EGEE 448: Digital Systems Design with FPGA Fall 2020

Homework/Lab Project 3: Design a Keyless Entry System using FPGA Board

Instructor: Dr. Jidong Huang

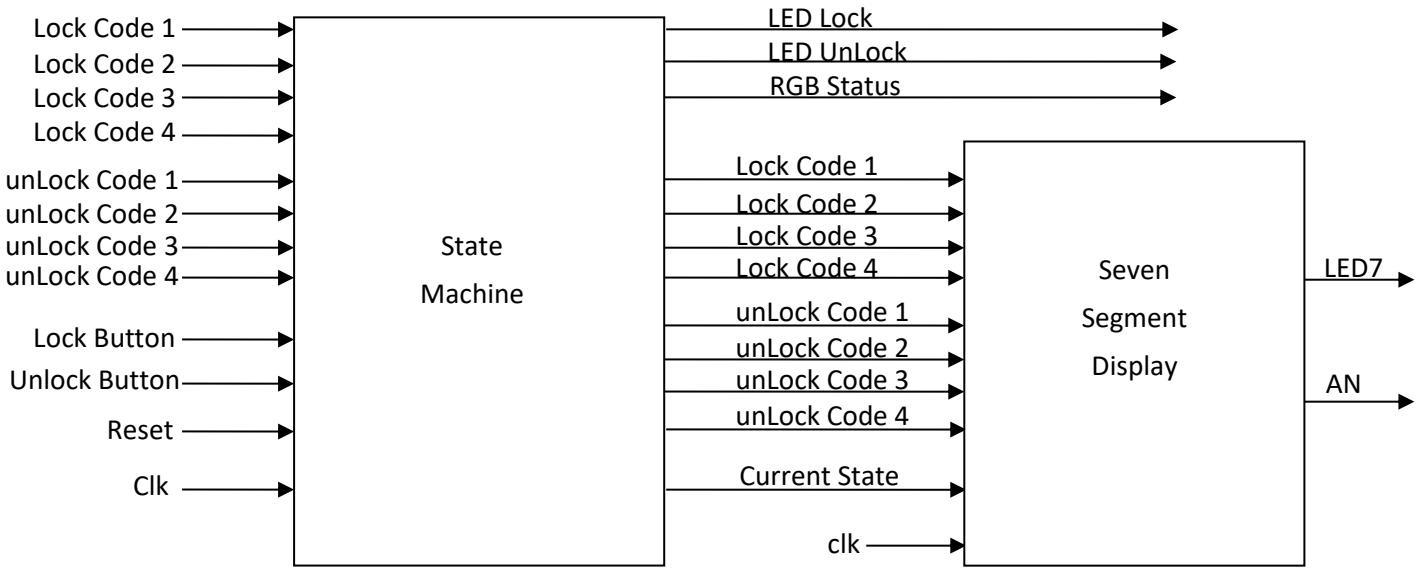
Student: David Castro

PART A) Digital Design Problem

Question 1) Design a Keyless Entry System using VHDL

Design Project:

Based on the above project design requirements, how will you design this keyless entry system? **Please draw below a block diagram for the subcomponents needed for this system; and explain briefly the functionality of each block/subsystem.**



The State machine module does most of the work. It will take the user inputs and decide what state to go to based off the conditions it meets. Each state has its own functionality so the outputs will differ from each. Based off what state it currently is in the LED's and RGB light will change from the state machine module. It will then output the current state to the seven-segment display module. The seven-segment display module is meant to control the seven-segments and decide what will be outputted based off the state it received. The "clk" acts as a timer where it controls the rate things are outputted (e.g. making the seven-segment display at rate of 60 Hz).

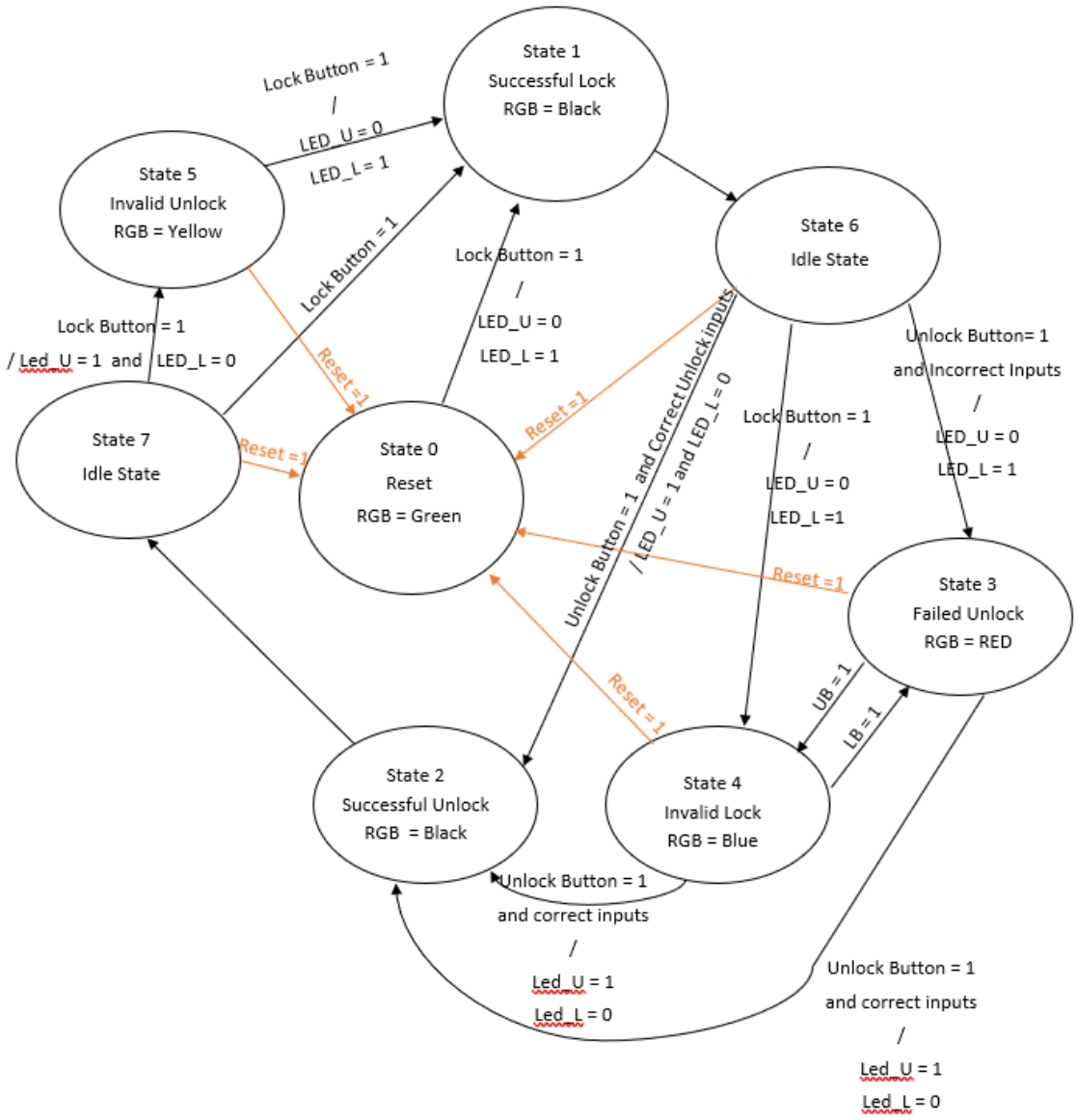
Additionally, what will be the state diagram describing the transition of states among the above-mentioned six states and their corresponding input/outputs?

Every state is there as mentioned in the instructions, but there are two extra states added (state 6 and 7). These states do not do anything but act as idle states. They are transitioned to automatically. This was done to make the code operate correctly.

For the State Diagram below it does not include the outputs for the seven-segment display. This was done because they could not be fit in. They are based off user inputs but certain states do have special characteristics like outputting 'C' for the lock code 7Segment display portion in state 1, 3, and 4 (this follows the example pictures in the instructions document). Another one would be outputting 'A' for the unlock code 7segment display portion in state 2 and 5 (follows the example pictures in the instruction document.)

*Note – LB = lock button UB = Unlock Button

"Reset = 1" makes Unlock LED = 1 and Lock LED = 0 (not include below for space)



Per your design, what is your VHDL entity code for this digital keyless system? Please insert below the source code for the VHDL entity file(s) and describe the functionality of each VHDL entity file included in your project.

Structural File:

This puts everything together. It connects the two modules together. The signals in this file are simply meant to connect the two entity files together. There isn't anything else to it, but to act as the whole structure as seen by the block diagram above.

