Koç University

College of Engineering

Department of Electrical & Electronics Engineering

Project Report

ASCII LOCKER

Date: 11/27/2019

Prepared by: Oğuzhan Taş, Metehan Gelgi

1. Introduction and Objective

1.1 Aim of the Project:

Design and Implementation of Lock systems to the FPGA Board with replaceable password. This project is simulating simple lock system: There is 60 seconds or 3 attempts to unlock system. Otherwise system will be locked. Then the only way to unlock system is Personal Unblocking Code (PUC). Also, this lock system has changeable password. All these procedures similar with regular lock systems.

1.2 Equipment & Software used:

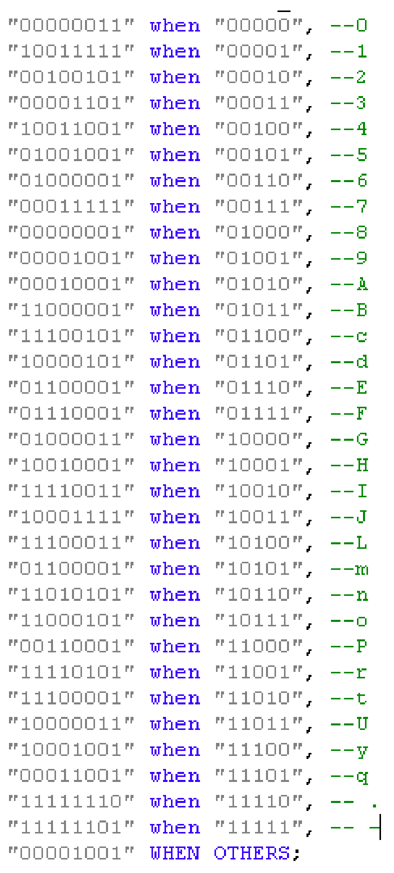
- IBM Compatible PC with Windows 7 operating system
- Xilinx ISE v1.47 & Prometheus software packages
- Prometheus FPGA (Xilinx Spartan 3A) board
- USB cable for programming.

2. Methodology

2.1 ASCII Code:

ASCII, abbreviated from American Standard Code for Information Interchange, is a character encoding standard for electronic communication. It's a 7-bit character code where every single bit represents a unique character.

In this Project instead of 7-bit, characters coded with 5-bit binary numbers due to insufficient number of pins. Some characters are not used because they cannot be shown in 7-segment display. By this kind of ASCII code 32 different characters can be shown on on 7-segment display.



```
"00000011" when "00000", --0
"10011111" when "00001", --1
"00100101" when "00010", --2
"00001101" when "00011", --3
"10011001" when "00100", --4
"01001001" when "00101", --5
"01000001" when "00110", --6
"00011111" when "00111", --7
"00000001" when "01000", --8
"00001001" when "01001", --9
"00010001" when "01010", --A
"11000001" when "01011", --B
"11100101" when "01100", --c
"10000101" when "01101", --d
"01100001" when "01110", --E
"01110001" when "01111", --F
"01000011" when "10000", --G
"10010001" when "10001", --H
"11110011" when "10010", --I
"10001111" when "10011", --J
"11100011" when "10100", --L
"01100001" when "10101", --m
"11010101" when "10110", --n
"11000101" when "10111", --o
"00110001" when "11000", --P
"11110101" when "11001", --r
"11100001" when "11010", --t
"10000011" when "11011", --U
"10001001" when "11100", --y
"00011001" when "11101", --q
"11111110" when "11110", -- .
"11111101" when "11111", -- -
"00001001" WHEN OTHERS;
```

Figure 1. binary codes of characters

2.2 Design:

Steps of the Lock:

1) Initial Password is AA (in **Figure (1)**. AA= “01010 01010”)
2) The clock starts countdown (60 seconds to 0) before remaining time is 0 change pins to AA then push to P36, which is change statement button, to lock system. (“off” will appear on the screen, time and number of attempts will reset itself)
3) And also, there is 3 attempts to enter password before the time is up.

4) After entering right password, system can be unlocked again(“opn” will appear on the screen, time and number of attempts will reset itself) by pushing P36 again.
5) For both lock and unlock states if password is wrong, timer starts countdown. Also, one of the trial rights lost.
6) If time ends or trial runs out, there will be needed the Personal Unblocking Code (PUC). (mc = “10101 01100”) There is no other way to bring the system back. (current password will not work.)
7) To change password, system should be unlocked (“opn” has to be appear on the screen). Then while pushing P32, push P36(which is change statement button). Then change password state can be accessible “‘chn’ can be seen on screen”. The new password can be set without releasing P32 button. After choosing new password, P36 is pressed again.

