```
-- Philip Krogh-Pedersen
-- 10/11 - 2014

-- Quartus II VHDL Template
-- Four-State Moore State Machine
-- A Moore machine's outputs are dependent only on the current state.
-- The output is written only when the state changes. (State
-- transitions are synchronous.)

library ieee;
use ieee.std_logic_1164.all;

entity codeLock is
    port(
        clk : in std_logic;
        code : in std_logic_vector(3 downto 0);
        reset : in std_logic;
        enter : in std_logic;
        lockGreenLED, lockSignal : out std_logic;
        lockRedLED : out std_logic_vector(17 downto 0));

        end entity;

architecture rtl of codeLock is

    -- Build an enumerated type for the state machine
    type state is (Idle, Code1Accepted, Code2Accepted, Unlocked, WrongCode,
PermanentlyLocked);

    -- Register to hold the current state
    signal present_state : state;

    -- Signals to hold codes
    signal code1, code2, code3 : std_logic_vector(3 downto 0);

    -- signal for enter event detector
    signal enter_event, enter_last : std_logic;

begin

    code1 <= "0001";
    code2 <= "0010";
    code3 <= "0011";

    -- Logic to advance to the next state
    process (clk, reset)
    -- variable to count number of failed trys
        variable err_cnt : integer := 0;
    begin
        if reset = '0' then
            present_state <= Idle;
        elsif (rising_edge(clk)) then
            case present_state is
                when Idle=>
                    if enter_event = '1' then
                        if code = code1 then
                            present_state <= Code1Accepted;
                        else
```

```
                        present_state <= WrongCode;
                    end if;
                else
                    present_state <= Idle;
                end if;
            when Code1Accepted=>
                if enter_event = '1' then
                    if code = code2 then
                        present_state <= code2Accepted;
                    else
                        present_state <= WrongCode;
                    end if;
                else
                    present_state <= Code1Accepted;
                end if;
            when Code2Accepted=>
                if enter_event = '1' then
                    if code = code3 then
                        present_state <= Unlocked;
                    else
                        present_state <= WrongCode;
                    end if;
                else
                    present_state <= Code2Accepted;
                end if;
            when Unlocked =>
                if enter_event = '1' then
                    present_state <= Idle;
                else
                    present_state <= Unlocked;
                end if;
            when WrongCode =>
                err_cnt := err_cnt + 1;
                if err_cnt >= 3 then
                    present_state <= PermanentlyLocked;
                else
                    present_state <= Idle;
                end if;
            when PermanentlyLocked =>
                    present_state <= permanentlyLocked;
        end case;
    end if;
end process;

-- Output depends solely on the current state
process (present_state)
begin
    case present_state is
        when Idle =>
            lockGreenLED <= '0';
            lockSignal <= '1';
            lockRedLED <= "0000000000000111";
        when Code1Accepted =>
            lockGreenLED <= '0';
            lockSignal <= '1';
            lockRedLED <= "0000000000000110";
        when Code2Accepted =>
            lockGreenLED <= '0';
            lockSignal <= '1';
```

```
                lockRedLED <= "0000000000000000100";
            when Unlocked =>
                lockGreenLED <= '1';
                lockSignal <= '0';
                lockRedLED <= "0000000000000000000";
            when WrongCode =>
                lockGreenLED <= '0';
                lockSignal <= '1';
            when PermanentlyLocked =>
                lockGreenLED <= '0';
                lockSignal <= '1';
                lockRedLED <= "1111111111111111";
        end case;
    end process;

    -- "enter" event detector
    -- output, enter_event, goes high one clock-cycle
    process (clk, enter) is
    begin -- process
        if enter = '1' then
            enter_event <= '0';
            enter_last <= '1';
        elsif clk'event and clk = '1' then
            enter_last <= enter;
            enter_event <= '0';
        if enter_last = '1' and enter = '0' then
            enter_event <= '1';
        end if;
        end if;
    end process;

end rtl;
```