的结果生成逻辑 \*/*  reg [1:0] open\_result;  always @(posedge sure or posedge open or posedge change or negedge rst\_open or negedge rst\_code) begin      if(!rst\_open)begin          cnt\_open <= 3'b100;          open\_result <= 3'b000;      end      else if(!rst\_code) begin          right\_code <= 0;          change\_result <= 2'b10;      end      else begin  *// 如果当前状态为初始状态，若要开锁，则初始化计时器和开锁结果*      case (currentState)      START\_STATE:begin          if(open) begin              cnt\_open <= 3'b100;              open\_result <= 3'b010;          end          else if (change) begin              change\_result <= 2'b10;          end      end  */\**  *如果当前状态为开锁，当按下确认sure，比较当前输入的密码和正确密码，如果正确则结果为成功开锁*  *否则计数器减一，当剩余操作次数为0的时候，自锁 \*/*        OPEN\_STATE: begin      if(sure) begin          if(cnt\_open == 3'b000) begin              open\_result <= 2'b11;              cnt\_open <= 3'b000;          end          else begin              if({din\_4, din\_3, din\_2, din\_1} == right\_code) begin                  open\_result <= 2'b01;                  cnt\_open <= 3'b100;              end else begin                  open\_result <= 2'b10;                  cnt\_open <= cnt\_open - 1'b1;              end  *// 将数据拼接*              in\_code <= {din\_4, din\_3, din\_2, din\_1};          end      end      end  */\* 如果当前状态 为修改密码*  *进行身份验证，输入原密码和新密码，校验成功则设置成功，否则设置失败*  *\*/*      CHANGE\_STATE:begin      if(currentState == CHANGE\_STATE) begin          if ({din\_4, din\_3, din\_2, din\_1} == right\_code) begin              right\_code <= {din\_temp\_4, din\_temp\_3, din\_temp\_2, din\_temp\_1};              change\_result <= 2'b01;          end          else begin              change\_result <= 2'b10;          end      end      end        endcase      end  end    *//change result*  reg [9:0] din\_temp\_1, din\_temp\_2, din\_temp\_3, din\_temp\_4;  reg [1:0] change\_result;  *// 计时模块，用于状态返回*  integer cnt\_timer;  *//parameter T = 1000000000; // 10s*  parameter T = 20; *// 10s*  reg time\_stop; *// 5s倒计时*  always @(posedge clk or negedge rst\_timer) begin      if(!rst\_timer) begin          time\_stop <= 0;          cnt\_timer <= 0;      end      else if(currentState == PASS\_STATE || currentState == LOCKED\_STATE || currentState == CHANGE\_STATE) begin          if(cnt\_timer < T) begin              cnt\_timer = cnt\_timer + 1'b1;              time\_stop <= 0;          end else begin              time\_stop <= 1;              cnt\_timer <= 0;          end      end  end  endmodule *// Lock\_stastusssssss*  s  由于开发板资源限制，在实际下板实验时采用了模块化设计方式，即方案二，取消了状态机设计，对项目进行了简化，但值得指出，仍然可以体现本项目设计的正确性，完全符合项目要求的全部要求，可以实现正常的功能。  下面是子模块代码。  module **Compare**(//比较模块      din\_1, din\_2, succ, fail  );  input [39:0] din\_1, din\_2;  output succ, fail;  assign succ = (din\_1 == din\_2) ? 