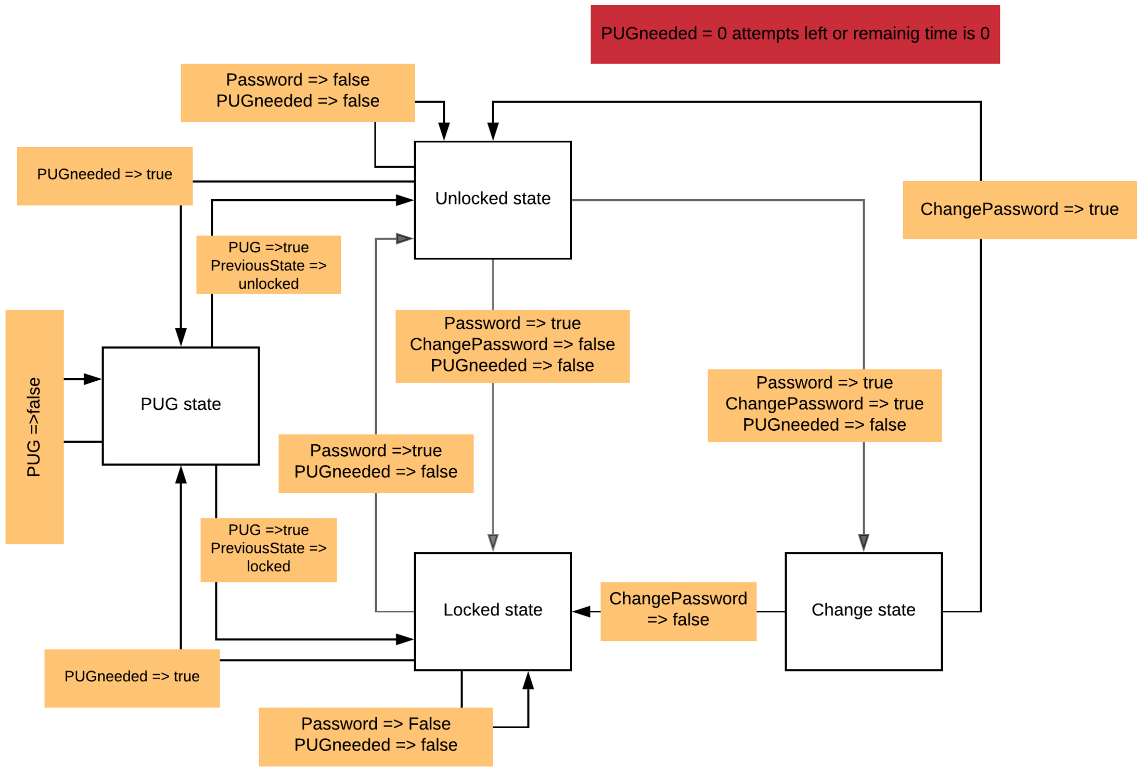


Figure 2. Schematic representation of the states and steps of the lock system.

Implementation:

The lock system is implemented with the following states:

Unlocked State

Unlocked state is the initiation state. When in Unlocked state and the password is true next state is locked unless changepassword button pressed. Otherwise next state is change state. After PUGneeded is true, next state is PUG state.

Locked State

In this state system is locked. Next state is unlock system if right password comes with button. Otherwise trial right is reduced by 1. If PUGneeded is true next state becomes PUG state.

PUG State

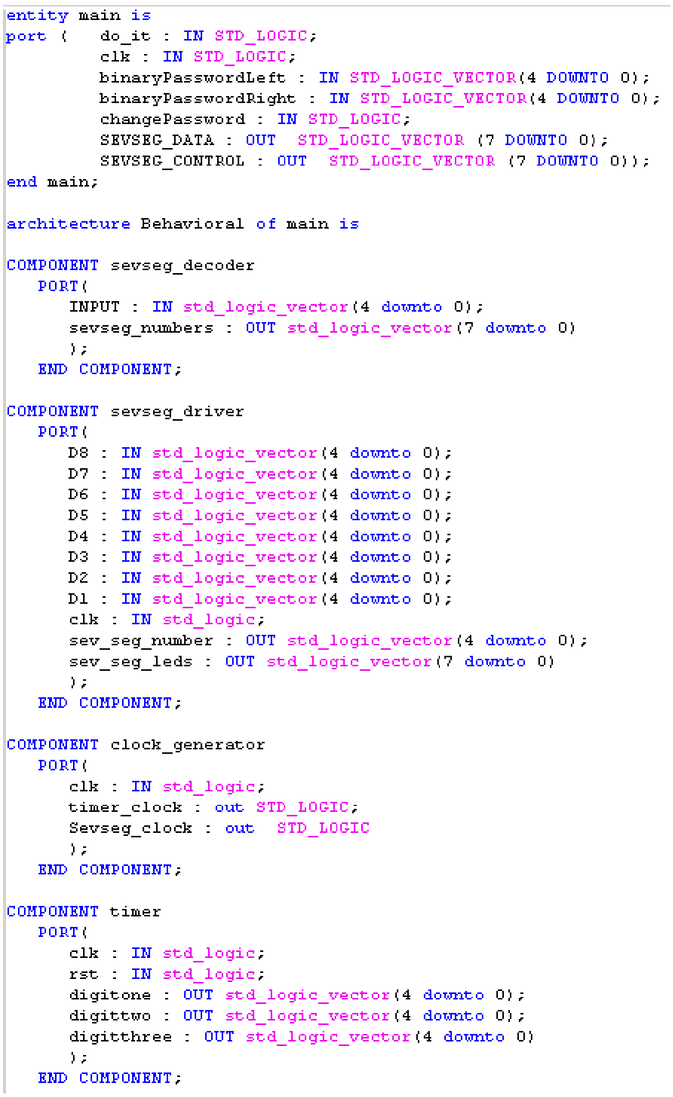
In this state password will not work anymore. The PUC code needed (mc = “10101 01100”). It remains in the same state until the correct puc code is entered. After right puc code is entered next state is will be same with previous state.

Change State

Change state helps to change current password with new one. While pushing changePassword button new password can be entered. Then automatically next state becomes unlocked state.
If changePassword is no pressed next state becomes locked without changing password.