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--structural file of Keyless Entry system

entity structure_keyless_entry is
    Port(
        lock_code_1: in std_logic_vector(1 downto 0);
        lock_code_2: in std_logic_vector(1 downto 0);
        lock_code_3: in std_logic_vector(1 downto 0);
        lock_code_4: in std_logic_vector(1 downto 0);
        unlock_code_1: in std_logic_vector(1 downto 0);
        unlock_code_2: in std_logic_vector(1 downto 0);
        unlock_code_3: in std_logic_vector(1 downto 0);
        unlock_code_4: in std_logic_vector(1 downto 0);
        unlock_button: in std_logic;
        lock_button: in std_logic;
        reset: in std_logic;
        clk: in std_logic;
        AN: out std_logic_vector(7 downto 0);
        led_locked: out std_logic;      --indicates the system is locked
        led_unlocked: out std_logic;    --indicates the system is unlocked
        led_status: out std_logic_vector(2 downto 0);
        LED7: out std_logic_vector(7 downto 0) –output for the 7SEG
    );
end structure_keyless_entry;

architecture Behavioral of structure_keyless_entry is
  --signals for connecting the components
    --signal for connecting state logic
    signal signal_state: Integer range 0 to 7;
```

```
--will connect the stored numbers to sevseg display
signal signal_lock_code_storage_1: std_logic_vector(1 downto 0);
signal signal_lock_code_storage_2: std_logic_vector(1 downto 0);
signal signal_lock_code_storage_3: std_logic_vector(1 downto 0);
signal signal_lock_code_storage_4: std_logic_vector(1 downto 0);
signal signal_unlock_code_storage_1: std_logic_vector(1 downto 0);
signal signal_unlock_code_storage_2: std_logic_vector(1 downto 0);
signal signal_unlock_code_storage_3: std_logic_vector(1 downto 0);
signal signal_unlock_code_storage_4: std_logic_vector(1 downto 0);

component keyless_entry
  Port(
   clk: in std_logic;
   lock_code_1: in std_logic_vector(1 downto 0);
   lock_code_2: in std_logic_vector(1 downto 0);
   lock_code_3: in std_logic_vector(1 downto 0);
   lock_code_4: in std_logic_vector(1 downto 0);
   unlock_code_1: in std_logic_vector(1 downto 0);
   unlock_code_2: in std_logic_vector(1 downto 0);
   unlock_code_3: in std_logic_vector(1 downto 0);
   unlock_code_4: in std_logic_vector(1 downto 0);
   unlock_button: in std_logic;
   lock_button: in std_logic;
   reset: in std_logic;
   led_locked: out std_logic;      --indicates the system is locked
   led_unlocked: out std_logic;    --indicates the system is unlocked
   led_status: out std_logic_vector(2 downto 0);
   lock_code_storage_1: inout std_logic_vector(1 downto 0);
   lock_code_storage_2: inout std_logic_vector(1 downto 0);
   lock_code_storage_3: inout std_logic_vector(1 downto 0);
   lock_code_storage_4: inout std_logic_vector(1 downto 0);
   unlock_code_storage_1: out std_logic_vector(1 downto 0);
   unlock_code_storage_2: out std_logic_vector(1 downto 0);
   unlock_code_storage_3: out std_logic_vector(1 downto 0);
   unlock_code_storage_4: out std_logic_vector(1 downto 0);
   state_out: out Integer range 0 to 7
   );
end Component;
```

```
    Component SevSeg
      Port(
       lock_code_storage_1: in std_logic_vector(1 downto 0);
       lock_code_storage_2: in std_logic_vector(1 downto 0);
       lock_code_storage_3: in std_logic_vector(1 downto 0);
       lock_code_storage_4: in std_logic_vector(1 downto 0);
       unlock_code_storage_1: in std_logic_vector(1 downto 0);
       unlock_code_storage_2: in std_logic_vector(1 downto 0);
       unlock_code_storage_3: in std_logic_vector(1 downto 0);
       unlock_code_storage_4: in std_logic_vector(1 downto 0);
       AN: out std_logic_vector(7 downto 0);
       LED7: out std_logic_vector(7 downto 0);
       clk: in std_logic;
       state_input: in Integer range 0 to 7
       );
    end Component;

begin
    States: keyless_entry Port Map
    (
    clk => clk,
    lock_code_1 => lock_code_1,
    lock_code_2 => lock_code_2,
    lock_code_3 => lock_code_3,
    lock_code_4 => lock_code_4,
    unlock_code_1 => unlock_code_1,
    unlock_code_2 => unlock_code_2,
    unlock_code_3 => unlock_code_3,
    unlock_code_4 => unlock_code_4,
    unlock_button => unlock_button,
    lock_button => lock_button,
    reset => reset,
    led_locked => led_locked,
    led_unlocked => led_unlocked,
    led_status => led_status,
    lock_code_storage_1 => signal_lock_code_storage_1,
    lock_code_storage_2 => signal_lock_code_storage_2,
    lock_code_storage_3 => signal_lock_code_storage_3,
    lock_code_storage_4 => signal_lock_code_storage_4,
    unlock_code_storage_1 => signal_unlock_code_storage_1,
```

```
unlock_code_storage_2 => signal_unlock_code_storage_2,
unlock_code_storage_3 => signal_unlock_code_storage_3,
unlock_code_storage_4 => signal_unlock_code_storage_4,
state_out => signal_state
);

SevenSegment: SevSeg Port Map
(
lock_code_storage_1 => signal_lock_code_storage_1,
lock_code_storage_2 => signal_lock_code_storage_2,
lock_code_storage_3 => signal_lock_code_storage_3,
lock_code_storage_4 => signal_lock_code_storage_4,
unlock_code_storage_1 => signal_unlock_code_storage_1,
unlock_code_storage_2 => signal_unlock_code_storage_2,
unlock_code_storage_3 => signal_unlock_code_storage_3,
unlock_code_storage_4 => signal_unlock_code_storage_4,
LED7 => LED7,
AN => AN,
clk => clk,
state_input => signal_state
);

end Behavioral;
```

States Entity:

This entity decides what state entity decides what state the system will be in. It takes in the user inputs and based off the conditions will decide what next state to go to. The reason for there being a clock inside this entity is to act as a debounce mechanism. The clock will allow for the program to count to a certain amount of time (0.4 seconds in this case) and not allow for any other input to be accepted. After it has decided what state to go to it will change the status LED (RGB) to what was given in the instructions. It will also turn on or off the unlocked and locked status LEDs depending on what the current state it is. The last thing it does is store the code for unlocking.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity keyless_entry is
```

```
    Port(
        clk: in std_logic;
        lock_code_1: in std_logic_vector(1 downto 0);
        lock_code_2: in std_logic_vector(1 downto 0);
        lock_code_3: in std_logic_vector(1 downto 0);
        lock_code_4: in std_logic_vector(1 downto 0);
        unlock_code_1: in std_logic_vector(1 downto 0);
        unlock_code_2: in std_logic_vector(1 downto 0);
        unlock_code_3: in std_logic_vector(1 downto 0);
        unlock_code_4: in std_logic_vector(1 downto 0);
        unlock_button: in std_logic;
        lock_button: in std_logic;
        reset: in std_logic;
        led_locked: out std_logic;      --indicates the system is locked
        led_unlocked: out std_logic;    --indicates the system is unlocked
        led_status: out std_logic_vector(2 downto 0);
        lock_code_storage_1: inout std_logic_vector(1 downto 0);
        lock_code_storage_2: inout std_logic_vector(1 downto 0);
        lock_code_storage_3: inout std_logic_vector(1 downto 0);
        lock_code_storage_4: inout std_logic_vector(1 downto 0);
        unlock_code_storage_1: out std_logic_vector(1 downto 0);
        unlock_code_storage_2: out std_logic_vector(1 downto 0);
        unlock_code_storage_3: out std_logic_vector(1 downto 0);
        unlock_code_storage_4: out std_logic_vector(1 downto 0);
        state_out: out Integer range 0 to 7
    );
end keyless_entry;

architecture Behavioral of keyless_entry is
    signal state: Integer range 0 to 7;    --signal for state
    signal next_state: Integer range 0 to 7;
    signal start: std_logic;         --signal for debounce (triggers when there is input)
    signal count: Integer := 0; --counter for slwclk
    signal slowclock: std_logic;

    --signals for acting as memory
        signal storage_lock_1: std_logic_vector(1 downto 0);
        signal storage_lock_2: std_logic_vector(1 downto 0);
        signal storage_lock_3: std_logic_vector(1 downto 0);
        signal storage_lock_4: std_logic_vector(1 downto 0);
```

```
        signal storage_unlock_1: std_logic_vector(1 downto 0);
        signal storage_unlock_2: std_logic_vector(1 downto 0);
        signal storage_unlock_3: std_logic_vector(1 downto 0);
        signal storage_unlock_4: std_logic_vector(1 downto 0);

begin



    -- change 50000000 to 1 when simulating
    --process that will act as buff so there's delay between inputs(minimize bounce)
    debounce: Process(clk)
      begin
            if((rising_edge(clk)) and (start = '1')) then
                if(count = 50000000) then -- 0.5 sec delay based off 100MHZ clk before
next input can be taken
                    slowclock <= '1';
                    count <= 0; --reset counter
                else
                    count <= count + 1;
                end if;
            end if;
            if(slowclock = '1') then
               slowclock <= '0';
            end if;
    end Process debounce;

    --Process that will tranist the state to the nextstate
    transit: Process(slowclock)
      begin
        if(rising_edge(slowclock)) then
         state <= next_state;
        end if;
    end Process ;

    --dont want this to trigger off clock because then it will reset "start" back to 0 too fast
    starter: Process(lock_button, unlock_button, reset, state, clk)
      begin
         if(lock_button = '1' or unlock_button = '1' or reset = '1' or state = 1 or state = 2)
then
```

```
            start <= '1';

        elsif(state = next_state) then
            start <= '0';
        end if;
    end Process starter;

   --process that holds conditions for entering states
   Process(lock_button, unlock_button, reset, clk)
    begin
      if(state = next_state) then -- to act as a debounce (already in the next state)
        --condition for entering state 0 (reset state = unlocked state) - green
        if(reset = '1') then
            next_state <= 0;
        end if;

        --condition for entering state 1 (locked state) - no led
        if(lock_button = '1' and (state = 0 or state = 7 or state = 5)) then
            next_state <= 1;
        end if;

        --condition for entering state 2 (unlocked state) -- no led
        if(unlock_button = '1' and (state = 6 or state = 3 or state = 4)  and    --making
sure codes match
              unlock_code_1 = lock_code_storage_1 and
              unlock_code_2 = lock_code_storage_2 and
              unlock_code_3 = lock_code_storage_3 and
              unlock_code_4 = lock_code_storage_4) then
                next_state <= 2;
        end if;

        --condition for entering state 3 (failed unlock - remain locked) - red
        if((unlock_button = '1' and (state = 6 or state = 4)) and   --checking if codes dont
match
              (unlock_code_1 /= lock_code_storage_1 or
              unlock_code_2 /= lock_code_storage_2 or
              unlock_code_3 /= lock_code_storage_3 or
              unlock_code_4 /= lock_code_storage_4)) then
                next_state <= 3;
       end if;
```

```
        --condition for entering state 4 (invalid lock - remain locked) - blue
        if(lock_button = '1' and (state = 6 or state = 3)) then
              next_state <= 4;
        end if;

        --condition for entering state 5 (invalid unlock - remain unlocked) - yellow
        if(unlock_button = '1' and (state = 7 or state = 0))  then
              next_state <= 5;
        end if;

        if(state = 1) then
           next_state <= 6;
        end if;

        if(state = 2) then
           next_state <= 7;
        end if;
     end if;
end Process;



--process holding each state's actions
Process(state)
  begin
        case state is
            when 0 => -- reset state
                led_unlocked <= '1'; --indicate system is unlocked
                led_locked <= '0';
                led_status <= "010"; --outputs green
                state_out <= 0;

            when 1 =>  --successful locked state
              --led status
                led_unlocked <= '0'; --indicate system is locked
                led_locked <= '1';
                led_status <= "000"; --outputs black/nothing
                state_out <= 1;
```

```
            when 2 =>  --successful unlocked state
                led_unlocked <= '1'; --indicate system is unlocked
                led_locked <= '0';
                led_status <= "000"; --outputs black/nothing
                state_out <= 2;

            when 3 =>  --failed unlock operation
                led_unlocked <= '0'; --indicate system is locked
                led_locked <= '1';
                led_status <= "100"; --outputs red
                state_out <= 3;

            when 4 => --invalid lock operation
                led_unlocked <= '0'; --indicate system is locked
                led_locked <= '1';
                led_status <= "001"; --outputs blue
                state_out <= 4;

            when 5 =>  --invalid unlock operation
                led_unlocked <= '1'; --indicate system is unlocked
                led_locked <= '0';
                led_status <= "110"; --outputs yellow
                state_out <= 5;

            when 6 => --idle state for state 1
                led_status <= "000"; --outputs black/nothing
                state_out <= 6;

            when 7 => --idle state for state 2
                led_status <= "000"; --outputs black/nothing
                state_out <= 7;

        end case;
    end Process;

--storing the lock codes
    storage_lock_1 <= "00" when state = 0 else
                lock_code_1 when state = 1;

    storage_lock_2 <= "00" when state = 0 else
```

```
                    lock_code_2 when state = 1;

        storage_lock_3 <= "00" when state = 0 else
                        lock_code_3 when state = 1;

        storage_lock_4 <= "00" when state = 0 else
                    lock_code_4 when state = 1;

        storage_unlock_1 <= "00" when state = 0 else
                    unlock_code_1 when state = 2;

        storage_unlock_2 <= "00" when state = 0 else
                    unlock_code_2 when state = 2;

        storage_unlock_3 <= "00" when state = 0 else
                    unlock_code_3 when state = 2;

        storage_unlock_4 <= "00" when state = 0 else
                    unlock_code_4 when state = 2;

    --outputting the stored locked codes
        lock_code_storage_1 <= storage_lock_1;
        lock_code_storage_2 <= storage_lock_2;
        lock_code_storage_3 <= storage_lock_3;
        lock_code_storage_4 <= storage_lock_4;
        unlock_code_storage_1 <= storage_unlock_1;
        unlock_code_storage_2 <= storage_unlock_2;
        unlock_code_storage_3 <= storage_unlock_3;
        unlock_code_storage_4 <= storage_unlock_4;

end Behavioral;
```