1 : 0;  assign fail = ~succ;  endmodule *// Compare*  module **debounce**(//按键消抖模块      input clk,      input load,      input change,      input open,      output load\_o,      output change\_o,      output open\_o      );      reg load\_rrr, load\_rr, load\_r;      reg change\_rrr, change\_rr, change\_r;      reg  open\_rrr, open\_rr,  open\_r;      always @(posedge clk)      begin          load\_rrr = load\_rr;          load\_rr = load\_r;          load\_r = load;            change\_rrr = change\_rr;          change\_rr = change\_r;          change\_r = change;            open\_rrr = open\_rr;          open\_rr = open\_r;          open\_r = open;      end      assign load\_o = load\_rrr && load\_rr && load\_r;      assign change\_o = change\_rrr && change\_rr & change\_r;      assign open\_o = open\_rrr && open\_rr && open\_r;  endmodule  module **Decode**(//decode模块      din, dout  );  input [9:0] din;  output reg [3:0] dout;  always @ (din)       begin       case (din)          10'b00\_0000\_0001 : dout<=4'b0000;          10'b00\_0000\_0010 : dout<=4'b1001;          10'b00\_0000\_0100 : dout<=4'b1000;          10'b00\_0000\_1000 : dout<=4'b0111;          10'b00\_0001\_0000 : dout<=4'b0110;          10'b00\_0010\_0000 : dout<=4'b0101;          10'b00\_0100\_0000 : dout<=4'b0100;          10'b00\_1000\_0000 : dout<=4'b0011;          10'b01\_0000\_0000 : dout<=4'b0010;          10'b10\_0000\_0000 : dout<=4'b0001;           default : dout<= 4'b0000;       endcase  end  endmodule *//*  module **smg**(clk,data,sm\_wei,sm\_duan,rst);//数码管显示模块  input clk,rst;  input [15:0] data;  output [3:0] sm\_wei;  output [7:0] sm\_duan;  *//----------------------------------------------------------*  *//分频：更新频率为100Hz*  integer clk\_cnt;    *//分频周期计数*  reg clk\_100Hz;  always @(posedge clk)  if(rst == 1) begin  *//重置数码管分频显示*      clk\_cnt <= 1'b0;      clk\_100Hz <= 1'b0;  end  else if(clk\_cnt == 32'd25000)  begin      clk\_cnt <= 1'b0;  *//半个周期结束，重新计数*      clk\_100Hz <= ~clk\_100Hz;    *//时钟翻转*  end  else clk\_cnt <= clk\_cnt + 1'b1;  *//分频周期计数+1，又过去一个时钟周期*  *//----------------------------------------------------------*  *//位控制*  *//reg [2:0]wei\_ctrl;*  *//always @(posedge clk\_100Hz)*  *//begin*  *//    if(rst == 1) begin*  *//        wei\_ctrl <= 0;*  *//    end*  *//    else begin*  *//    wei\_ctrl <= wei\_ctrl + 1;*  *//    if(wei\_ctrl == 3'b100) wei\_ctrl <= 3'b0;*  *//    end*  *//end*  *//让数码管于每个分频了的时钟周期在4个位交替闪烁*  *//利用人眼视觉暂留，让人看到四个数码管显示的不同内容*  reg [3:0] wei\_ctrl=4'b1110;  always @(posedge clk\_100Hz)      wei\_ctrl <= {wei\_ctrl[2:0],wei\_ctrl[3]};  *//段控制：每一位的数码管的8个led灯的输入*  reg [3:0]duan\_ctrl;  always @(wei\_ctrl)      case(wei\_ctrl)          4'b1110:duan\_ctrl=data[3:0];          4'b1101:duan\_ctrl=data[7:4];          4'b1011:duan\_ctrl=data[11:8];          4'b0111:duan\_ctrl=data[15:12];          default:duan\_ctrl=4'hf;        endcase  *//----------------------------------------------------------*  *//解码模块：8个led灯的输入的解码，让led灯对应的位闪烁（低电平亮灯）*  reg [7:0]duan;  always @(duan\_ctrl)      case(duan\_ctrl)          4'h0:duan=8'b1100\_0000;*//0*          4'h1:duan=8'b1111\_1001;*//1*          4'h2:duan=8'b1010\_0100;*//2*          4'h3:duan=8'b1011\_0000;*//3*          4'h4:duan=8'b1001\_1001;*//4*          4'h5:duan=8'b1001\_0010;*//5*          4'h6:duan=8'b1000\_0010;*//6*          4'h7:duan=8'b1111\_1000;*//7*          4'h8:duan=8'b1000\_0000;*//8*          4'h9:duan=8'b1001\_0000;*//9*          4'ha:duan=8'b1000\_1000;*//a*          4'hb:duan=8'b1000\_0011;*//b*          4'hc:duan=8'b1100\_0110;*//c*          4'hd:duan=8'b1010\_0001;*//d*          4'he:duan=8'b1000\_0110;*//e*          4'hf:duan=8'b1000\_1110;*//f*          default : duan = 8'b1100\_0000;*//0*      endcase  *//----------------------------------------------------------*  assign sm\_wei = wei\_ctrl;  assign sm\_duan = duan;  endmodule  下面是实际下板实验时的顶层模块代码。  