**《数字逻辑》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | 陈鹏宇，  帕肉合·帕尔哈提 | | **年级** | | 2020级，  2020级 |
| **学号** | | 20204227，  20201641 | | **专业、班级** | | 计算机科学与技术01班，  计算机科学与技术05班 |
| **实验名称** | **密码锁设计** | | | | | |
| **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  通过实验，巩固有限状态机设计方法，并能设计实现密码锁控制器。 | | | | | | |
| 1. 实验项目内容   1、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。  2、画出密码锁电路的状态转换图。  3、编写密码锁模块并仿真。  4、 编写顶层模块，综合、实验、生成 bit 流，下载到开发板进行验证。 | | | | | | |
| 三、实验设计  状态转换图  IMG_0036  电路图    t1为密码输入，t2为按键消抖，t3为密码检测  波形图 | | | | | | |
| 四、实验过程或算法  **源代码：**  /\*顶层模块\*/  module top\_keylock(  input write, reset, clk,  input [7:0] sw\_in,  input [4:0] bt\_in,  output [1:0] led,  output [7:0] seg,  output [3:0] sel,  output [9:0] out,  output [3:0] state\_out  );  /\*中间导线，用来连接各个模块\*/  wire [1:0] m0,m1,m2,m3;  wire [4:0] bt\_in\_en;  /\*初始密码写入模块\*/  key\_write t1(.write(write), .seg(seg), .sel(sel), .clk(clk), .sw\_in(sw\_in),  .m0(m0), .m1(m1), .m2(m2), .m3(m3));  /\*按键消抖模块\*/  key\_deb t2(.clk(clk), .reset(reset), .bt\_in(bt\_in), .bt\_in\_en(bt\_in\_en));  /\*密码检测模块\*/  key\_check t3(.m0(m0), .m1(m1), .m2(m2), .m3(m3), .clk(clk), .bt\_in\_en(bt\_in\_en),  .led(led), .reset(reset), .out(out), .state\_out(state\_out));  endmodule  /\*按键消抖模块\*/  module key\_deb(  input clk,  input reset,  input [4:0] bt\_in,  output reg [4:0] bt\_in\_en  );    parameter DURATION = 500000;//延时10ms  reg [19:0] cnt; //按下计数器  reg [19:0] cnto;//未按计数器  wire key\_enable;  wire bt\_in\_float;    assign key\_enable = (bt\_in[4] | bt\_in[3] | bt\_in[2] | bt\_in[1] | bt\_in[0]); //任意按键被按下  assign bt\_in\_float = ~(bt\_in[4] & bt\_in[3] & bt\_in[2] & bt\_in[1] & bt\_in[0]);//按键没被按下    always @(posedge clk or posedge reset)  begin  if(reset)  begin  bt\_in\_en <= 5'b0;  cnt <= 20'd0;  cnto <= 20'd0;  end  else if(key\_enable)  begin//计数开始  cnto <= 20'd0;//抬起计数器清零  if(cnt == DURATION-1)  begin  cnt <= cnt;  if (bt\_in[4] | bt\_in[3] | bt\_in[2] | bt\_in[1] | bt\_in[0])  begin  bt\_in\_en<=bt\_in;  cnt<=20'd0;  end  end  else  cnt <= cnt + 1'b1;  end  else if(bt\_in\_float)  begin//计数开始  cnt <= 20'd0;//按下计数器清零  if(cnto == DURATION-1)  begin  cnto <= cnto;  if(~(bt\_in[4] & bt\_in[3] & bt\_in[2] & bt\_in[1] & bt\_in[0]))  begin  bt\_in\_en <= bt\_in;  cnto<=20'd0;  end  end  else  cnto <= cnto + 1'b1;  end  else  begin  cnt <= 20'd0;  cnto <= 20'd0;  end  end  Endmodule  `timescale 1ns / 1ps  /\*密码检测模块\*/  module key\_check(m0, m1, m2, m3, clk, bt\_in\_en, led, reset, state\_out, out);  output reg [1:0] led;//输出解锁成功与否  output wire [9:0] out; //输出check2和按键状态  output wire [3:0] state\_out;//输出当前状态  input clk; //时钟信号  input reset; //重置状态  input [1:0] m0, m1, m2, m3;//memery中的密码  input [4:0] bt\_in\_en; //通过button输入密码，其中bt[0-3]分别表示00，01，10，11    reg [3:0] state; //当前状态机状态    /\*led灯状态\*/  parameter led\_begin = 2'b00;  parameter led\_true = 2'b10;  parameter led\_false = 2'b01;  /\*按键输入状态\*/  parameter put0 = 5'b00001;  parameter put1 = 5'b00010;  parameter put2 = 5'b00100;  parameter put3 = 5'b01000;  parameter putok = 5'b10000;  parameter putd = 5'b00000;  /\*实际的检测顺序，从左至右(按实际场景，应从左至右检测密码所以从m3开始)\*/  reg [4:0] check1 = (m3 == 3) ? put3:  (m3 == 2) ? put2:  (m3 == 1) ? put1:  put0;  reg [4:0] check2 = (m2 == 3) ? put3:  (m2 == 2) ? put2:  (m2 == 1) ? put1:  put0;  reg [4:0] check3 = (m1 == 3) ? put3:  (m1 == 2) ? put2:  (m1 == 1) ? put1:  put0;  reg [4:0] check4 = (m0 == 3) ? put3:  (m0 == 2) ? put2:  (m0 == 1) ? put1:  put0;  reg [4:0] checkok = putok;  /\*状态机状态\*/  parameter s0 = 4'b0000;  parameter s1 = 4'b0001;  parameter s2 = 4'b0010;  parameter s3 = 4'b0011;  parameter s4 = 4'b0100;  parameter s5 = 4'b0101;  parameter s6 = 4'b0110;  parameter s7 = 4'b0111;  parameter s8 = 4'b1000;  parameter s9 = 4'b1001;  parameter s10 = 4'b1010;  parameter s11 = 4'b1011;    always @(posedge clk)  if (reset) // 重置  begin  led <= led\_false;  state <= s0;  end  else  case (state) // mealy有限状态机  s0:  begin  if (bt\_in\_en == check1)  begin  led <= led\_begin;  state <= s1;  end  else if (bt\_in\_en == putd)  begin  led <= led\_begin;  state <= s0;  end  else  begin  led <= led\_begin;  state <= s10;  end  end  s1:  begin  if (bt\_in\_en == putd)  begin  led <= led\_begin;  state <= s2;  end  else if (bt\_in\_en == check1)  begin  led <= led\_begin;  state <= s1;  end  else  begin  led <= led\_begin;  state <= s10;  end  end  s2:  begin  if (bt\_in\_en == check2)  begin  led <= led\_begin;  state <= s3;  end  else if (bt\_in\_en == putd)  begin  led <= led\_begin;  state <= s2;  end  else  begin  led <= led\_begin;  state <= s10;  end  end  s3:  begin  if (bt\_in\_en == putd)  begin  led <= led\_begin;  state <= s4;  end  else if (bt\_in\_en == check2)  begin  led <= led\_begin;  state <= s3;  end  else  begin  led <= led\_begin;  state <= s10;  end  end  s4:  begin  if (bt\_in\_en == check3)  begin  led <= led\_begin;  state <= s5;  end  else if (bt\_in\_en == putd)  begin  led <= led\_begin;  state <= s4;  end  else  begin  led <= led\_begin;  state <= s10;  end  end  s5:  begin  if (bt\_in\_en == putd)  begin  led <= led\_begin;  state <= s6;  end  else if (bt\_in\_en == check3)  begin  led <= led\_begin;  state <= s5;  end  else  begin  led <= led\_begin;  state <= s10;  end  end  s6:  begin  if (bt\_in\_en == check4)  begin  led <= led\_begin;  state <= s7;  end  else if (bt\_in\_en == putd)  begin  led <= led\_begin;  state <= s6;  end  else  begin  led <= led\_begin;  state <= s10;  end  end  s7:  begin  if (bt\_in\_en == putd)  begin  led <= led\_begin;  state <= s8;  end  else if (bt\_in\_en == check4)  begin  led <= led\_begin;  state <= s7;  end  else  begin  led <= led\_begin;  state <= s10;  end  end  s8:  begin  if (bt\_in\_en == checkok)  begin  led <= led\_begin;  state <= s9;  end  else if (bt\_in\_en == putd)  begin  led <= led\_begin;  state <= s8;  end  else  begin  led <= led\_begin;  state <= s10;  end  end  s9:  begin  if (bt\_in\_en == putd)  begin  led <= led\_true;  state <= s9;  end  else if (bt\_in\_en == checkok)  begin  led <= led\_begin;  state <= s9;  end  else  begin  led <= led\_begin;  state <= s10;  end  end  s10:  begin  if (bt\_in\_en == putok)  begin  led <= led\_begin;  state <= s11;  end  else  begin  led <= led\_begin;  state <= s10;  end  end  s11:  begin  if (bt\_in\_en == putd)  begin  led <= led\_false;  state <= s11;  end  else if (bt\_in\_en == checkok)  begin  led <= led\_begin;  state <= s11;  end  else  begin  led <= led\_begin;  state <= s10;  end  end  endcase  assign state\_out = state;  assign out = {check3,bt\_in\_en};  