串行密码锁实验报告

傅舟涛 2017010682

一、实验任务

1、用状态机设计一个串行密码锁，包括如下内容：

功能名称	功能描述
密码预制	创建管理员万能密码（4位十六进制）以备管理，它在任何时候都能开锁
设置密码	用户可以设置4位十六进制密码，验证密码错误后锁定此功能直到再次解锁
验证密码	用户串行输入密码，密码符合用户密码或管理员密码（报警灯亮时 只能用管理员密码）则亮开锁灯，均不符合则点亮错误灯
系统报警	连续开锁三次失败后点亮报警灯，此时锁定设置密码与 用户密码验证功能，只有输入管理员密码才可以解锁

密码预制 创建管理员万能密码（4位十六进制）以备管理，它在任何时候都能开锁； 设置密码 用户可以设置4位十六进制密码； 验证密码 用户串行输入密码，密码符合用户密码或管理员密码则亮开锁灯，均不符合则点亮错误灯； 系统报警 连续开锁三次失败后点亮报警灯，只有输入管理员密码才可以解锁。

2、输入包括code[3...0], clk, rst, mode[1...0]，输出包括unlock, alarm, error，含义为：

接口名称	输入/输出	接口功能
code[3...0]	输入	设置密码或验证密码时用户输入的一位十六进制密码
clk	输入	为确认按钮，在对应模式下按下一次clk设置或验证一位密码，上边沿触发
rst	输入	为复位和初始化按钮，设置或验证密码开始前需按一次， 设置或验证密码时也可按此按钮回到初始状态
mode[1..0]	输入	"00"为设置密码模式，"01"为验证密码模式，其余不动
unlock	输出	开锁灯，见“验证密码”功能
alarm	输出	报警灯，见“系统报警”功能
error	输出	错误灯，见“验证密码”功能

二、状态机的设定

设置两个中间信号us和ad，为一个std_logic，分别表示用户密码检测可用和管理员密码检测可用，只有us='1'时才会检测用户密码，只有ad='1'时才会检测管理员密码。

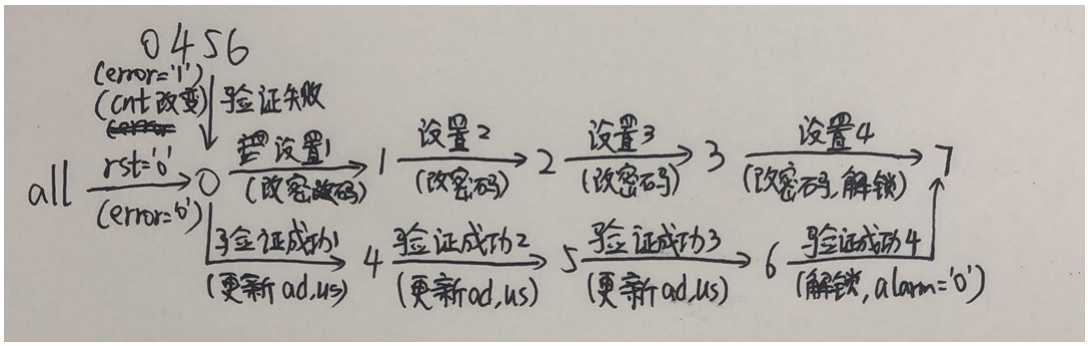
设置一个中间信号cnt，为0-2的整数，是错误次数计数器，当cnt=2时再次发生错误，会使警报响起。

us='1'的条件： 1、alarm='0' 2、之前所有位输入密码均与用户密码相同

ad='1'的条件： 1、之前所有位输入密码均与管理员密码相同

设置8个状态0-7，含义如下： 状态0：等待状态，同时也是设置和验证第一位密码的起始状态，mode="00"且设置可用时，设置第一位密码，切换至状态1；mode="01"时，验证第一位密码，如果验证成功则切换至状态4，如果验证失败则进行失败处理。 状态1-3：设置状态，mode="00"且设置可用时，设置第二至四位密码，并切换至下一个状态，如果当前是状态3则切换至状态7，否则切换至当前状态数+1的状态。. 状态4-6：验证状态，mode="01"时，验证第二至四位密码，如果验证成功则切换至下一个状态（当前状态数+1），如果验证失败则进行失败处理。 状态7：解锁状态，切换至状态7时，使报警灯熄灭，开锁灯亮。 重置处理：当rst='0'时，无论当前状态如何，切换至状态0，且使错误灯和开锁灯熄灭。 失败处理：当验证失败时，切换至状态0，使error='1'，如果cnt=2则使警报灯亮，否则使cnt加1.