Seven-segment Display:

This entity files controls the seven-segment display. It multiplexes the seven-segment display since we are not able to control each of the seven-segments separately with their own individual output. This entity receives the state the system is in and the user code inputs. Based off that state it decides what to output. For example, it will output the lock code in the locked

states. The clock in here act as a timer for the seven-segment display. It makes it so that it acts at 60 Hz (0.02 sec for each seven-segment).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SevSeg is
  Port(
      lock_code_storage_1: in std_logic_vector(1 downto 0);
      lock_code_storage_2: in std_logic_vector(1 downto 0);
      lock_code_storage_3: in std_logic_vector(1 downto 0);
      lock_code_storage_4: in std_logic_vector(1 downto 0);
      unlock_code_storage_1: in std_logic_vector(1 downto 0);
      unlock_code_storage_2: in std_logic_vector(1 downto 0);
      unlock_code_storage_3: in std_logic_vector(1 downto 0);
      unlock_code_storage_4: in std_logic_vector(1 downto 0);
      LED7: out std_logic_vector(7 downto 0); --output for the LEDS
      AN: out std_logic_vector(7 downto 0);  --decides what sevenseg is on
      clk: in std_logic;
      state_input: in Integer range 0 to 7
  );
end SevSeg;

architecture Behavioral of SevSeg is
   --signals for tranlsating Binary to number on SevSeg
   signal Bin: std_logic_vector(3 downto 0);

   --creating a type for the states
   type eg_state_type is (s0, s1, s2, s3, s4, s5, s6, s7);

   --signals for states
   signal state_reg, state_next: eg_state_type;    --for FSM states

begin

--internal clock is at 100MHZ
--wanna refresh at 60 Hz which is 0.016 sec
-- there are 8 sevseg therefore we want each on for 0.002 sec
-- 0.002 sec = 500 HZ
-- 100MHZ/550 HZ = 200,000
```

```
-- therefore have to count up to 200,000
--for simulation change counter to 1
    timer: Process(clk)
        variable counter: integer := 0;          -- setting counter to implement a cycle
        begin
            if(rising_edge(clk)) then
                if(counter = 200000) then
                    state_reg <= state_next;
                    counter := 0;
                else
                    counter := counter + 1;
                end if;
            end if;
    end Process timer;


   --on simulation the AN will appear wrong because we have the LED refreshing right to left here and not basesd off numbers
   --Process for switching the 7SEG Light (AN)
   LED: Process(state_reg)
       begin
          if(state_input = 1) then
           case state_reg is
                when  s0 =>
                   Bin <= "00" & lock_code_storage_4;
                   AN <= "11111110";    --enabling first LED
                   state_next <= s1; --incrementing to next state


                when  s1 =>
                   Bin <= "00" & lock_code_storage_3;
                   AN <= "11111101";    --enabling second LED
                   state_next <= s2; --incrementing to next state


                when  s2 =>
                   Bin <= "00" & lock_code_storage_2;
                   AN <= "11111011";    --enabling third LEDS
                   state_next <= s3; --incrementing to next state
```

```
        when  s3 =>
           Bin <= "00" & lock_code_storage_1;
           AN <= "11110111";    --enabling fourth LED
           state_next <= s4; --incrementing to next state


        when  s4 =>
           Bin <= "1100";       -- display C
           AN <= "11101111";    --enabling fifth LED
           state_next <= s5; --incrementing to next state


        when  s5 =>
           Bin <= "1100";       -- display C
           AN <= "11011111";    --enabling sixth LED
           state_next <= s6; --incrementing to next state


        when  s6 =>
           Bin <= "1100";       -- display C
           AN <= "10111111";    --enabling seventh LED
           state_next <= s7; --incrementing to next state

        when  s7 =>
           Bin <= "1100";       -- display C
           AN <= "01111111";    --enabling 8th LED
           state_next <= s0; --incrementing to next state
    end case;

elsif(state_input = 6) then
  case state_reg is
        when  s0 =>
           Bin <= "00" & lock_code_storage_4;
           AN <= "11111110";    --enabling first LED
           state_next <= s1; --incrementing to next state


        when  s1 =>
           Bin <= "00" & lock_code_storage_3;
```

```
            AN <= "11111101";    --enabling second LED
            state_next <= s2; --incrementing to next state


         when  s2 =>
            Bin <= "00" & lock_code_storage_2;
            AN <= "11111011";    --enabling third LEDS
            state_next <= s3; --incrementing to next state


         when  s3 =>
            Bin <= "00" & lock_code_storage_1;
            AN <= "11110111";    --enabling fourth LED
            state_next <= s4; --incrementing to next state


         when  s4 =>
            Bin <= "1100";       -- display C
            AN <= "11101111";    --enabling fifth LED
            state_next <= s5; --incrementing to next state


         when  s5 =>
            Bin <= "1100";       -- display C
            AN <= "11011111";    --enabling sixth LED
            state_next <= s6; --incrementing to next state


         when  s6 =>
            Bin <= "1100";       -- display C
            AN <= "10111111";    --enabling seventh LED
            state_next <= s7; --incrementing to next state

         when  s7 =>
            Bin <= "1100";       -- display C
            AN <= "01111111";    --enabling 8th LED
            state_next <= s0; --incrementing to next state
      end case;

   elsif(state_input = 2) then
```

```
case state_reg is
      when  s0 =>
         Bin <= "1010";       -- display A
         AN <= "11111110";    --enabling first LED
         state_next <= s1; --incrementing to next state


      when  s1 =>
         Bin <= "1010";       -- display A
         AN <= "11111101";    --enabling second LED
         state_next <= s2; --incrementing to next state


      when  s2 =>
         Bin <= "1010";       -- display A
         AN <= "11111011";    --enabling third LEDS
         state_next <= s3; --incrementing to next state


      when  s3 =>
         Bin <= "1010";       -- display A
         AN <= "11110111";    --enabling fourth LED
         state_next <= s4; --incrementing to next state


      when  s4 =>
         Bin <= "00" & unlock_code_storage_4;
         AN <= "11101111";    --enabling fifth LED
         state_next <= s5; --incrementing to next state


      when  s5 =>
         Bin <= "00" & unlock_code_storage_3;
         AN <= "11011111";    --enabling sixth LED
         state_next <= s6; --incrementing to next state


      when  s6 =>
         Bin <= "00" & unlock_code_storage_2;
         AN <= "10111111";    --enabling seventh LED
```