endmodule  `timescale 1ns / 1ps  /\*初始密码写入模块\*/  module key\_write(write, clk, seg, sel, sw\_in, m0, m1, m2, m3);  output reg [7:0] seg;  output reg [3:0] sel; //七段数码管显示和选通  output wire [1:0] m0, m1, m2, m3;  input clk; //时钟信号  input write; //写数据  input [7:0] sw\_in; //初始密码拨键输入    reg [2:0] dispsel; //选通  reg [1:0] mem [3:0]; //存储write\_btword的数据  reg [1:0] dpdat; //tmp data  reg [19:0] count = 0;  always @(posedge clk or posedge write)  begin  if(write)  begin  mem[0] <= sw\_in[1:0];  mem[1] <= sw\_in[3:2];  mem[2] <= sw\_in[5:4];  mem[3] <= sw\_in[7:6];  end  else begin end  end  assign m0 = mem[0], m1 = mem[1], m2 = mem[2], m3 = mem[3];  /\*七段数码管显示模块\*/  always@(posedge clk) //分时复用  begin  count <= count + 1;  if(count == 50000)  begin  count <= 0;  dispsel <= dispsel + 1;  if(dispsel == 3)  dispsel <= 0;  end  end  always@(posedge clk)//分配选通  case(dispsel)  0: begin sel = 14; dpdat = mem[0]; end  1: begin sel = 13; dpdat = mem[1]; end  2: begin sel = 11; dpdat = mem[2]; end  3: begin sel = 7 ; dpdat = mem[3]; end  default: begin sel = 7; dpdat = 15; end  endcase    always@(posedge clk)//数码管显示  begin  seg[0] <= 1;  case(dpdat)  2'b00 : seg[7:1] <= 7'b0000001;  2'b01 : seg[7:1] <= 7'b1001111;  2'b10 : seg[7:1] <= 7'b0010010;  2'b11 : seg[7:1] <= 7'b0000110;  default: seg[7:1] <= 7'b1111111;  endcase  end  endmodule  仿真文件：  `timescale 1ns / 1ps  module sim\_lock;  reg write = 1, reset = 0, clk = 0;  reg [7:0] sw\_in = 8'b11100100; //3210  reg [4:0] bt\_in = 5'b00000;  wire [1:0] led;  wire [3:0] state\_out;    default\_password u(write,reset,clk,bt\_in,sw\_in,led,state\_out);    always #50 clk = ~clk;  initial  begin  #100 bt\_in = 5'b01000; //3  #100 bt\_in = 5'b00000;  #100 bt\_in = 5'b00100; //2  #100 bt\_in = 5'b00000;  #100 bt\_in = 5'b00010; //1  #100 bt\_in = 5'b00000;  #100 bt\_in = 5'b00001; //0  #100 bt\_in = 5'b00000;  #100 bt\_in = 5'b10000; //ok  end  endmodule | | | | | | |
| 五、实验过程中遇到的问题及解决情况  首先是按键防抖模块，对于教程中的代码一直不能防抖成功，所以只能在网上搜寻相关模块，如下图，测试多个模块后才能成功防抖。    如果在两个always语句中对同一个变量赋值，会出现multi\_driven net on pin的错误，需要修改或删去其中一个赋值。  出现以下错误，错误原因尚不明确，但在xdc文件中，加入  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets clk]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets reset\_IBUF]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets check\_IBUF]  便可修复。    在状态机设计时，会出现状态不动的情况，只需加入停滞状态即可修复，即每个检测密码状态之后加上一个没有按键输入的状态，在开发板中显示为按下和抬起按键为不同状态。  编码过程中还出现了一些语法错误，便不在此赘述了。 | | | | | | |
| 六、实验结果及分析和（或）源程序调试过程  为了使代码看起来简洁，逻辑更加清晰，采用多模块的编码方式。分为初始密码输入模块，通过拨键开关输入，输入完成后密码将显示在七段数码管上；密码检测模块，通过按键开关输入，其中输入完后需要按确认键才能显示开锁与否；按键防抖模块。  I/O接口：  Input ：sw[7:0]从左到右每两位输入一个密码数字  sw[15]write接口，高电平时能修改密码  sw[14]reset接口，高电平时重置状态  bt[3:0]输入密码接口，分别对应输入3，2，1，0  bt[4]确认密码接口，按下后将显示开锁与否  Output:led[15:14]开锁与否接口，10为成功，01为失败  led[13:10]状态输出接口，将输出当前状态  led[9:5]自定义check接口，用来调试程序，这里为check2  led[4:0]按键输出接口，按下按键对应led便会高电平  七段数码管，显示默认密码IMG_20211219_202315(1)  调试： 通过改变自定义接口的绑定，通过led灯的输出，来判断某个变量是否正确。  实验结果（演示视频将放在压缩包中）：  刚开始上板时，默认密码是0000  IMG_20211219_201249  通过write和sw，来修改密码为3210  IMG_20211219_201257  现来展示检测密码，每按下或抬起按键为不同状态，led能显示按下的  是哪一个按键和当前的状态IMG_20211219_201325  正确输入四位密码后，按下确认键，即密码解锁，led显示为10  IMG_20211219_201330  若错误密码，或密码长度不正确，按下确认键后都会失败，led为01（注：必须按下ok键后才能显示状态）  IMG_20211219_201403   1. 小组分工情况说明   陈鹏宇：自主完成实验，完成实验报告  帕肉合帕尔哈提：自主完成实验 | | | | | | |