module Lock(      clk, rst\_in, rst\_code, rst\_open,load, din, change, open, succ, fail, sm\_wei, sm\_duan, rst\_smg  );  input clk, rst\_in, rst\_code, rst\_open, load;  input change, open;  input [9:0] din;  output succ, fail;  input rst\_smg;  wire load\_o, open\_o, change\_o;  // debounce  debounce deb(clk, load, change, open, load\_o, change\_o, open\_o);  reg [9:0] din\_1, din\_2, din\_3, din\_4;  reg [39:0] right\_code, in\_code;  reg [2:0] cnt;  // read data  always @(posedge load\_o or negedge rst\_in) begin      if(!rst\_in) begin          cnt = 3'b100;          din\_1 = 0;          din\_2 = 0;          din\_3 = 0;          din\_4 = 0;          in\_code = 0;      end      else begin          if(cnt == 3'b000) begin              cnt = 3'b000;          end          else begin              if(cnt == 3'b100) din\_4 = din;              else if(cnt == 3'b011) din\_3 = din;              else if(cnt == 3'b010) din\_2 = din;              else if(cnt == 3'b001) din\_1 = din;              cnt = cnt - 1'b1;          end      end  end  // set password  always @(posedge change\_o or negedge rst\_code) begin      if(!rst\_code) begin          right\_code = 0;      end else begin          right\_code = {din\_4, din\_3, din\_2, din\_1};      end  end  reg after\_open;  // open lock  always @(posedge open\_o or negedge rst\_open) begin          if(!rst\_open) begin              after\_open <= 0;          end          else begin              in\_code <= {din\_4, din\_3, din\_2, din\_1};              after\_open <= 1;           end  end  wire succ\_cmp, fail\_cmp;  assign succ = (after\_open) ? succ\_cmp : 0;  assign fail = (after\_open) ? fail\_cmp : 0;  // compare  Compare comp(right\_code, in\_code, succ\_cmp, fail\_cmp);  // decode  wire [3:0] dout\_1, dout\_2, dout\_3, dout\_4;  Decode decode\_0(din\_1, dout\_1);  Decode decode\_1(din\_2, dout\_2);  Decode decode\_2(din\_3, dout\_3);  Decode decode\_3(din\_4, dout\_4);  wire [15:0] data;  assign data[15:12] = dout\_1;  assign data[11:8] = dout\_2;  assign data[7:4] = dout\_3; //不需要的计算位，将其置为0  assign data[3:0] = dout\_4;  output [3:0] sm\_wei;    //位控制信号  output [7:0] sm\_duan;   //段控制信号  //----------------------------------------------------------  wire [3:0]sm\_wei;  wire [7:0]sm\_duan;  // smg display  smg mysmg (.clk(clk),.data(data),.sm\_wei(sm\_wei),.sm\_duan(sm\_duan),.rst(rst\_smg));  endmodule // Lock | | | | | | |
| 1. 实验过程中遇到的问题及解决情况 2. 使用状态机实现电子密码锁时，操作逻辑和FPGA硬件存在一定的矛盾。如上述设计，模拟按键change或者open来触发相应的always块，由于一次操作只需要触发一次，因此使用了posedge边沿触发，然而，对于FPGA来说，触发器只允许存在单时钟，另外只同时允许复位信号的存在。由于逻辑需要，例如在按下open进入开锁和按下sure确认开锁时，会涉及对同一寄存器值的改变，因此上述代码在同一个always块里使用了多个posedge边沿触发信号，相当于多时钟，虽然通过了良好的仿真测试，但无法综合。   **解决方法：**  调整逻辑设计，不再使用“按键”的逻辑来修改密码或者开锁，而是使用电平来表示一个持续的过程，即在开锁时，将开锁操作信号置高电平，输入密码并打开后，再将开锁操作信号置低电平。修改密码的逻辑类似。  此外，上述设计以“按键”的逻辑进行开锁或者修改密码，操作成功或者失败后时间T后，会自动返回初始状态。而按照现在的逻辑设计，手动将开锁信号置低电平后即结束开锁过程，如果时间T后没有结束开锁过程，将自动结束返回初始状态。  