```
            state_next <= s7; --incrementing to next state

         when  s7 =>
            Bin <= "00" & unlock_code_storage_1;
            AN <= "01111111";    --enabling 8th LED
            state_next <= s0; --incrementing to next state
      end case;

elsif(state_input = 7) then
   case state_reg is
         when  s0 =>
            Bin <= "1010";       -- display A
            AN <= "11111110";    --enabling first LED
            state_next <= s1; --incrementing to next state


         when  s1 =>
            Bin <= "1010";       -- display A
            AN <= "11111101";    --enabling second LED
            state_next <= s2; --incrementing to next state


         when  s2 =>
            Bin <= "1010";       -- display A
            AN <= "11111011";    --enabling third LEDS
            state_next <= s3; --incrementing to next state


         when  s3 =>
            Bin <= "1010";       -- display A
            AN <= "11110111";    --enabling fourth LED
            state_next <= s4; --incrementing to next state


         when  s4 =>
            Bin <= "00" & unlock_code_storage_4;
            AN <= "11101111";    --enabling fifth LED
            state_next <= s5; --incrementing to next state
```

```
        when  s5 =>
           Bin <= "00" & unlock_code_storage_3;
           AN <= "11011111";    --enabling sixth LED
           state_next <= s6; --incrementing to next state


        when  s6 =>
           Bin <= "00" & unlock_code_storage_2;
           AN <= "10111111";    --enabling seventh LED
           state_next <= s7; --incrementing to next state

        when  s7 =>
           Bin <= "00" & unlock_code_storage_1;
           AN <= "01111111";    --enabling 8th LED
           state_next <= s0; --incrementing to next state
    end case;

elsif(state_input = 3) then
  case state_reg is
        when  s0 =>
           Bin <= "00" & lock_code_storage_4;
           AN <= "11111110";    --enabling first LED
           state_next <= s1; --incrementing to next state


        when  s1 =>
           Bin <= "00" & lock_code_storage_3;
           AN <= "11111101";    --enabling second LED
           state_next <= s2; --incrementing to next state


        when  s2 =>
           Bin <= "00" & lock_code_storage_2;
           AN <= "11111011";    --enabling third LEDS
           state_next <= s3; --incrementing to next state


        when  s3 =>
           Bin <= "00" & lock_code_storage_1;
           AN <= "11110111";    --enabling fourth LED
```

```
            state_next <= s4; --incrementing to next state


         when  s4 =>
            Bin <= "1100";       -- display C
            AN <= "11101111";    --enabling fifth LED
            state_next <= s5; --incrementing to next state


         when  s5 =>
            Bin <= "1100";       -- display C
            AN <= "11011111";    --enabling sixth LED
            state_next <= s6; --incrementing to next state


         when  s6 =>
            Bin <= "1100";       -- display C
            AN <= "10111111";    --enabling seventh LED
            state_next <= s7; --incrementing to next state

         when  s7 =>
            Bin <= "1100";       -- display C
            AN <= "01111111";    --enabling 8th LED
            state_next <= s0; --incrementing to next state
      end case;

elsif(state_input = 4) then
   case state_reg is
         when  s0 =>
            Bin <= "00" & lock_code_storage_4;
            AN <= "11111110";    --enabling first LED
            state_next <= s1; --incrementing to next state


         when  s1 =>
            Bin <= "00" & lock_code_storage_3;
            AN <= "11111101";    --enabling second LED
            state_next <= s2; --incrementing to next state
```

```
        when  s2 =>
           Bin <= "00" & lock_code_storage_2;
           AN <= "11111011";    --enabling third LEDS
           state_next <= s3; --incrementing to next state


        when  s3 =>
           Bin <= "00" & lock_code_storage_1;
           AN <= "11110111";    --enabling fourth LED
           state_next <= s4; --incrementing to next state


        when  s4 =>
           Bin <= "1100";       -- display C
           AN <= "11101111";    --enabling fifth LED
           state_next <= s5; --incrementing to next state


        when  s5 =>
           Bin <= "1100";       -- display C
           AN <= "11011111";    --enabling sixth LED
           state_next <= s6; --incrementing to next state


        when  s6 =>
           Bin <= "1100";       -- display C
           AN <= "10111111";    --enabling seventh LED
           state_next <= s7; --incrementing to next state

        when  s7 =>
           Bin <= "1100";       -- display C
           AN <= "01111111";    --enabling 8th LED
           state_next <= s0; --incrementing to next state
     end case;

elsif(state_input = 5) then
   case state_reg is
        when  s0 =>
           Bin <= "1010";       -- display A
           AN <= "11111110";    --enabling first LED
```

```
    state_next <= s1; --incrementing to next state


when  s1 =>
    Bin <= "1010";       -- display A
    AN <= "11111101";    --enabling second LED
    state_next <= s2; --incrementing to next state


when  s2 =>
    Bin <= "1010";       -- display A
    AN <= "11111011";    --enabling third LEDS
    state_next <= s3; --incrementing to next state


when  s3 =>
    Bin <= "1010";       -- display A
    AN <= "11110111";    --enabling fourth LED
    state_next <= s4; --incrementing to next state


when  s4 =>
    Bin <= "00" & unlock_code_storage_4;
    AN <= "11101111";    --enabling fifth LED
    state_next <= s5; --incrementing to next state


when  s5 =>
    Bin <= "00" & unlock_code_storage_3;
    AN <= "11011111";    --enabling sixth LED
    state_next <= s6; --incrementing to next state


when  s6 =>
    Bin <= "00" & unlock_code_storage_2;
    AN <= "10111111";    --enabling seventh LED
    state_next <= s7; --incrementing to next state

when  s7 =>
    Bin <= "00" & unlock_code_storage_1;
```

```
            AN <= "01111111";    --enabling 8th LED
            state_next <= s0; --incrementing to next state
        end case;

    else
        case state_reg is
            when  s0 =>
                Bin <= "00" & lock_code_storage_4;
                AN <= "11111110";    --enabling first LED
                state_next <= s1; --incrementing to next state


            when  s1 =>
                Bin <= "00" & lock_code_storage_3;
                AN <= "11111101";    --enabling second LED
                state_next <= s2; --incrementing to next state


            when  s2 =>
                Bin <= "00" & lock_code_storage_2;
                AN <= "11111011";    --enabling third LEDS
                state_next <= s3; --incrementing to next state


            when  s3 =>
                Bin <= "00" & lock_code_storage_1;
                AN <= "11110111";    --enabling fourth LED
                state_next <= s4; --incrementing to next state


            when  s4 =>
                Bin <= "00" & unlock_code_storage_4;
                AN <= "11101111";    --enabling fifth LED
                state_next <= s5; --incrementing to next state


            when  s5 =>
                Bin <= "00" & unlock_code_storage_3;
                AN <= "11011111";    --enabling sixth LED
                state_next <= s6; --incrementing to next state
```

```
                when  s6 =>
                   Bin <= "00" & unlock_code_storage_2;
                   AN <= "10111111";    --enabling seventh LED
                   state_next <= s7; --incrementing to next state

                when  s7 =>
                   Bin <= "00" & unlock_code_storage_1;
                   AN <= "01111111";    --enabling 8th LED
                   state_next <= s0; --incrementing to next state
            end case;
        end if;
    end Process LED;

     --output that results from each input
     --LED7 = | CA | CB | CC | CD | CE | CF | CG | DP
         LED7 <= "00000011" when Bin <= "0000" else   -- display 0 (03- hex)
                 "10011111" when Bin <= "0001" else   -- display 1 (9F- hex)
                 "00100101" when Bin <= "0010" else   -- display 2 (25- hex)
                 "00001101" when Bin <= "0011" else   -- display 3 (0D- hex)
                 "00010001" when Bin <= "1010" else   -- display A
                 "01100011" when Bin <= "1100" else   -- display C
                 "11111111";