状态图如下，箭头上未用括号括起来的部分表示更改状态条件，括号括起来的部分表示需要进行的处理。



三、代码及简要语法说明

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity lock is
    port(
        code: in std_logic_vector(3 downto 0);
        mode: in std_logic_vector(1 downto 0);
        clk, rst: in std_logic;
        unlock: out std_logic;
        alarm, err: buffer std_logic
    );
    subtype pwd1 is integer range 0 to 15;
    type pwd4 is array (3 downto 0) of pwd1;
    --password is a 4x4 2D vector
    subtype states is integer range 0 to 7;
    --0: check/set bit 0; 1-3: set bit 1-3;
    --4-6: check bit 4-6; 7: unlock.
end lock;

architecture arc of lock is
    constant admin_pwd: pwd4 := (8,8,8,8);
    --password of admin can be changed here.
    signal pwd: pwd4 := admin_pwd;
    signal pwd_in: pwd1;
```

```
    signal ad, us: std_logic;
    --to control the checking process
    signal state: states;
    signal cnt: integer range 0 to 2:= 0;
    --failed times
begin
    pwd_in<=conv_integer(code);
    process(clk, rst)
    begin
        if (rst = '0') then
            unlock <= '0';
            err <= '0';
            state <= 0;
        elsif (clk'event and clk = '1') then
            if (mode = "00" and alarm = '0' and cnt = 0) then
            --set passwd, only when not checking and haven't been wrong
            --(which means it's unlock or just turn from unlock to lock)
                case state is
                    when 0|1|2 => pwd(state) <= pwd_in; state <= state + 1;
                    when 3 => pwd(3) <= pwd_in; state <= 7; unlock <= '1';
                    when others => NULL;
                end case;
            elsif (mode = "01") then --check password
                case state is
                    when 0 =>
                        if ((pwd_in = pwd(0) and alarm = '0')
                            or pwd_in = admin_pwd(0)) then
                            if (pwd_in = pwd(0) and alarm = '0') then
                                us <= '1';
                            else
                                us <= '0';
                            end if;
                            if (pwd_in = admin_pwd(0)) then
                                ad <= '1';
                            else
                                ad <= '0';
                            end if;
                            state <= 4;
                            err <= '0';
                        else
                            err <= '1';
                            if (cnt > 1) then
                                alarm <= '1';
                                cnt <= 0;
                            else
                                cnt <= cnt + 1;
                            end if;
                        end if;
                    when 4|5|6 =>
                        if (((pwd_in = pwd(state-3)) and (us = '1')) or
                            ((pwd_in = admin_pwd(state-3)) and (ad = '1'))) then
                            if (state = 6) then
                                unlock <= '1';
```