因此对部分代码进行了改进（只展示核心部分）：  always @(sure or open or change or rst\_open or rst\_code) begin      if(!rst\_open)begin          cnt\_open <= 3'b100;          open\_result <= 3'b000;      end      else if(!rst\_code) begin          right\_code <= 0;          change\_result <= 2'b10;      end      else begin      // 如果当前状态为初始状态，若要开锁，则初始化计时器和开锁结果      case (currentState)      START\_STATE:begin          if(open) begin              cnt\_open <= 3'b100;              open\_result <= 3'b010;          end          else if (change) begin              change\_result <= 2'b10;          end      end      /\*      如果当前状态为开锁，当按下确认sure，比较当前输入的密码和正确密码，如果正确则结果为成功开锁      否则计数器减一，当剩余操作次数为0的时候，自锁 \*/        OPEN\_STATE: begin          if(sure) begin          if(cnt\_open == 3'b000) begin              open\_result <= 2'b11;              cnt\_open <= 3'b000;          end          else begin              if({din\_4, din\_3, din\_2, din\_1} == right\_code) begin                  open\_result <= 2'b01;                  cnt\_open <= 3'b100;              end else begin                  open\_result <= 2'b10;                  cnt\_open <= cnt\_open - 1'b1;              end              // 将数据拼接              in\_code <= {din\_4, din\_3, din\_2, din\_1};          end      end      end      /\* 如果当前状态 为修改密码          进行身份验证，输入原密码和新密码，校验成功则设置成功，否则设置失败          \*/      CHANGE\_STATE:begin         if(sure) begin          if ({din\_4, din\_3, din\_2, din\_1} == right\_code) begin              right\_code <= {din\_temp\_4, din\_temp\_3, din\_temp\_2, din\_temp\_1};              change\_result <= 2'b01;          end          else begin              change\_result <= 2'b10;          end      end      end      endcase      end  end  此外，取消了action激励，而是直接将change或者open作为状态机的激励输入。  always @(currentState or change or open or open\_result or time\_stop or change\_result or rst) begin      if (rst) nextState <= START\_STATE;      else begin          case (currentState)              START\_STATE :                  if(!change && open) nextState <= OPEN\_STATE;                  else if (!open && change) nextState <= CHANGE\_STATE;                  else nextState <= START\_STATE;              OPEN\_STATE :                  if(open\_result == 2'b01) nextState <= PASS\_STATE;                  else if (open\_result == 2'b10) nextState <= OPEN\_STATE;                  else if (open\_result == 2'b11) nextState <= LOCKED\_STATE; //  debug                  else nextState <= START\_STATE;                CHANGE\_STATE :                  if(change\_result == 2'b01) nextState <= CHANGE\_SUCC\_STATE;                  else if(change\_result == 2'b10) nextState <= CHANGE\_STATE;                  else nextState <= START\_STATE;                PASS\_STATE :                  if(time\_stop == 0 || open == 1) nextState <= PASS\_STATE;                  else nextState <= START\_STATE;              LOCKED\_STATE :                  if(time\_stop == 0 || open == 1) nextState <= LOCKED\_STATE;                  else nextState <= START\_STATE;              CHANGE\_SUCC\_STATE :                  if(time\_stop == 0 || change == 1) nextState <= CHANGE\_SUCC\_STATE;                  else nextState <= START\_STATE;              default: nextState <= START\_STATE;          endcase      end  end   1. 