end Behavioral;
```

Based on your design, what will be a good test bench for simulating the operation of your system? Please describe it and insert below the source code from your VHDL testbench file.

This test bench simulates each of the states. There is an extra thing added and that is the output “state_light.” This is done to make the simulation easy to understand since this says what state the simulation is currently in. This change requires making some small changes to the main entity file (structural entity), so I’ve also included that below. Also, the processes in the entity files that utilize “clk” have to be changed to ‘1’ for simulation purposes. Since the testbench doesn’t use exact same entities as above with some small changes I’ve also include their code below. The test bench is straightforward in that it doesn’t do anything complicated.

It simulates a clock that has a period of 10 ns, and then the inputs make it go through each of the states.

Structure Entity: (slight changes for "state_light")

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--structural file of Keyless Entry system

entity sim_structure is
    Port(
        lock_code_1: in std_logic_vector(1 downto 0);
        lock_code_2: in std_logic_vector(1 downto 0);
        lock_code_3: in std_logic_vector(1 downto 0);
        lock_code_4: in std_logic_vector(1 downto 0);
        unlock_code_1: in std_logic_vector(1 downto 0);
        unlock_code_2: in std_logic_vector(1 downto 0);
        unlock_code_3: in std_logic_vector(1 downto 0);
        unlock_code_4: in std_logic_vector(1 downto 0);
        unlock_button: in std_logic;
        lock_button: in std_logic;
        reset: in std_logic;
        clk: in std_logic;
        AN: out std_logic_vector(7 downto 0);
        led_locked: out std_logic;      --indicates the system is locked
        led_unlocked: out std_logic;    --indicates the system is unlocked
        led_status: out std_logic_vector(2 downto 0);
        LED7: out std_logic_vector(7 downto 0); --output the turns the leds on
        state_light: out Integer range 0 to 7       --current state it is in
    );
end sim_structure;

architecture Behavioral of sim_structure is
  --signals for connecting the components
    --signal for connecting state logic
    signal signal_state: Integer range 0 to 7;

    --will connect the stored numbers to sevseg display
    signal signal_lock_code_storage_1: std_logic_vector(1 downto 0);
```

```
signal signal_lock_code_storage_2: std_logic_vector(1 downto 0);
signal signal_lock_code_storage_3: std_logic_vector(1 downto 0);
signal signal_lock_code_storage_4: std_logic_vector(1 downto 0);
signal signal_unlock_code_storage_1: std_logic_vector(1 downto 0);
signal signal_unlock_code_storage_2: std_logic_vector(1 downto 0);
signal signal_unlock_code_storage_3: std_logic_vector(1 downto 0);
signal signal_unlock_code_storage_4: std_logic_vector(1 downto 0);

component states_simulation
   Port(
      clk: in std_logic;
      lock_code_1: in std_logic_vector(1 downto 0);
      lock_code_2: in std_logic_vector(1 downto 0);
      lock_code_3: in std_logic_vector(1 downto 0);
      lock_code_4: in std_logic_vector(1 downto 0);
      unlock_code_1: in std_logic_vector(1 downto 0);
      unlock_code_2: in std_logic_vector(1 downto 0);
      unlock_code_3: in std_logic_vector(1 downto 0);
      unlock_code_4: in std_logic_vector(1 downto 0);
      unlock_button: in std_logic;
      lock_button: in std_logic;
      reset: in std_logic;
      led_locked: out std_logic;      --indicates the system is locked
      led_unlocked: out std_logic;    --indicates the system is unlocked
      led_status: out std_logic_vector(2 downto 0);
      lock_code_storage_1: inout std_logic_vector(1 downto 0);
      lock_code_storage_2: inout std_logic_vector(1 downto 0);
      lock_code_storage_3: inout std_logic_vector(1 downto 0);
      lock_code_storage_4: inout std_logic_vector(1 downto 0);
      unlock_code_storage_1: out std_logic_vector(1 downto 0);
      unlock_code_storage_2: out std_logic_vector(1 downto 0);
      unlock_code_storage_3: out std_logic_vector(1 downto 0);
      unlock_code_storage_4: out std_logic_vector(1 downto 0);
      state_out: out Integer range 0 to 7
   );
end Component;

Component SevSeg_simulation
      Port(
      lock_code_storage_1: in std_logic_vector(1 downto 0);
```

```
        lock_code_storage_2: in std_logic_vector(1 downto 0);
        lock_code_storage_3: in std_logic_vector(1 downto 0);
        lock_code_storage_4: in std_logic_vector(1 downto 0);
        unlock_code_storage_1: in std_logic_vector(1 downto 0);
        unlock_code_storage_2: in std_logic_vector(1 downto 0);
        unlock_code_storage_3: in std_logic_vector(1 downto 0);
        unlock_code_storage_4: in std_logic_vector(1 downto 0);
        LED7: out std_logic_vector(7 downto 0); --output for the LEDS
        AN: out std_logic_vector(7 downto 0);
        clk: in std_logic;
        state_input: in Integer range 0 to 7
    );
  end Component;

begin
  States: states_simulation Port Map
  (
  clk => clk,
  lock_code_1 => lock_code_1,
  lock_code_2 => lock_code_2,
  lock_code_3 => lock_code_3,
  lock_code_4 => lock_code_4,
  unlock_code_1 => unlock_code_1,
  unlock_code_2 => unlock_code_2,
  unlock_code_3 => unlock_code_3,
  unlock_code_4 => unlock_code_4,
  unlock_button => unlock_button,
  lock_button => lock_button,
  reset => reset,
  led_locked => led_locked,
  led_unlocked => led_unlocked,
  led_status => led_status,
  lock_code_storage_1 => signal_lock_code_storage_1,
  lock_code_storage_2 => signal_lock_code_storage_2,
  lock_code_storage_3 => signal_lock_code_storage_3,
  lock_code_storage_4 => signal_lock_code_storage_4,
  unlock_code_storage_1 => signal_unlock_code_storage_1,
  unlock_code_storage_2 => signal_unlock_code_storage_2,
  unlock_code_storage_3 => signal_unlock_code_storage_3,
  unlock_code_storage_4 => signal_unlock_code_storage_4,
```

```
state_out => signal_state
);

SevenSegment: SevSeg_simulation Port Map
(
lock_code_storage_1 => signal_lock_code_storage_1,
lock_code_storage_2 => signal_lock_code_storage_2,
lock_code_storage_3 => signal_lock_code_storage_3,
lock_code_storage_4 => signal_lock_code_storage_4,
unlock_code_storage_1 => signal_unlock_code_storage_1,
unlock_code_storage_2 => signal_unlock_code_storage_2,
unlock_code_storage_3 => signal_unlock_code_storage_3,
unlock_code_storage_4 => signal_unlock_code_storage_4,
LED7 => LED7,
AN => AN,
clk => clk,
state_input => signal_state
);

 --for checking what state im in and outputting it for debugging
Process(signal_state)
   begin
      case signal_state is
         when 0 =>
            state_light <= 0;

         when 1 =>
            state_light <= 1;

         when 2 =>
            state_light <= 2;

         when 3 =>
            state_light <= 3;

         when 4 =>
            state_light <= 4;

         when 5 =>
            state_light <= 5;
```

```
                when 6 =>
                    state_light <= 6;

                when 7 =>
                    state_light <= 7;

            end case;
        end Process;
end Behavioral;
```

STATES ENTITY (CLK CHANGES):

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity states_simulation is
    Port(
        clk: in std_logic;
        lock_code_1: in std_logic_vector(1 downto 0);
        lock_code_2: in std_logic_vector(1 downto 0);
        lock_code_3: in std_logic_vector(1 downto 0);
        lock_code_4: in std_logic_vector(1 downto 0);
        unlock_code_1: in std_logic_vector(1 downto 0);
        unlock_code_2: in std_logic_vector(1 downto 0);
        unlock_code_3: in std_logic_vector(1 downto 0);
        unlock_code_4: in std_logic_vector(1 downto 0);
        unlock_button: in std_logic;
        lock_button: in std_logic;
        reset: in std_logic;
        led_locked: out std_logic;      --indicates the system is locked
        led_unlocked: out std_logic;    --indicates the system is unlocked
        led_status: out std_logic_vector(2 downto 0);
        lock_code_storage_1: inout std_logic_vector(1 downto 0);
        lock_code_storage_2: inout std_logic_vector(1 downto 0);
        lock_code_storage_3: inout std_logic_vector(1 downto 0);
        lock_code_storage_4: inout std_logic_vector(1 downto 0);
        unlock_code_storage_1: out std_logic_vector(1 downto 0);
        unlock_code_storage_2: out std_logic_vector(1 downto 0);
```

```
        unlock_code_storage_3: out std_logic_vector(1 downto 0);
        unlock_code_storage_4: out std_logic_vector(1 downto 0);
        state_out: out Integer range 0 to 7
    );
end states_simulation;

architecture Behavioral of states_simulation is
    signal state: Integer range 0 to 7;
    signal next_state: Integer range 0 to 7;
    signal start: std_logic;
    signal count: Integer := 0; --counter for slwclk
    signal slowclock: std_logic;

    --signals for acting as memory
        signal storage_lock_1: std_logic_vector(1 downto 0);
        signal storage_lock_2: std_logic_vector(1 downto 0);
        signal storage_lock_3: std_logic_vector(1 downto 0);
        signal storage_lock_4: std_logic_vector(1 downto 0);
        signal storage_unlock_1: std_logic_vector(1 downto 0);
        signal storage_unlock_2: std_logic_vector(1 downto 0);
        signal storage_unlock_3: std_logic_vector(1 downto 0);
        signal storage_unlock_4: std_logic_vector(1 downto 0);



begin



     -- change 50000000 to 1 when simulating
    --process that will act as buff so there's delay between inputs(minimize bounce)
     debounce: Process(clk)
        begin
            if((rising_edge(clk)) and (start = '1')) then
                if(count = 1) then -- 0.5 sec delay based off 100MHZ clk before next input
can be taken
                    slowclock <= '1';
                    count <= 0; --reset counter
                else
                    count <= count + 1;
```

```
            end if;
        end if;
        if(slowclock = '1') then
            slowclock <= '0';
        end if;
    end Process debounce;

    --Process that will tranist the state to the nextstate
    transit: Process(slowclock)
      begin
        if(rising_edge(slowclock)) then
         state <= next_state;
        end if;
    end Process ;

    --dont want this to trigger off clock because then it will reset "start" back to 0 too fast
    starter: Process(lock_button, unlock_button, reset, state, clk)
      begin
        if(lock_button = '1' or unlock_button = '1' or reset = '1' or state = 1 or state = 2) then
            start <= '1';

        elsif(state = next_state) then
            start <= '0';
        end if;
    end Process starter;

   --process that holds conditions for entering states
   Process(lock_button, unlock_button, reset, clk)
    begin
      if(state = next_state) then -- to act as a debounce (already in the next state)
        --condition for entering state 0 (reset state = unlocked state) - green
        if(reset = '1') then
            next_state <= 0;
        end if;

        --condition for entering state 1 (locked state) - no led
        if(lock_button = '1' and (state = 0 or state = 7 or state = 5)) then
            next_state <= 1;
```

```
        end if;

        --condition for entering state 2 (unlocked state) -- no led
        if(unlock_button = '1' and (state = 6 or state = 3 or state = 4)  and    --making
sure codes match
             unlock_code_1 = lock_code_storage_1 and
             unlock_code_2 = lock_code_storage_2 and
             unlock_code_3 = lock_code_storage_3 and
             unlock_code_4 = lock_code_storage_4) then
              next_state <= 2;
        end if;