2.3 Procedure:

Main

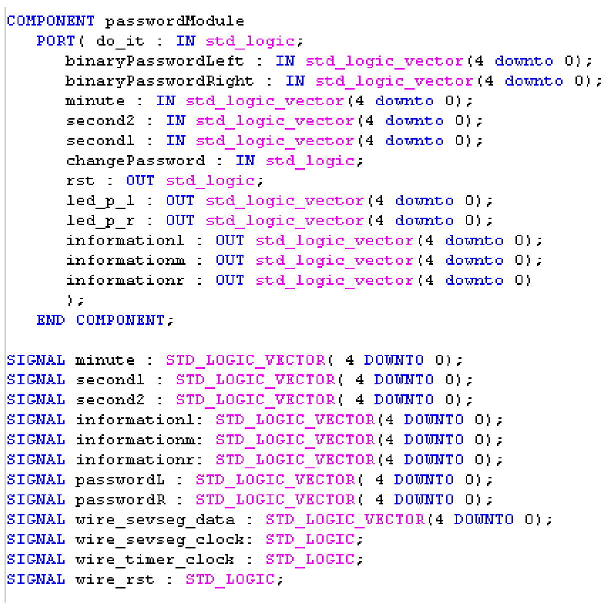


```
entity main is
port (    do_it : IN STD_LOGIC;
          clk : IN STD_LOGIC;
          binaryPasswordLeft : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
          binaryPasswordRight : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
          changePassword : IN STD_LOGIC;
          SEVSEG_DATA : OUT  STD_LOGIC_VECTOR (7 DOWNTO 0);
          SEVSEG_CONTROL : OUT  STD_LOGIC_VECTOR (7 DOWNTO 0));
end main;

architecture Behavioral of main is

COMPONENT sevseg_decoder
   PORT(
      INPUT : IN std_logic_vector(4 downto 0);
      sevseg_numbers : OUT std_logic_vector(7 downto 0)
      );
   END COMPONENT;

COMPONENT sevseg_driver
   PORT(
      D8 : IN std_logic_vector(4 downto 0);
      D7 : IN std_logic_vector(4 downto 0);
      D6 : IN std_logic_vector(4 downto 0);
      D5 : IN std_logic_vector(4 downto 0);
      D4 : IN std_logic_vector(4 downto 0);
      D3 : IN std_logic_vector(4 downto 0);
      D2 : IN std_logic_vector(4 downto 0);
      D1 : IN std_logic_vector(4 downto 0);
      clk : IN std_logic;
      sev_seg_number : OUT std_logic_vector(4 downto 0);
      sev_seg_leds : OUT std_logic_vector(7 downto 0)
      );
   END COMPONENT;

COMPONENT clock_generator
   PORT(
      clk : IN std_logic;
      timer_clock : out STD_LOGIC;
      Sevseg_clock : out  STD_LOGIC
      );
   END COMPONENT;

COMPONENT timer
   PORT(
      clk : IN std_logic;
      rst : IN std_logic;
      digitone : OUT std_logic_vector(4 downto 0);
      digittwo : OUT std_logic_vector(4 downto 0);
      digitthree : OUT std_logic_vector(4 downto 0)
      );
   END COMPONENT;
```

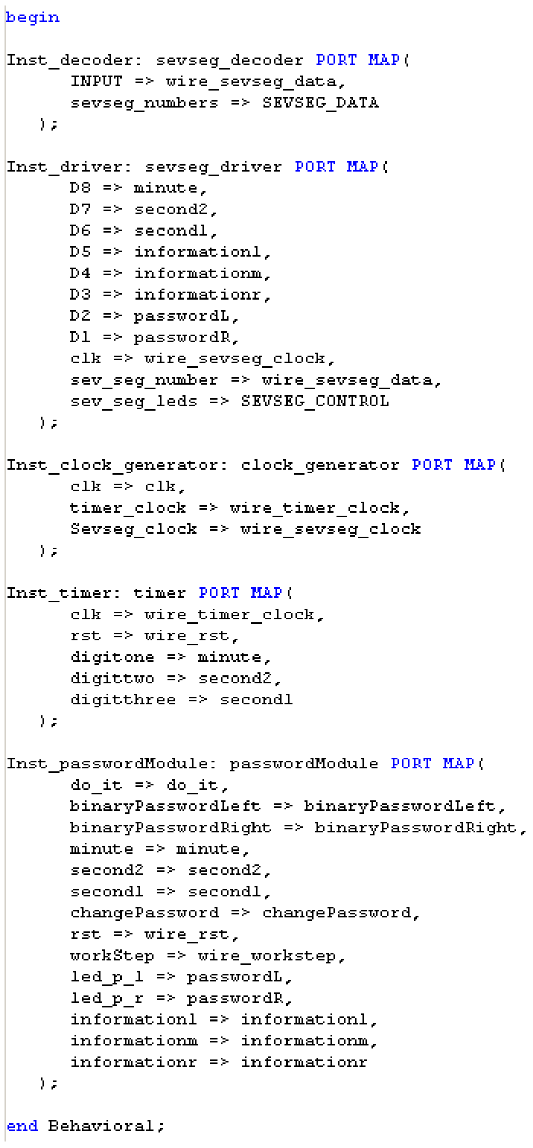
In Main module, five inputs were declared, and they are respectively do_it, clk, binarryPasswordLeft, binaryPasswordRight and changePassword. Except clk, four inputs’ values come from user, and depending on user inputs, they change the flown of the code. Clk input comes from FPGA board itself.

SEVSEG_DATA and SEVSEG_CONTROL are outputs and they send data which they receive from inputs to seven segments in FPGA board.