原先的状态机，状态编码使用的为普通二进制编码，在状态转换时改变的较多。   解决：使用格雷编码，相邻状态转换时只有一个状态位发生翻转，这样不仅能消除状态转换时由多条状态信号线的传输延迟所造成的毛刺，又可以降低功耗。      parameter START\_STATE = 3'b000; // 初始      parameter OPEN\_STATE = 3'b001; // 开锁      parameter CHANGE\_STATE = 3'b011; // 设置密码      parameter PASS\_STATE = 3'b010; // 成功开锁      parameter LOCKED\_STATE = 3'b110; // 锁定状态      parameter CHANGE\_SUCC\_STATE = 3'b111; // 成功修改密码   1. 方案二在FPGA上实现时，初始化密码之后，由于初始化的密码和初始化的输入密码相同，会在还没有输入密码的时候，就显示密码正确   解决： 加入after\_open变量，只有当选择open模式，该变量才为1，当该变量为1且密码正确，表明正确的灯亮；当该变量为1而密码错误，表明错误的灯亮。 | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程   ①顶层模块使用状态机编写时的仿真  `timescale 1ns / 1ps  *//////////////////////////////////////////////////////////////////////////////////*  *// Company:*  *// Engineer:*  *//*  *// Create Date: 2019/12/28 22:22:51*  *// Design Name:*  *// Module Name: sim\_lock\_imp*  *// Project Name:*  *// Target Devices:*  *// Tool Versions:*  *// Description:*  *//*  *// Dependencies:*  *//*  *// Revision:*  *// Revision 0.01 - File Created*  *// Additional Comments:*  *//*  *//////////////////////////////////////////////////////////////////////////////////*  `timescale 1ns / 1ps  *//////////////////////////////////////////////////////////////////////////////////*  *// Company:*  *// Engineer:*  *//*  *// Create Date: 2019/12/27 14:59:03*  *// Design Name:*  *// Module Name: sim\_lock*  *// Project Name:*  *// Target Devices:*  *// Tool Versions:*  *// Description:*  *//*  *// Dependencies:*  *//*  *// Revision:*  *// Revision 0.01 - File Created*  *// Additional Comments:*  *//*  *//////////////////////////////////////////////////////////////////////////////////*  module **sim\_lock\_imp**;  reg clk, rst\_in, rst\_code, rst\_open, load, change, open, rst, rst\_timer;  reg [9:0] din;  reg sure;  wire rest\_times;  wire succ, fail, locked, open\_now, change\_now;  wire change\_succ;  initial begin      clk = 0;      rst\_timer = 0;      rst = 0;      rst\_code = 0;      rst\_in = 0;      rst\_open = 0;      load = 0;      change = 0;        sure = 0;      open = 0;      #10 rst\_code = 1; rst\_in = 1; rst\_open = 1; rst\_timer = 1;      #10 rst\_code = 0; rst\_in = 0; rst\_open = 0; rst\_timer = 1;      #10 rst\_code = 1; rst\_in = 1; rst\_open = 1; rst\_timer = 1;      #50  *// #10 rst\_code = 1; rst\_in = 1;*    *// 默认密码为0000*      #10 change = 1;      #10 change = 0;  *// 初始化密码*      #10 rst\_code = 0;      #10 rst\_code = 1;      #10 din = 10'b00\_0000\_0000;      #10 load = 1;      #10  load = 0;          #10 din = 10'b00\_0000\_0000;      #10 load = 1;      #10  load = 0;          #10 din = 10'b00\_0000\_0000;      #10 load = 1;      #10  load = 0;          #10 din = 10'b00\_0000\_0000;      #10 load = 1;      #10  load = 0;    *// 1*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// set password*  *//    #20 change = 1;*  *//    #10 change = 0;*      #20 sure = 1;      #20 sure = 0;      #20 rst = 1;      #20 rst = 0;      #100  *// -------------------------------- open*  *// 1*      #10 rst\_open = 0;    #10 rst\_open = 1;      #10 open = 1;      #10 open = 0;     #10 rst\_in = 0;      #10 