        --condition for entering state 3 (failed unlock - remain locked) - red
        if((unlock_button = '1' and (state = 6 or state = 4)) and   --checking if codes dont
match
             (unlock_code_1 /= lock_code_storage_1 or
             unlock_code_2 /= lock_code_storage_2 or
             unlock_code_3 /= lock_code_storage_3 or
             unlock_code_4 /= lock_code_storage_4)) then
              next_state <= 3;
       end if;

        --condition for entering state 4 (invalid lock - remain locked) - blue
        if(lock_button = '1' and (state = 6 or state = 3)) then
              next_state <= 4;
        end if;

        --condition for entering state 5 (invalid unlock - remain unlocked) - yellow
        if(unlock_button = '1' and (state = 7 or state = 0))  then
              next_state <= 5;
        end if;

        if(state = 1) then
           next_state <= 6;
        end if;

        if(state = 2) then
           next_state <= 7;
        end if;
      end if;
```

```
end Process;



--process holding each state's actions
Process(state)
  begin
       case state is
           when 0 => -- reset state
               led_unlocked <= '1'; --indicate system is unlocked
               led_locked <= '0';
               led_status <= "010"; --outputs green
               state_out <= 0;

           when 1 =>  --successful locked state
             --led status
               led_unlocked <= '0'; --indicate system is locked
               led_locked <= '1';
               led_status <= "000"; --outputs black/nothing
               state_out <= 1;

           when 2 =>  --successful unlocked state
               led_unlocked <= '1'; --indicate system is unlocked
               led_locked <= '0';
               led_status <= "000"; --outputs black/nothing
               state_out <= 2;

           when 3 =>  --failed unlock operation
               led_unlocked <= '0'; --indicate system is locked
               led_locked <= '1';
               led_status <= "100"; --outputs red
               state_out <= 3;

           when 4 => --invalid lock operation
               led_unlocked <= '0'; --indicate system is locked
               led_locked <= '1';
               led_status <= "001"; --outputs blue
               state_out <= 4;

           when 5 =>  --invalid unlock operation
```

```
            led_unlocked <= '1'; --indicate system is unlocked
            led_locked <= '0';
            led_status <= "110"; --outputs yellow
            state_out <= 5;

        when 6 => --idle state for state 1
            led_status <= "000"; --outputs black/nothing
            state_out <= 6;

        when 7 => --idle state for state 2
            led_status <= "000"; --outputs black/nothing
            state_out <= 7;

    end case;
end Process;

  storage_lock_1 <= "00" when state = 0 else
              lock_code_1 when state = 1;

  storage_lock_2 <= "00" when state = 0 else
              lock_code_2 when state = 1;

  storage_lock_3 <= "00" when state = 0 else
                lock_code_3 when state = 1;

  storage_lock_4 <= "00" when state = 0 else
              lock_code_4 when state = 1;

  storage_unlock_1 <= "00" when state = 0 else
              unlock_code_1 when state = 2;

  storage_unlock_2 <= "00" when state = 0 else
              unlock_code_2 when state = 2;

  storage_unlock_3 <= "00" when state = 0 else
             unlock_code_3 when state = 2;

  storage_unlock_4 <= "00" when state = 0 else
              unlock_code_4 when state = 2;
```

```
    lock_code_storage_1 <= storage_lock_1;
    lock_code_storage_2 <= storage_lock_2;
    lock_code_storage_3 <= storage_lock_3;
    lock_code_storage_4 <= storage_lock_4;
    unlock_code_storage_1 <= storage_unlock_1;
    unlock_code_storage_2 <= storage_unlock_2;
    unlock_code_storage_3 <= storage_unlock_3;
    unlock_code_storage_4 <= storage_unlock_4;


end Behavioral;
```

SEVEN SEGMENT (CLK CHANGES):

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SevSeg_simulation is
  Port(
      lock_code_storage_1: in std_logic_vector(1 downto 0);
      lock_code_storage_2: in std_logic_vector(1 downto 0);
      lock_code_storage_3: in std_logic_vector(1 downto 0);
      lock_code_storage_4: in std_logic_vector(1 downto 0);
      unlock_code_storage_1: in std_logic_vector(1 downto 0);
      unlock_code_storage_2: in std_logic_vector(1 downto 0);
      unlock_code_storage_3: in std_logic_vector(1 downto 0);
      unlock_code_storage_4: in std_logic_vector(1 downto 0);
      LED7: out std_logic_vector(7 downto 0); --output for the LEDS
      AN: out std_logic_vector(7 downto 0);
      clk: in std_logic;
      state_input: in Integer range 0 to 7
  );
end SevSeg_simulation;

architecture Behavioral of SevSeg_simulation is
  --signals for tranlsating Binary to number on SevSeg
  signal Bin: std_logic_vector(3 downto 0);
```

```
    --creating a type for the states
    type eg_state_type is (s0, s1, s2, s3, s4, s5, s6, s7);

    --signals for states
    signal state_reg, state_next: eg_state_type;    --for FSM states

begin

--internal clock is at 100MHZ
--wanna refresh at 60 Hz which is 0.016 sec
-- there are 8 sevseg therefore we want each on for 0.002 sec
-- 0.002 sec = 500 HZ
-- 100MHZ/550 HZ = 200,000
-- therefore have to count up to 200,000
--for simulation change counter to 1
    timer: Process(clk)
        variable counter: integer := 0;          -- setting counter to implement a cycle
        begin
            if(rising_edge(clk)) then
                if(counter = 1) then
                    state_reg <= state_next;
                    counter := 0;
                else
                    counter := counter + 1;
                end if;
            end if;
    end Process timer;


     --on simulation the AN will appear wrong because we have the LED refreshing
right to left here and not based off numbers
   --Process for switching the 7SEG Light (AN)
   LED: Process(state_reg)
       begin
          if(state_input = 3) then
           case state_reg is
                when  s0 =>
                    Bin <= "00" & lock_code_storage_4;
                    AN <= "11111110";    --enabling first LED
                    state_next <= s1; --incrementing to next state
```

```
when  s1 =>
   Bin <= "00" & lock_code_storage_3;
   AN <= "11111101";    --enabling second LED
   state_next <= s2; --incrementing to next state


when  s2 =>
   Bin <= "00" & lock_code_storage_2;
   AN <= "11111011";    --enabling third LEDS
   state_next <= s3; --incrementing to next state


when  s3 =>
   Bin <= "00" & lock_code_storage_1;
   AN <= "11110111";    --enabling fourth LED
   state_next <= s4; --incrementing to next state


when  s4 =>
   Bin <= "1100";
   AN <= "11101111";    --enabling fifth LED
   state_next <= s5; --incrementing to next state


when  s5 =>
   Bin <= "1100";
   AN <= "11011111";    --enabling sixth LED
   state_next <= s6; --incrementing to next state


when  s6 =>
   Bin <= "1100";
   AN <= "10111111";    --enabling seventh LED
   state_next <= s7; --incrementing to next state

when  s7 =>
   Bin <= "1100";
   AN <= "01111111";    --enabling 8th LED
```

```
            state_next <= s0; --incrementing to next state
        end case;
elsif(state_input = 4) then
    case state_reg is
        when  s0 =>
            Bin <= "00" & lock_code_storage_4;
            AN <= "11111110";    --enabling first LED
            state_next <= s1; --incrementing to next state


        when  s1 =>
            Bin <= "00" & lock_code_storage_3;
            AN <= "11111101";    --enabling second LED
            state_next <= s2; --incrementing to next state


        when  s2 =>
            Bin <= "00" & lock_code_storage_2;
            AN <= "11111011";    --enabling third LEDS
            state_next <= s3; --incrementing to next state


        when  s3 =>
            Bin <= "00" & lock_code_storage_1;
            AN <= "11110111";    --enabling fourth LED
            state_next <= s4; --incrementing to next state


        when  s4 =>
            Bin <= "1100";
            AN <= "11101111";    --enabling fifth LED
            state_next <= s5; --incrementing to next state


        when  s5 =>
            Bin <= "1100";
            AN <= "11011111";    --enabling sixth LED
            state_next <= s6; --incrementing to next state
```

```
        when  s6 =>
           Bin <= "1100";
           AN <= "10111111";    --enabling seventh LED
           state_next <= s7; --incrementing to next state

        when  s7 =>
           Bin <= "1100";
           AN <= "01111111";    --enabling 8th LED
           state_next <= s0; --incrementing to next state
     end case;

elsif(state_input = 5) then
   case state_reg is
        when  s0 =>
           Bin <= "1010";
           AN <= "11111110";    --enabling first LED
           state_next <= s1; --incrementing to next state


        when  s1 =>
           Bin <= "1010";
           AN <= "11111101";    --enabling second LED
           state_next <= s2; --incrementing to next state


        when  s2 =>
           Bin <= "1010";
           AN <= "11111011";    --enabling third LEDS
           state_next <= s3; --incrementing to next state


        when  s3 =>
           Bin <= "1010";
           AN <= "11110111";    --enabling fourth LED
           state_next <= s4; --incrementing to next state


        when  s4 =>
           Bin <= "00" & unlock_code_storage_4;
           AN <= "11101111";    --enabling fifth LED
```

```
            state_next <= s5; --incrementing to next state


        when  s5 =>
            Bin <= "00" & unlock_code_storage_3;
            AN <= "11011111";    --enabling sixth LED
            state_next <= s6; --incrementing to next state


        when  s6 =>
            Bin <= "00" & unlock_code_storage_2;
            AN <= "10111111";    --enabling seventh LED
            state_next <= s7; --incrementing to next state

        when  s7 =>
            Bin <= "00" & unlock_code_storage_1;
            AN <= "01111111";    --enabling 8th LED
            state_next <= s0; --incrementing to next state
    end case;

else
    case state_reg is
        when  s0 =>
            Bin <= "00" & lock_code_storage_4;
            AN <= "11111110";    --enabling first LED
            state_next <= s1; --incrementing to next state


        when  s1 =>
            Bin <= "00" & lock_code_storage_3;
            AN <= "11111101";    --enabling second LED
            state_next <= s2; --incrementing to next state


        when  s2 =>
            Bin <= "00" & lock_code_storage_2;
            AN <= "11111011";    --enabling third LEDS
            state_next <= s3; --incrementing to next state
```

```
            when  s3 =>
               Bin <= "00" & lock_code_storage_1;
               AN <= "11110111";    --enabling fourth LED
               state_next <= s4; --incrementing to next state


            when  s4 =>
               Bin <= "00" & unlock_code_storage_4;
               AN <= "11101111";    --enabling fifth LED
               state_next <= s5; --incrementing to next state


            when  s5 =>
               Bin <= "00" & unlock_code_storage_3;
               AN <= "11011111";    --enabling sixth LED
               state_next <= s6; --incrementing to next state


            when  s6 =>
               Bin <= "00" & unlock_code_storage_2;
               AN <= "10111111";    --enabling seventh LED
               state_next <= s7; --incrementing to next state

            when  s7 =>
               Bin <= "00" & unlock_code_storage_1;
               AN <= "01111111";    --enabling 8th LED
               state_next <= s0; --incrementing to next state
         end case;
      end if;
end Process LED;

 --output that results from each input
 --LED7 = | CA | CB | CC | CD | CE | CF | CG | DP
      LED7 <= "00000011" when Bin <= "0000" else   -- display 0 (03- hex)
            "10011111" when Bin <= "0001" else   -- display 1 (9F- hex)
            "00100101" when Bin <= "0010" else   -- display 2 (25- hex)
            "00001101" when Bin <= "0011" else   -- display 3 (0D- hex)
            "00010001" when Bin <= "1010" else   -- display A
            "01100011" when Bin <= "1100" else   -- display C
            "11111111";
```

```
end Behavioral;
```