There are five port maps in using. They are sevseg_decoder, sevseg_driver, clock_generator, timer and passwordModule. They use the inputs data, produce outputs, and send them into each other



```vhdl
COMPONENT passwordModule
   PORT( do_it : IN std_logic;
      binaryPasswordLeft : IN std_logic_vector(4 downto 0);
      binaryPasswordRight : IN std_logic_vector(4 downto 0);
      minute : IN std_logic_vector(4 downto 0);
      second2 : IN std_logic_vector(4 downto 0);
      second1 : IN std_logic_vector(4 downto 0);
      changePassword : IN std_logic;
      rst : OUT std_logic;
      led_p_l : OUT std_logic_vector(4 downto 0);
      led_p_r : OUT std_logic_vector(4 downto 0);
      informationl : OUT std_logic_vector(4 downto 0);
      informationm : OUT std_logic_vector(4 downto 0);
      informationr : OUT std_logic_vector(4 downto 0)
      );
   END COMPONENT;

SIGNAL minute : STD_LOGIC_VECTOR( 4 DOWNTO 0);
SIGNAL second1 : STD_LOGIC_VECTOR( 4 DOWNTO 0);
SIGNAL second2 : STD_LOGIC_VECTOR( 4 DOWNTO 0);
SIGNAL informationl: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL informationm: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL informationr: STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL passwordL : STD_LOGIC_VECTOR( 4 DOWNTO 0);
SIGNAL passwordR : STD_LOGIC_VECTOR( 4 DOWNTO 0);
SIGNAL wire_sevseg_data : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL wire_sevseg_clock: STD_LOGIC;
SIGNAL wire_timer_clock : STD_LOGIC;
SIGNAL wire_rst : STD_LOGIC;
```



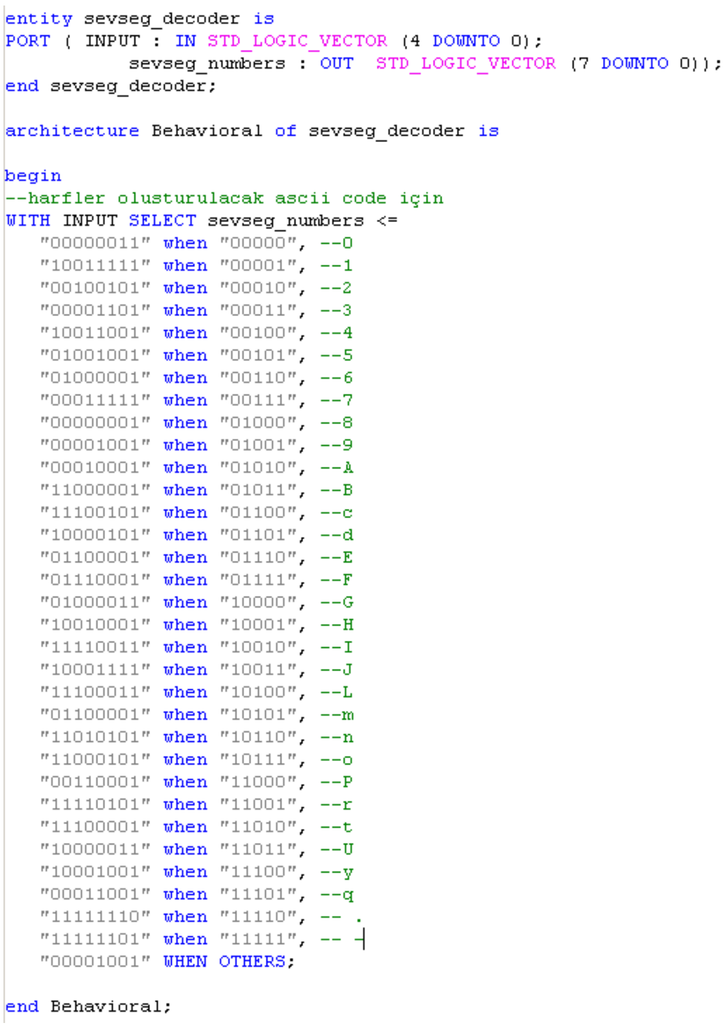
```vhdl
begin

Inst_decoder: sevseg_decoder PORT MAP(
      INPUT => wire_sevseg_data,
      sevseg_numbers => SEVSEG_DATA
   );

Inst_driver: sevseg_driver PORT MAP(
      D8 => minute,
      D7 => second2,
      D6 => second1,
      D5 => informationl,
      D4 => informationm,
      D3 => informationr,
      D2 => passwordL,
      D1 => passwordR,
      clk => wire_sevseg_clock,
      sev_seg_number => wire_sevseg_data,
      sev_seg_leds => SEVSEG_CONTROL
   );

Inst_clock_generator: clock_generator PORT MAP(
      clk => clk,
      timer_clock => wire_timer_clock,
      Sevseg_clock => wire_sevseg_clock
   );

Inst_timer: timer PORT MAP(
      clk => wire_timer_clock,
      rst => wire_rst,
      digitone => minute,
      digittwo => second2,
      digitthree => second1
   );

Inst_passwordModule: passwordModule PORT MAP(
      do_it => do_it,
      binaryPasswordLeft => binaryPasswordLeft,
      binaryPasswordRight => binaryPasswordRight,
      minute => minute,
      second2 => second2,
      second1 => second1,
      changePassword => changePassword,
      rst => wire_rst,
      workStep => wire_workstep,
      led_p_l => passwordL,
      led_p_r => passwordR,
      informationl => informationl,
      informationm => informationm,
      informationr => informationr
   );

end Behavioral;
```

Figure 3. VHDL code for main module

Decoder

The Decoder Module includes the numbers and the letters will be used for encrypting. The letters were built by using an ASCI systems which was created for this project. For numbers, binary values which represents their decimal values were used. For representing them, 5 to 8 decoder was used , that's why only 32 characters could be available to use. These 5 bits are converted to 8 bits seven segment.

For password, 2 bits were used in this project, that's why there are 32*32 different probabilities in password creating. This variety of password provides enough security for the lock.