rst\_in = 1;      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// open*  *//    #20 open = 1;*  *//    #10 open = 0;*      #20 sure = 1;      #20 sure = 0;      #50 rst\_open = 0;      #50 rst\_open = 1;      #100    *// wrong password*  *// 1*      #10 open = 1;      #10 open = 0;      #10 rst\_in = 0;      #10 rst\_in = 1;      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;      #20 sure = 1;      #20 sure = 0;      #100      #10 rst\_in = 0;      #10 rst\_in = 1;      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// open*  *//    #20 open = 1;*  *//    #10 open = 0;*      #20 sure = 1;      #20 sure = 0;        #100      #10 rst\_in = 0;      #10 rst\_in = 1;      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// open*  *//    #20 open = 1;*  *//    #10 open = 0;*      #20 sure = 1;      #20 sure = 0;        #100      #10 rst\_in = 0;      #10 rst\_in = 1;      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// open*  *//    #20 open = 1;*  *//    #10 open = 0;*      #20 sure = 1;      #20 sure = 0;        #100      #10 rst\_in = 0;      #10 rst\_in = 1;      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// open*  *//    #20 open = 1;*  *//    #10 open = 0;*      #20 sure = 1;      #20 sure = 0;        #100      #10 rst\_in = 0;      #10 rst\_in = 1;      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// open*  *//    #20 open = 1;*  *//    #10 open = 0;*      #20 sure = 1;      #20 sure = 0;        #100      #10 rst\_in = 0;      #10 rst\_in = 1;      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// open*  *//    #20 open = 1;*  *//    #10 open = 0;*      #20 sure = 1;      #20 sure = 0;        #50 rst\_open = 0;      #50 rst\_open = 1;        #100  *// reset*      #10 change = 1;      #10 change = 0;      #10 rst\_in = 0;      #10 rst\_in = 1;  *// 验证原密码*      #10 rst\_in = 0;      #10 rst\_in = 1;       #10 din = 10'b00\_1000\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0100\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 新密码*      #10 din = 10'b00\_1000\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0100\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;      #10 sure = 1;      #10 sure = 0;  *//*  *//原密码*  *//*      #100      #10 rst\_in = 0;      #10 rst\_in = 1;      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *//*  *// 新密码*      #10 din = 10'b00\_1000\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0100\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;      #10 sure = 1;      #10 sure = 0;      #100  *// 应该修改成功*  *// 再次开锁*    *//之前的密码，应该不对*      #10 rst = 1;      #10 rst = 0;      #10 open = 1;      #10 open = 0;      #10 rst\_in = 0;      #10 rst\_in = 1;      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;      #10 sure = 1;      #10 sure = 0;      #100  *// 正确的密码*      #10 rst\_in = 0;      #10 rst\_in = 1;        #10 din = 10'b00\_1000\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0100\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;      #10 sure = 1;      #10 sure = 0;  end  always # 1 clk = ~clk;  Lock  lock(      clk, rst\_in, rst\_code, rst\_open, rst\_timer, rst, *// 系统时钟，输入复位，密码复位，开锁复位，计时器复位，状态复位*      load, sure, change, open,   *// 数字加载， 操作确认，修改密码，开锁操作*      din, *// 输入数字*      succ, fail, locked, change\_succ, rest\_times,*// 成功开锁信号，开锁失败信号，自锁信号，修改密码成功信号*      open\_now, change\_now *// 当前处于开锁过程，当前处于修改密码过程*  );  endmodule *//*  仿真结果：  **一 设置初始密码**     1. 