TEST BENCH:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity simstructure_TB is
end simstructure_TB;

architecture Behavioral of simstructure_TB is
   Component sim_structure
      Port(
         lock_code_1: in std_logic_vector(1 downto 0);
         lock_code_2: in std_logic_vector(1 downto 0);
         lock_code_3: in std_logic_vector(1 downto 0);
         lock_code_4: in std_logic_vector(1 downto 0);
         unlock_code_1: in std_logic_vector(1 downto 0);
         unlock_code_2: in std_logic_vector(1 downto 0);
         unlock_code_3: in std_logic_vector(1 downto 0);
         unlock_code_4: in std_logic_vector(1 downto 0);
         unlock_button: in std_logic;
         lock_button: in std_logic;
         reset: in std_logic;
         clk: in std_logic;
         AN: out std_logic_vector(7 downto 0);
         led_locked: out std_logic;      --indicates the system is locked
         led_unlocked: out std_logic;    --indicates the system is unlocked
         led_status: out std_logic_vector(2 downto 0);
         LED7: out std_logic_vector(7 downto 0); --output the turns the leds on
         state_light: out Integer range 0 to 7
      );
   end Component;

   --signals for simulation
    signal lock_code_1: std_logic_vector(1 downto 0);
    signal lock_code_2: std_logic_vector(1 downto 0);
    signal lock_code_3: std_logic_vector(1 downto 0);
```

```
    signal lock_code_4: std_logic_vector(1 downto 0);
    signal unlock_code_1: std_logic_vector(1 downto 0);
    signal unlock_code_2: std_logic_vector(1 downto 0);
    signal unlock_code_3: std_logic_vector(1 downto 0);
    signal unlock_code_4: std_logic_vector(1 downto 0);
    signal unlock_button: std_logic;
    signal lock_button: std_logic;
    signal reset: std_logic;
    signal clk: std_logic;
    signal AN: std_logic_vector(7 downto 0);
    signal led_locked: std_logic;      --indicates the system is locked
    signal led_unlocked: std_logic;    --indicates the system is unlocked
    signal led_status: std_logic_vector(2 downto 0);
    signal LED7: std_logic_vector(7 downto 0); --output the turns the leds on
    signal state_light: Integer range 0 to 7;

begin
   DUT: sim_structure Port Map
    (
      lock_code_1 => lock_code_1,
      lock_code_2 => lock_code_2,
      lock_code_3 => lock_code_3,
      lock_code_4 => lock_code_4,
      unlock_code_1 => unlock_code_1,
      unlock_code_2 => unlock_code_2,
      unlock_code_3 => unlock_code_3,
      unlock_code_4 => unlock_code_4,
      unlock_button => unlock_button,
      lock_button => lock_button,
      reset => reset,
      clk => clk,
      AN => AN,
      led_locked => led_locked,
      led_unlocked => led_unlocked,
      led_status => led_status,
      LED7 => LED7,
      state_light => state_light
    );

    --creating testbench clock with process
```

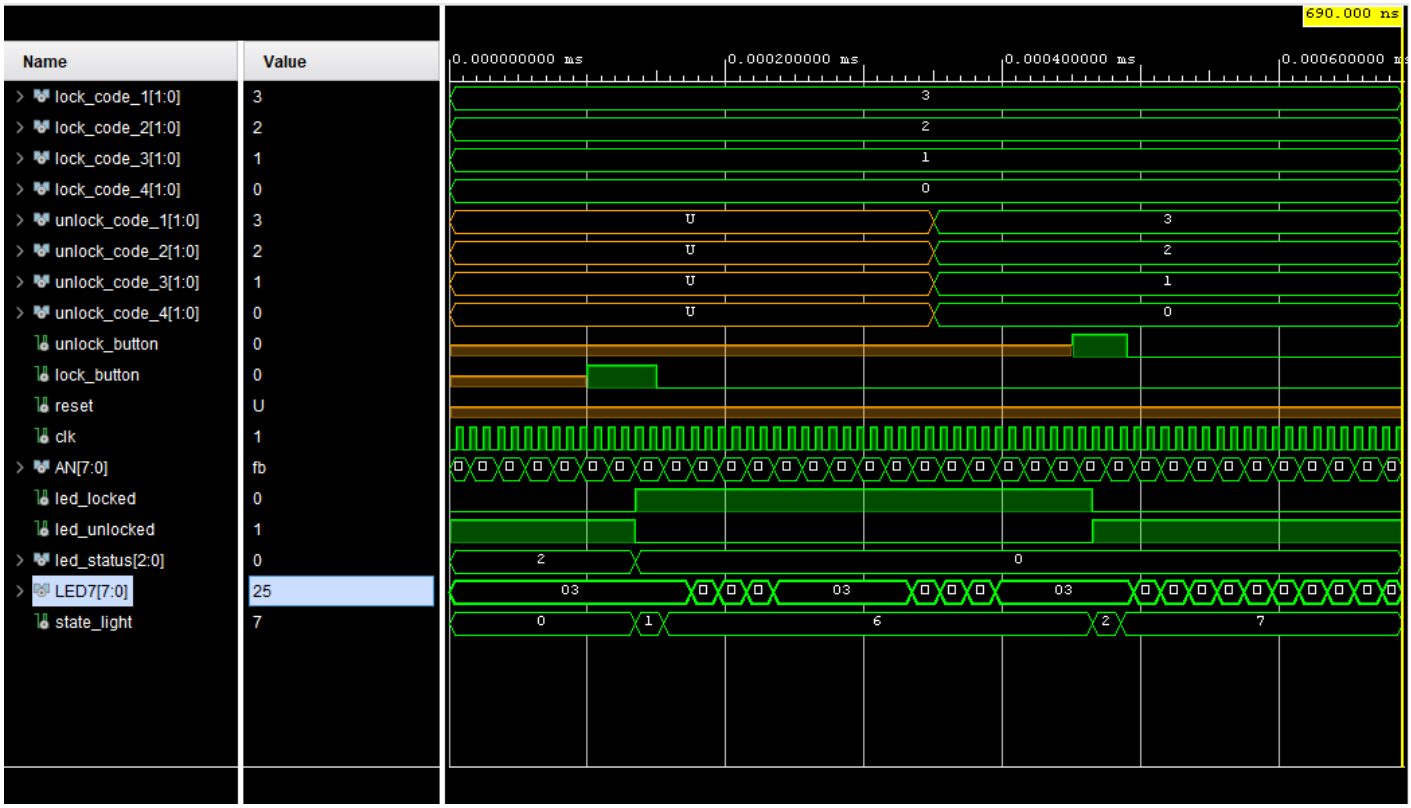
```
cycle: Process
   begin
       clk <= '0';
       wait for 5ns;
       clk <= '1';
       wait for 5ns;
end Process;

Stimulus: Process
   begin
      --testing lock case (state 1)
      lock_code_1 <= "11";
      lock_code_2 <= "10";
      lock_code_3 <= "01";
      lock_code_4 <= "00";
      wait for 100 ns;
      lock_button <= '1';
      wait for 50 ns;
      lock_button <= '0';
      wait for 100 ns;

      --testing invalid lock case (state 4)
      lock_button <= '1';
      wait for 50 ns;
      lock_button <= '0';
      wait for 100 ns;
      --results good

       --testing unlock (state 2 and 3)
      unlock_code_1 <= "11";
      unlock_code_2 <= "10";
      unlock_code_3 <= "01";
      unlock_code_4 <= "01";
      wait for 100ns;
      unlock_button <= '1';
      wait for 40ns;
      unlock_button <= '0';
      wait for 100ns;
      --results: passed "successful and failed" unlock status
```

```
            --general reset at the end (state 0)
            reset <= '1';
            wait for 50 ns;
            reset <= '0';
            wait for 50 ns;
            --results: good

            --testing invalid unlock(state 5)
            unlock_code_1 <= "10";
            unlock_code_2 <= "10";
            unlock_code_3 <= "10";
            unlock_code_4 <= "10";
            wait for 100ns;
            unlock_button <= '1';
            wait for 100ns;
            unlock_button <= '0';
            wait for 100ns;
            --results good