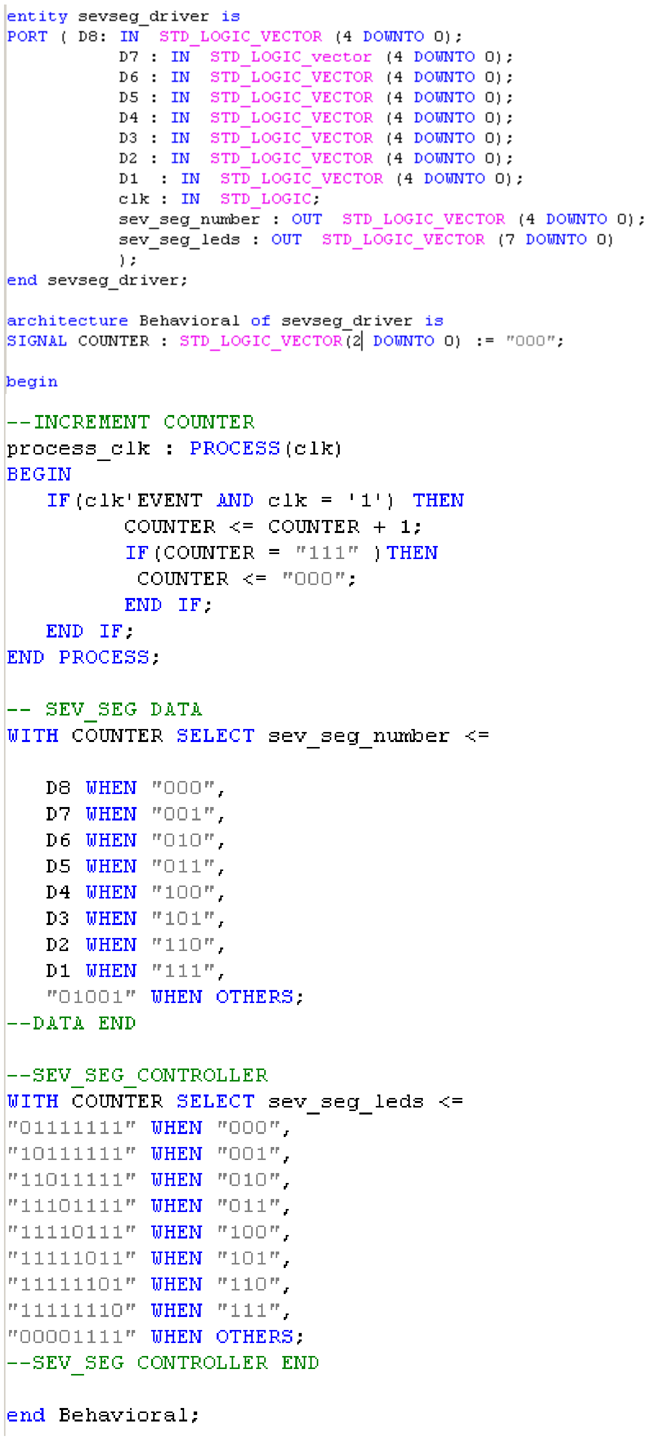


```
entity sevseg_decoder is
PORT ( INPUT : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
           sevseg_numbers : OUT  STD_LOGIC_VECTOR (7 DOWNTO 0));
end sevseg_decoder;

architecture Behavioral of sevseg_decoder is

begin
--harfler olusturulacak ascii code için
WITH INPUT SELECT sevseg_numbers <=
   "00000011" when "00000", --0
   "10011111" when "00001", --1
   "00100101" when "00010", --2
   "00001101" when "00011", --3
   "10011001" when "00100", --4
   "01001001" when "00101", --5
   "01000001" when "00110", --6
   "00011111" when "00111", --7
   "00000001" when "01000", --8
   "00001001" when "01001", --9
   "00010001" when "01010", --A
   "11000001" when "01011", --B
   "11100101" when "01100", --c
   "10000101" when "01101", --d
   "01100001" when "01110", --E
   "01110001" when "01111", --F
   "01000011" when "10000", --G
   "10010001" when "10001", --H
   "11110011" when "10010", --I
   "10001111" when "10011", --J
   "11100011" when "10100", --L
   "01100001" when "10101", --m
   "11010101" when "10110", --n
   "11000101" when "10111", --o
   "00110001" when "11000", --P
   "11110101" when "11001", --r
   "11100001" when "11010", --t
   "10000011" when "11011", --U
   "10001001" when "11100", --y
   "00011001" when "11101", --q
   "11111110" when "11110", -- .
   "11111101" when "11111", -- -
   "00001001" WHEN OTHERS;

end Behavioral;
```

Figure 4. VHDL code for sevseg_decoder module

Driver

Seven-segment is worked and controlled by driver module. This module decides which led is on. In this module numbers method is used to light Leds. From leftmost bit to right most bit in seven segment driver modules. That's why D(7:0) shows us which led will be next led to light up. And this module is controlled by clock which is came from Clock generator module(sevseg_clock).

