串行密码锁

计01　容逸朗　2020010869

1　实验内容

用状态机设计一个 4 位十六进制串行电子密码锁，具体功能如下所示：

- 设置密码。用户可以串行设置 4 位十六进制密码。
- 验证密码。用户串行输入密码，如果密码符合则点亮开锁灯，否则点亮错误灯。
- 密码预置。为管理员创建万用密码以备管理。
- 系统报警。开锁 3 次失败后点亮警灯，并锁定密码锁，只有输入管理员密码才可开锁并解除报警。

2　实验原理

2.1　状态机设计

状态机的状态如下表所示：

状态	用途	用户操作	条件1	次态	条件2	次态
0	初始状态	点击 rst	-	0		
0	设置密码的第一位	点击 clk	MODE: 00, 输入失败次败为 0	1		
0	检验密码的第一位	点击 clk	MODE: 01, 密码正确	4	密码错误	0
1	设置密码的第二位	点击 clk	设置成功	2		
2	设置密码的第三位	点击 clk	设置成功	3		
3	设置密码的第四位	点击 clk	设置成功	0		
4	检验密码的第二位	点击 clk	密码正确	5	密码错误	0
5	检验密码的第三位	点击 clk	密码正确	6	密码错误	0
6	检验密码的第四位	点击 clk	密码正确	0	密码错误	0

2.2　实验代码

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

```

```
entity lock is
  port(
    rst, clk: in std_logic;
    code: in std_logic_vector(3 downto 0);
    mode: in std_logic_vector(1 downto 0);
    unlock: out std_logic;
    alarm, err: buffer std_logic;
    state_id: buffer std_logic_vector(3 downto 0)
  );
  subtype pw1 is integer range 0 to 15;
  type pw4 is array(3 downto 0) of pw1;
  subtype states is integer range 0 to 6;
end lock;

architecture bhv of lock is
  signal state: states := 0;
  signal cnt: integer := 0; -- 错误次数
  signal pwd: pw4 := (0, 0, 0, 0);
  signal super_pwd: pw4 := (1, 1, 1, 2); -- 管理员密码
  -- 需要注意密码输入顺序是相反的，即上面的代码代表了 2111，而不是 1112
  signal tar_pwd: integer;
  signal c_user, c_admin: std_logic;
begin
  tar_pwd <= conv_integer(code);
  process(rst, clk)
  begin
    if (rst = '1') then
      state <= 0;
      unlock <= '0';
      err <= '0';
    elsif (clk'event and clk = '1') then
      if (mode = "00" and alarm = '0' and cnt = 0) then -- 若为设置
        case state is
          when 0 | 1 | 2 => pwd(state) <= tar_pwd; state <= state + 1;
          when 3 => pwd(state) <= tar_pwd; unlock <= '1'; state <= 0;
          when others => null;
        end case;
      elsif (mode = "01") then -- 若为验证
        case state is
          when 0 =>
            if (alarm = '0' and tar_pwd = pwd(0)) then
              -- 与用户设置的密码的首位匹配
              if (tar_pwd /= super_pwd(0)) then -- 不与管理员密码匹配
                c_admin <= '0';
```

```
              end if;
              c_user <= '1';
              state <= 4;
            elsif (tar_pwd = super_pwd(0)) then
              -- 与管理员密码的首位匹配
              if (tar_pwd /= pwd(0)) then -- 不与用户设置的密码匹配
                c_user <= '0';
              end if;
              c_admin <= '1';
              state <= 4;
            else -- 完全没有匹配任何一个密码
              state <= 0;
              err <= '1';
              if (cnt > 1) then -- 超过错误次数上限
                alarm <= '1';
              else
                cnt <= cnt + 1;
              end if;
            end if;
          when 4 | 5 | 6 =>
            if (alarm = '0' and c_user = '1' and tar_pwd = pwd(state -
3)) then -- 与用户设置的密码匹配
              if (tar_pwd /= super_pwd(state - 3)) then
                c_admin <= '0';
              end if;
              if (state = 6) then -- 与用户设置的密码完全匹配
                unlock <= '1';
                alarm <= '0';
                cnt <= 0;
                state <= 0;
              else
                state <= state + 1;
              end if;
            elsif (c_admin = '1' and tar_pwd = super_pwd(state - 3))
then
              -- 与管理员密码匹配
              if (tar_pwd /= pwd(state - 3)) then
                c_user <= '0';
              end if;
              if (state = 6) then -- 与管理员密码完全匹配
                unlock <= '1';
                alarm <= '0';
                cnt <= 0;
                state <= 0;
```