        --ending the process
        assert false report "Test: OK" severity failure;
    end Process;
end Behavioral;
```

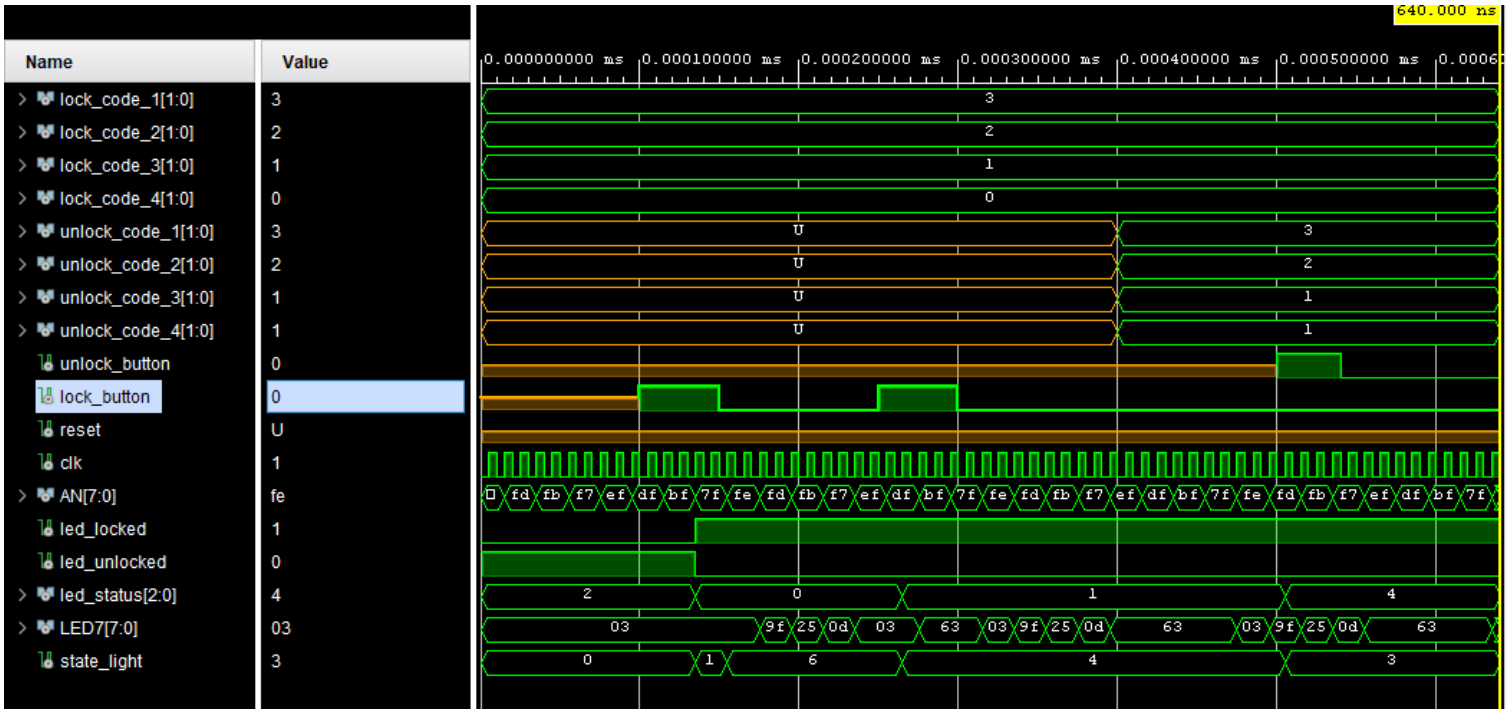
What is the result waveform from your simulation? Based on this simulation, is your system implementation functioning as designed?

Note* - I've split up the simulation to make it simpler and just show it accesses each state

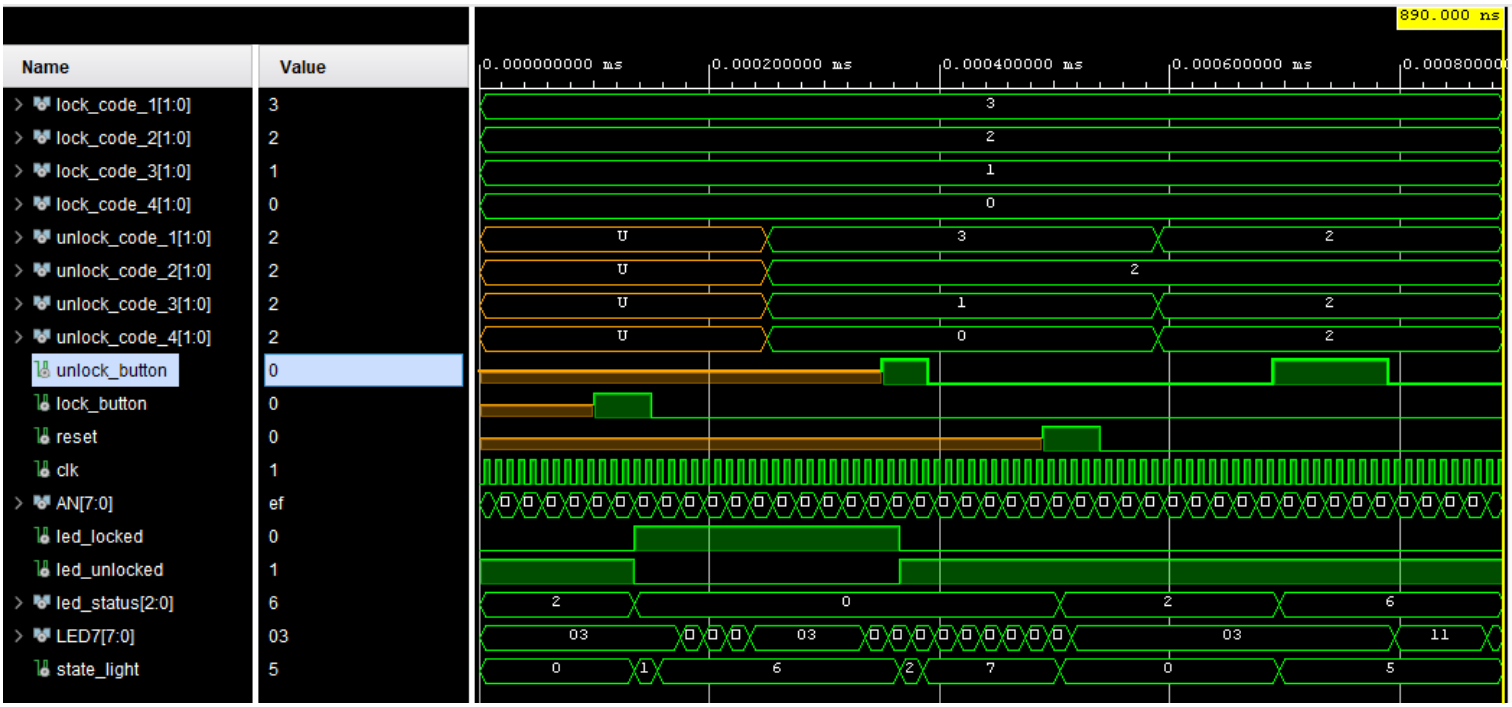
This simulation is only a portion for ease of understanding. The "state_light" is an integer used in the code to see what state it is currently in. I've included this in to make it easier to understand the simulation. There are 4 lock inputs simulated with the lock_button being triggered for 50ns at the 100ns mark. This causes it to go into state 1 (locked state) and then goes to state 6(idle state for state 1). There are then 4 unlocked codes submitted which is then taken when the "unlock_button" is triggered at 450ns mark for 40ns. Since the codes match it enters state 2 (unlocked state) which then enters state 7 on its own (idle state for state 2). Also the RGB led's make sense as it start off green (2_{dec}) and then stays black (0_{dec}) for the rest of the cycle.



Here the same thing is tested as above but with an invalid unlock state(state 4) and an incorrect unlock code(state 3). It goes through the same process initially, but I trigger “lock_button” again which causes it to enter the invalid unlock state (state 4) and cause “led_status” to become blue (1_{dec}). I then simulate an incorrect unlock code and trigger the “unlock button” so it enters the failed unlock state (state 3) and causes the “led_status” to become red (4_{dec}).



This tests the last two states: state 0 and state 5 (reset and unlock state). It goes through the same process as the first simulation and then the reset button is triggered at 490ns which causes it to enter state 0 and make "led_status" output green (2_{dec}). The "unlock_button" is triggered again here which causes it to enter the invalid unlock state (state 5) and make "led_status change to yellow (6_{dec} or 110_{bin}).



After the QuestaSim simulation, please implement your keyless entry system on the Nexys 4 DDR board, and use the setup as given earlier.

After implementing your keyless entry system design on the Nexys 4 board, please demonstrate it to the instructor; and create a short video demonstrating your test results. Please upload the short video as a separate file to this report in Titanium.

Google drive link:
https://drive.google.com/file/d/1yywyXUmRUoOtsTun0QwauKIdv6jaImkK/view?usp=sharing

Grading of the Project:

Your design project will be graded based on both the report and the in-class demonstrations before the due-date. The following factors will be considered when grading this project.

- Does your design fully comply with and meet all the design specifications as described?
    - Points will be deducted for each deviation from the specifications
    - Partial credits will be given depending on how much progress was made
- Your VHDL entity code and Testbench design with simulation analysis.
- The extent to which your design is implemented on the Nexys 4 DDR board.
    - Points will be deducted for each deviation from the specifications
    - Partial credits will be given depending on how much progress was made
- The extent to which your written report and oral presentation in the short video explains how your system is designed, and how the simulation and board implementation verifies your system design.