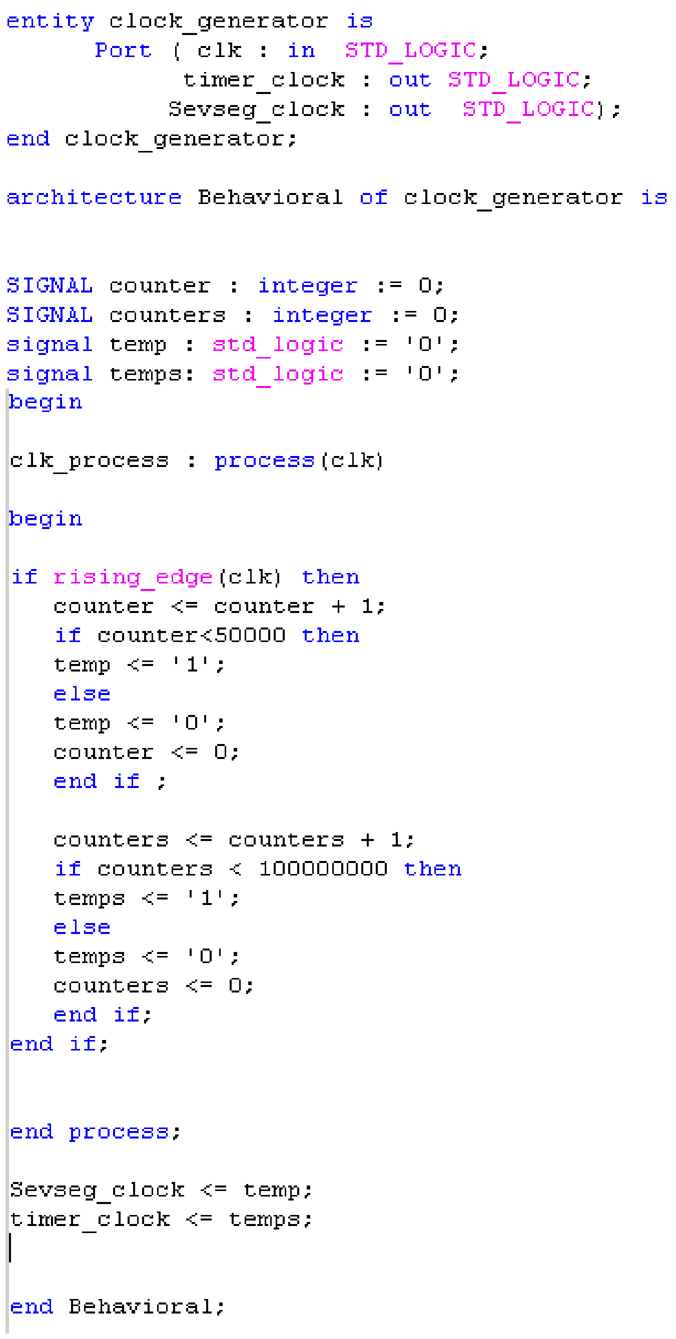


```
entity sevseg_driver is
PORT ( D8: IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          D7 : IN  STD_LOGIC_vector (4 DOWNTO 0);
          D6 : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          D5 : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          D4 : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          D3 : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          D2 : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          D1  : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          clk : IN  STD_LOGIC;
          sev_seg_number : OUT  STD_LOGIC_VECTOR (4 DOWNTO 0);
          sev_seg_leds : OUT  STD_LOGIC_VECTOR (7 DOWNTO 0)
          );
end sevseg_driver;

architecture Behavioral of sevseg_driver is
SIGNAL COUNTER : STD_LOGIC_VECTOR(2 DOWNTO 0)  := "000";

begin

--INCREMENT COUNTER
process_clk : PROCESS(clk)
BEGIN
   IF(clk'EVENT AND clk = '1') THEN
         COUNTER <= COUNTER + 1;
         IF(COUNTER = "111" )THEN
          COUNTER <= "000";
         END IF;
   END IF;
END PROCESS;

-- SEV_SEG DATA
WITH COUNTER SELECT sev_seg_number <=

   D8 WHEN "000",
   D7 WHEN "001",
   D6 WHEN "010",
   D5 WHEN "011",
   D4 WHEN "100",
   D3 WHEN "101",
   D2 WHEN "110",
   D1 WHEN "111",
   "01001" WHEN OTHERS;
--DATA END

--SEV_SEG_CONTROLLER
WITH COUNTER SELECT sev_seg_leds <=
"01111111" WHEN "000",
"10111111" WHEN "001",
"11011111" WHEN "010",
"11101111" WHEN "011",
"11110111" WHEN "100",
"11111011" WHEN "101",
"11111101" WHEN "110",
"11111110" WHEN "111",
"00001111" WHEN OTHERS;
--SEV_SEG CONTROLLER END

end Behavioral;
```

Figure 6. VHDL code for sevenseg_driver module

Clock Generator

Clock generator module generates counters for clock and seven-segment by using clk input. Clk input is inside of FPGA board and its value is 100 MHZ. This module decreases clock frequency around 1 Hz for our normal countdown which is Timer_clock. And there is one more output that take 100MHz clock and by dividing this clock frequency we got 200Hz Sevenseg_clock as output. Above 25Hz cannot seen by human eyes so this clock frequency can be chosen any number which is efficient. This 200Hz is our decide for 7-segment display.