```
93                else
94                  state <= state + 1;
95                end if;
96              else -- 完全没有匹配任何一个密码
97                err <= '1';
98                state <= 0;
99                if (cnt > 1) then -- 超过错误次数上限
100                 alarm <= '1';
101               else
102                 cnt <= cnt + 1;
103               end if;
104             end if;
105           when others => null;
106         end case;
107       end if;
108     end if;
109   end process;
110
111   process(state) -- 输出当前状态
112   begin
113     case state is
114       when 0 => state_id <= "0000";
115       when 1 => state_id <= "0001";
116       when 2 => state_id <= "0010";
117       when 3 => state_id <= "0011";
118       when 4 => state_id <= "0100";
119       when 5 => state_id <= "0101";
120       when 6 => state_id <= "0110";
121       when others => null;
122     end case;
123   end process;
124
125 end bhv;
126
```

工作原理：上面的代码是根据 2.1 节的状态机所设计的，这里利用了 c_user 和 c_admin 两个中间变量表示当前检测对应的密码是否正确。具体原理可以参考代码中的注释。

3 电路功能测试

3.1 实际操作

3.1.1 实验用具

本次使验使用了数字逻辑实验平台中的**带译码数码管**，一个**可编程模块**，两个**开关**和一个**触发器**。

3.1.2 实验步骤

根据书上的端口设计接线。然后按照下述步骤测试：

1. 调整模式为 **00** 并点击 **rst**；
2. 输入密码 **4**，点击 **clk** ，重复此操作 **4** 次；（即设置密码为 **4444**）
3. 此时开锁灯亮；
4. 调整模式为 **01** 并点击 **rst**；
5. 输入密码 **4**，点击 **clk** ，重复此操作 **4** 次；
6. 此时开锁灯亮；
7. 点击 **rst**，输入密码 **8**，点击 **clk** （此时可看到错误灯亮），重复此操作 **3** 次；
8. 此时警报灯亮；
9. 点击 **rst**，输入密码 **4**，点击 **clk** ，此时错误灯亮，说明密码锁被锁定；
10. 点击 **rst**，输入密码 **2**，点击 **clk** ；
11. 点击 **rst**，输入密码 **1**，点击 **clk** ，重复此操作 **3** 次；（管理员密码是 **2111**）
12. 此时警报灯灭，开锁灯亮。（密码锁解除锁定）

至此，代码和接线均无误。

3.2 仿真实验

3.2.1 实验步骤

与 3.1.2 节类似，但测试的密码不同。

3.2.2 仿真结果

随意选择可以得到如下结果：

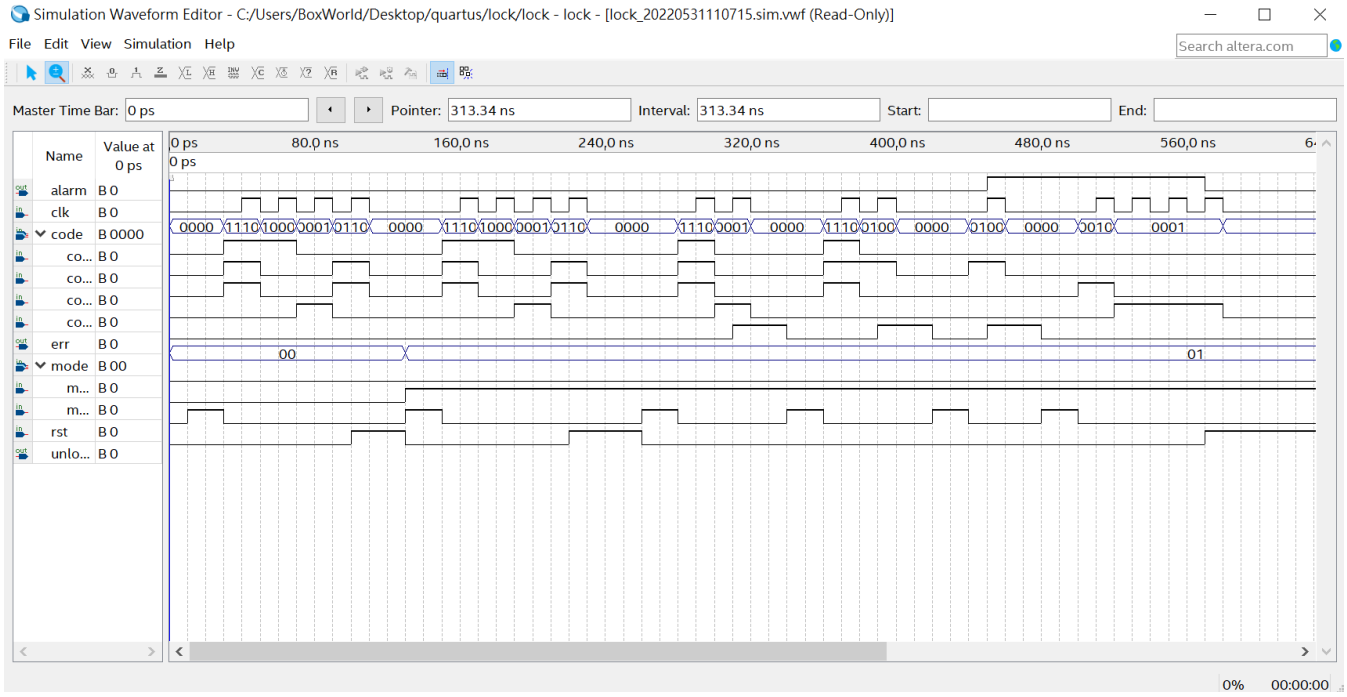

图1　密码锁仿真结果

可以看见，密码锁运作正常。

4　遇到的问题与解决方法

在实验过程中，曾经出现过点击 3 次 clk 便打开密码锁的情况，为此增加了一个带译码数码管来显示当前状态编号，这时发现问题是因由于按键过于灵敏而导致的。