```
                                alarm <= '0';
                                cnt <= 0;
                            end if;
                            state <= state + 1;
                            if (pwd_in /= pwd(state-3)) then
                                us <= '0';
                            end if;
                            if (pwd_in /= admin_pwd(state-3)) then
                                ad <= '0';
                            end if;
                        else
                            err <= '1';
                            state <= 0;
                            if (cnt > 1) then
                                alarm <= '1';
                                cnt <= 0;
                            else
                                cnt <= cnt + 1;
                            end if;
                        end if;
                    when others => NULL;
                end case;
            end if;
        end if;
    end process;
end arc;
```

状态机的设置、逻辑、输入输出和中间信号的含义已在前文大致讲述完毕，下面仅讲述前面没有提到的中间信号：

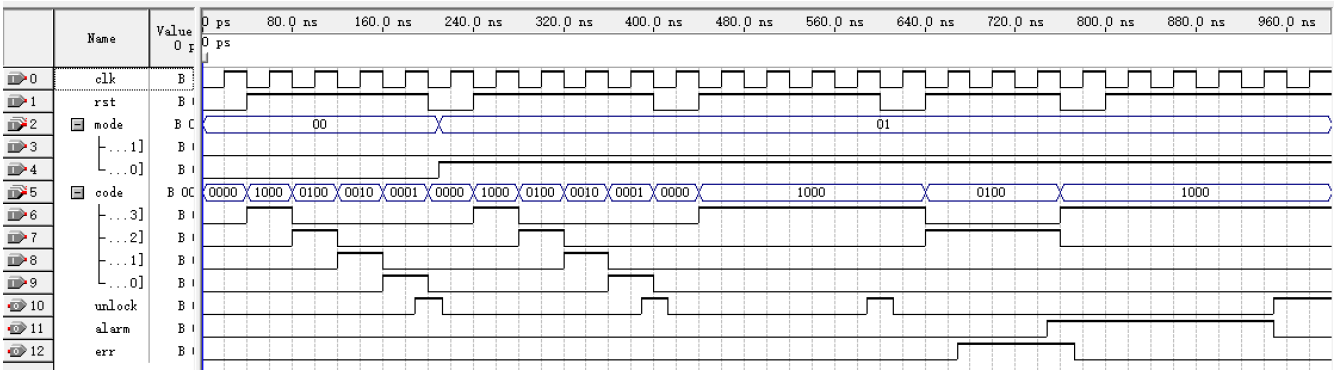
```
    subtype pwd1 is integer range 0 to 15;
    type pwd4 is array (3 downto 0) of pwd1;
    --password is a 4x4 2D vector
    subtype states is integer range 0 to 7;
```

这是一位密码、四位密码和状态机状态的类型定义，其中一位密码使用的是0-15的整数，也即一个十六进制数，用integer而不是std_logic_vector的原因是方便文件中管理员密码的设置，不用写成("1000", "1000", "1000", "1000")，而直接写成(8, 8, 8, 8)即可。状态机定义成0-7的整数而非枚举类的原因是整数方便代码复用（即状态123和状态456的很多逻辑是相同的，可以放在一起）。

```
    constant admin_pwd: pwd4 := (8,8,8,8);
    --password of admin can be changed here.
    signal pwd: pwd4 := admin_pwd;
    signal pwd_in: pwd1;
    signal state: states;
```

这是管理员密码、用户密码、输入的一位密码和状态机状态的定义。使用when-case进行状态转移，使用conv_integer将输入的四位code转化为一个0-15的整数。

四、仿真结果



包括了设置密码，用用户密码解锁，用管理员密码解锁，三次错误后报警，用管理员密码解除报警并解锁的操作，如上图所示。

先设置密码为8421，解锁。清零验证密码8421，解锁。清零验证管理员密码8888，解锁。接着输入4连续失败3次，报警灯亮，每次失败时错误灯亮。输入管理员密码，报警灯灭并解锁。

五、实验小结

1、学习了状态机的用法

其实我当时最先学习的是枚举类的状态机，在实现过程中发现有大量代码重复，因此又学习了用整数定义状态机的方法，增加了代码复用性。本次实验让我会使用状态机来设计电路。

2、学习了自定义类型设置、数据类型转换

学习了type与suptype用法的区别，知道了自定义类型的正确用法。学会了conv_integer等多个数据类型转换方法。懂得了subtype和type定义范围整数的区别。

```
type a is range 0 to 15;
subtype b is integer range 0 to 15;
```

则a类型的信号是一个自定义类型，b类型的信号是integer的子类型，也是自定义类型。因此b类型的信号可以用conv_integer()赋值，也可以用conv_std_logic_vector()将其转换为std_logic_vector，但a类型的信号均不可以。

3、学习了package的使用方法

本来想使用元件例化的方式实现，那样的话用package来定义管理员密码、密码和状态机类型是很有必要的。但是后来发现state需要在多个地方进行修改，用元件例化的方法反而降低了可读性，因此在最终的代码里没有使用。