初始密码为0000, 2. 按下修改密码键（change），修改初始密码，进入修改密码状态（change\_now为1）。 3. 输入原密码0000（分别输入00\_0000\_0000然后按下load加载，以此类推）和新密码6666，然后按下确认（sure）。 4. 校验成功，change\_succ为1。   **二 开锁**     1. 按下开锁键（open），进入开锁状态（open\_now为1）。 2. 同理，依次输入四位密码后，按下确认键，进行密码校验。 3. 第一次输入的密码为6666，正确，开锁成功，succ为1. 4. 进行错误密码输入测试，输入6566，确认，校验错误，可输入的次数减一（rest\_times）。 5. 继续输入，然后确认，直到达到最大输入次数，rest\_times为0，密码锁自锁。Locked变为1，持续T秒后自动返回。   **三 修改密码**     1. 修改密码，进入修改密码状态，change\_now为1. 2. 输入原密码和新密码，进行校验。第一次原密码不正确，因此没有成功设置。 3. 第二次输入正确，因此成功设置，chenge\_succ 为1. 4. 再次输入密码，发现无法开锁。而输入改后的，开锁成功。   **四 经过改进后的仿真**    可以看到，密码设置成功后change\_succ自动变为1，持续一段时间后变为0.  开锁成功后，succ变为1。  修改密码后，输入原先的密码无法开锁，而输入新的密码后校验成功，开锁成功。  ②下板实验系统的仿真  module **sim\_lock**(    );  reg clk, rst\_in, rst\_code, rst\_open, load, change, open;  reg [9:0] din;  wire succ, fail;  initial begin      clk = 0;      rst\_code = 0;      rst\_in = 0;      rst\_open = 0;      load = 0;      change = 0;      open = 0;      #10 rst\_code = 1; rst\_in = 1; rst\_open = 1;      #10 rst\_code = 0; rst\_in = 0; rst\_open = 0;      #10 rst\_code = 1; rst\_in = 1; rst\_open = 1;      #50  *// #10 rst\_code = 1; rst\_in = 1;*  *// 1*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// set password*      #20 change = 1;      #10 change = 0;      #100  *// -------------------------------- open*  *// 1*            #10 rst\_in = 0;            #10 rst\_in = 1;      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// open*      #20 open = 1;      #10 open = 0;      #50 rst\_open = 0;          #50 rst\_open = 1;      #100  *// wrong password*  *// 1*       #19 rst\_in = 0;            #10 rst\_in = 1;      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  *// open*      #20 open = 1;      #10 open = 0;      #50 rst\_open = 0;          #50 rst\_open = 1;      #100  *// reset*      #10 rst\_in = 0;          #10 rst\_in = 1;      #10 din = 10'b00\_1000\_0000;      #10 load = 1;      #10  load = 0;  *// 2*      #10 din = 10'b00\_0100\_0000;      #10 load = 1;      #10  load = 0;  *// 3*      #10 din = 10'b00\_0010\_0000;      #10 load = 1;      #10  load = 0;  *// 4*      #10 din = 10'b00\_0001\_0000;      #10 load = 1;      #10  load = 0;  end  always # 1 clk = ~clk;  Lock lock(clk, rst\_in, rst\_code,rst\_open, load, din, change, open, succ, fail);  endmodule *//*  BE~O9FZEI1W9N%5_}WYQYYN  ③开发板   1. 初始化   751707A2334D9BFFE4AE99F97D183B62   1. 设置密码   34EB3CD4EB41AA39934F5E53F0276760   1. 进行输入密码前的初始化   E980F99C52320332DBC5CF3A3A12E3D1751707A2334D9BFFE4AE99F97D183B62   1. 输入正确密码 2. 输出密码正确型号   34EB3CD4EB41AA39934F5E53F0276760   1. 输入错误密码   250975646CFED86986BEFE726DAC3C3C   1. 输出错误信号   IMG_256  七、小组分工情况说明  田润泽：负责实验设计部分（包括原理图、电路图、代码等），实验结果部分（编写仿真代码,测试等），状态机、模块设计实现，问题发掘以及优化改进，完善报告。  姚语涵：负责实验设计部分（包括原理图、电路图、代码等），负责实验结果部分（开发板的操作演示等），模块设计的架构与实现，测试并发现问题，完善报告。  尹宇慧：负责实验设计部分（包括原理图、电路图、代码等），实验结果部分（开发板的操作演示等），模块设计的架构与实现，测试并发现问题。  蔡嘉轩：整理资料，负责协同小组其他成员设计、调试程序，完善、补充相应的部分，进行状态机和模块设计的优化和改进，撰写、完善报告。 | | | | | | |