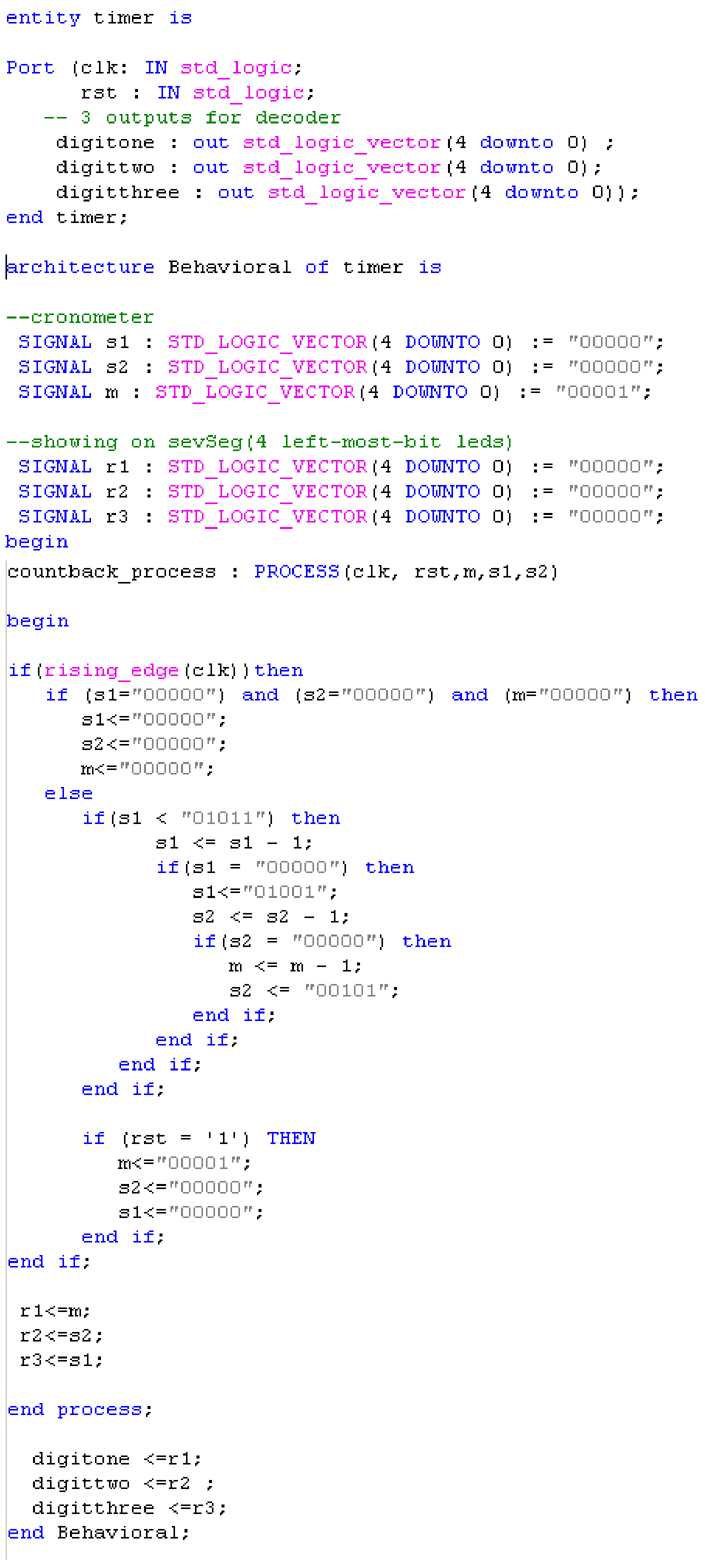


```
entity clock_generator is
      Port ( clk : in  STD_LOGIC;
            timer_clock : out STD_LOGIC;
           Sevseg_clock : out  STD_LOGIC);
end clock_generator;

architecture Behavioral of clock_generator is


SIGNAL counter : integer := 0;
SIGNAL counters : integer := 0;
signal temp : std_logic := '0';
signal temps: std_logic := '0';
begin

clk_process : process(clk)

begin

if rising_edge(clk) then
   counter <= counter + 1;
   if counter<50000 then
   temp <= '1';
   else
   temp <= '0';
   counter <= 0;
   end if ;

   counters <= counters + 1;
   if counters < 100000000 then
   temps <= '1';
   else
   temps <= '0';
   counters <= 0;
   end if;
end if;


end process;

Sevseg_clock <= temp;
timer_clock <= temps;


end Behavioral;
```

Figure 7. VHDL code for clock divider(generator) module

Timer

Timer module creates a clock which goes to 0 from 60 seconds. Important point of this clock is that it can be reset depends on the changes in password module. This module determines of the rest time when user can enter a password.
Digit one, digit two and digit three outputs send information to password module. If they become 0, then entering the password doesn't work at all in password module.



```
entity timer is

Port (clk: IN std_logic;
      rst : IN std_logic;
   -- 3 outputs for decoder
    digitone : out std_logic_vector(4 downto 0) ;
    digittwo : out std_logic_vector(4 downto 0);
    digitthree : out std_logic_vector(4 downto 0));
end timer;

architecture Behavioral of timer is

--cronometer
 SIGNAL s1 : STD_LOGIC_VECTOR(4 DOWNTO 0)  := "00000";
 SIGNAL s2 : STD_LOGIC_VECTOR(4 DOWNTO 0)  := "00000";
 SIGNAL m : STD_LOGIC_VECTOR(4 DOWNTO 0)  := "00001";

--showing on sevSeg(4 left-most-bit leds)
 SIGNAL r1 : STD_LOGIC_VECTOR(4 DOWNTO 0)  := "00000";
 SIGNAL r2 : STD_LOGIC_VECTOR(4 DOWNTO 0)  := "00000";
 SIGNAL r3 : STD_LOGIC_VECTOR(4 DOWNTO 0)  := "00000";
begin
countback_process : PROCESS(clk, rst,m,s1,s2)

begin

if(rising_edge(clk))then
   if (s1="00000") and (s2="00000") and (m="00000") then
      s1<="00000";
      s2<="00000";
      m<="00000";
   else
      if(s1 < "01011") then
            s1 <= s1 - 1;
            if(s1 = "00000") then
               s1<="01001";
               s2 <= s2 - 1;
               if(s2 = "00000") then
                  m <= m - 1;
                  s2 <= "00101";
               end if;
            end if;
         end if;
      end if;

      if (rst = '1') THEN
         m<="00001";
         s2<="00000";
         s1<="00000";
      end if;
end if;

 r1<=m;
 r2<=s2;
 r3<=s1;

end process;

  digitone <=r1;
  digittwo <=r2 ;
  digitthree <=r3;
end Behavioral;
```

Figure 8. VHDL code for timer module

Password Module

All significant processes were handled in this module. Comparing the correct passwords with user input, unlocking and locking the lock, changing the current password after being unlocked it are controlled in password module.

Defined signals current Password Left and current Password Right are stored the correct password bit by bit. Left one stores the left most bit, the right one stores the right most bit.
Puk Code Left and puk Code Right signals are used for the situations which is that entering the password wrong three times or not entering the correct pin in a minute. If user enters the correct pin when time out or trying right finishes, he cannot unlock the lock. The correct puk code must be entered to get this right again.

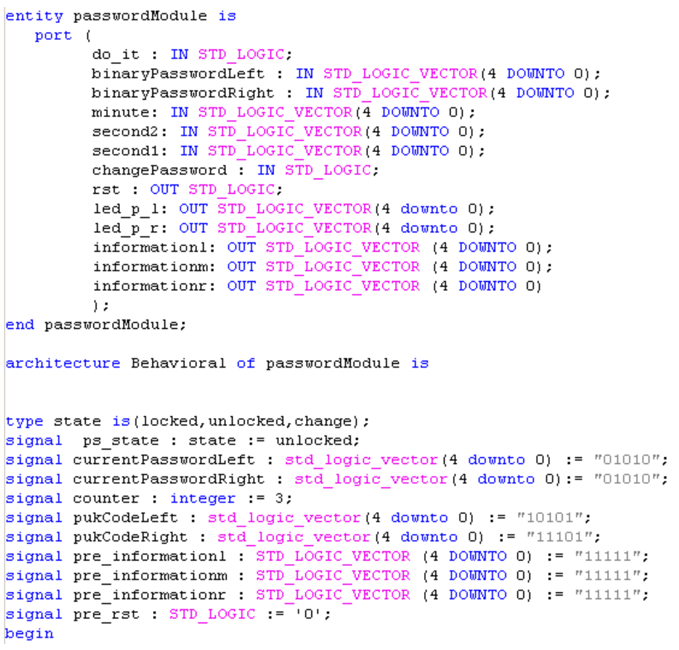
Information signals are used for giving knowledge to user about the current situation of the lock. If it is locked, off info is given; if it is unlocked, opn info is given; if it is asked to change the password, chn info is given.

Counter is used for showing and checking that how many times user can enter password. It is started by 3 and goes to 0 as user keys a wrong password. After it is 0, time is reset and puk code is expected.

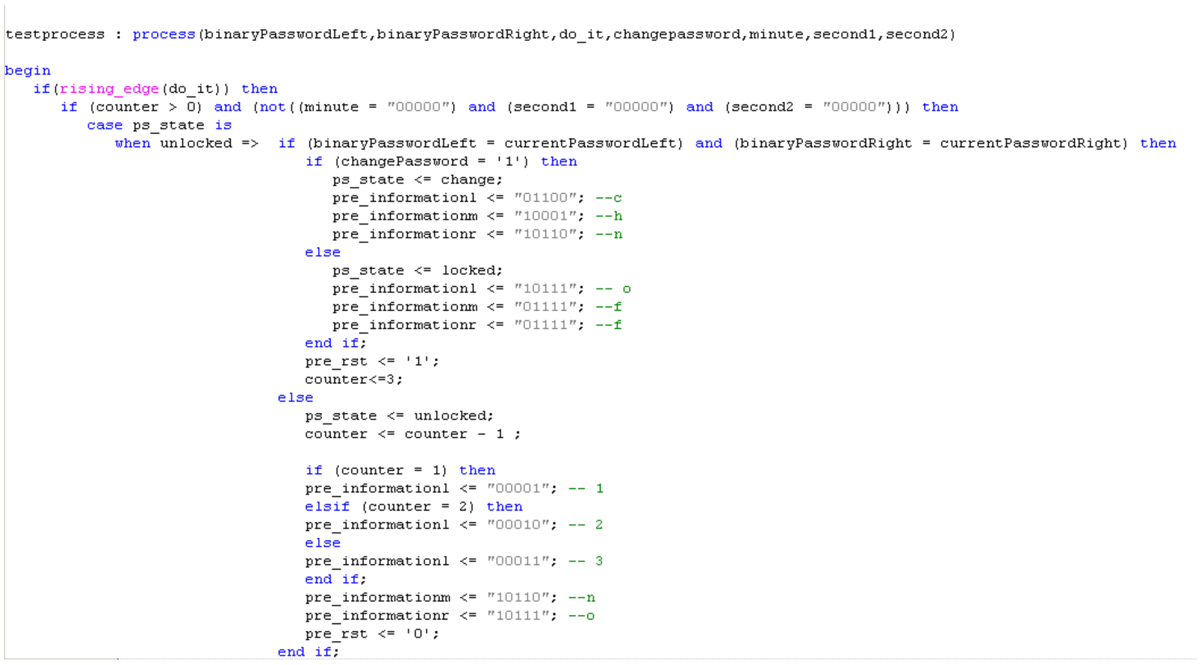
Ps_state has three different values which are locked, unlocked and change. Depends on the next situation of the lock, it changes between these statements. Also, the current statement is determined by Ps_state and controls are made by checking this Ps_state signal.

Minute, second1 and second2 inputs are used for controlling whether time is up or not.
changePassword input is used for checking if user want to change his password. If it becomes 1 and also the current state of lock is open, then changing password statement opens for user, and user can change his password while pushing the change password pin.
Led_p_l and led_p_r outputs make the password entered by user shown on the seven segments.

rst output used in timer to make the clock 0 gets its value in this module. If time is up or attempt of password entering is out then rst will be 1, and clock is prepared to start again.

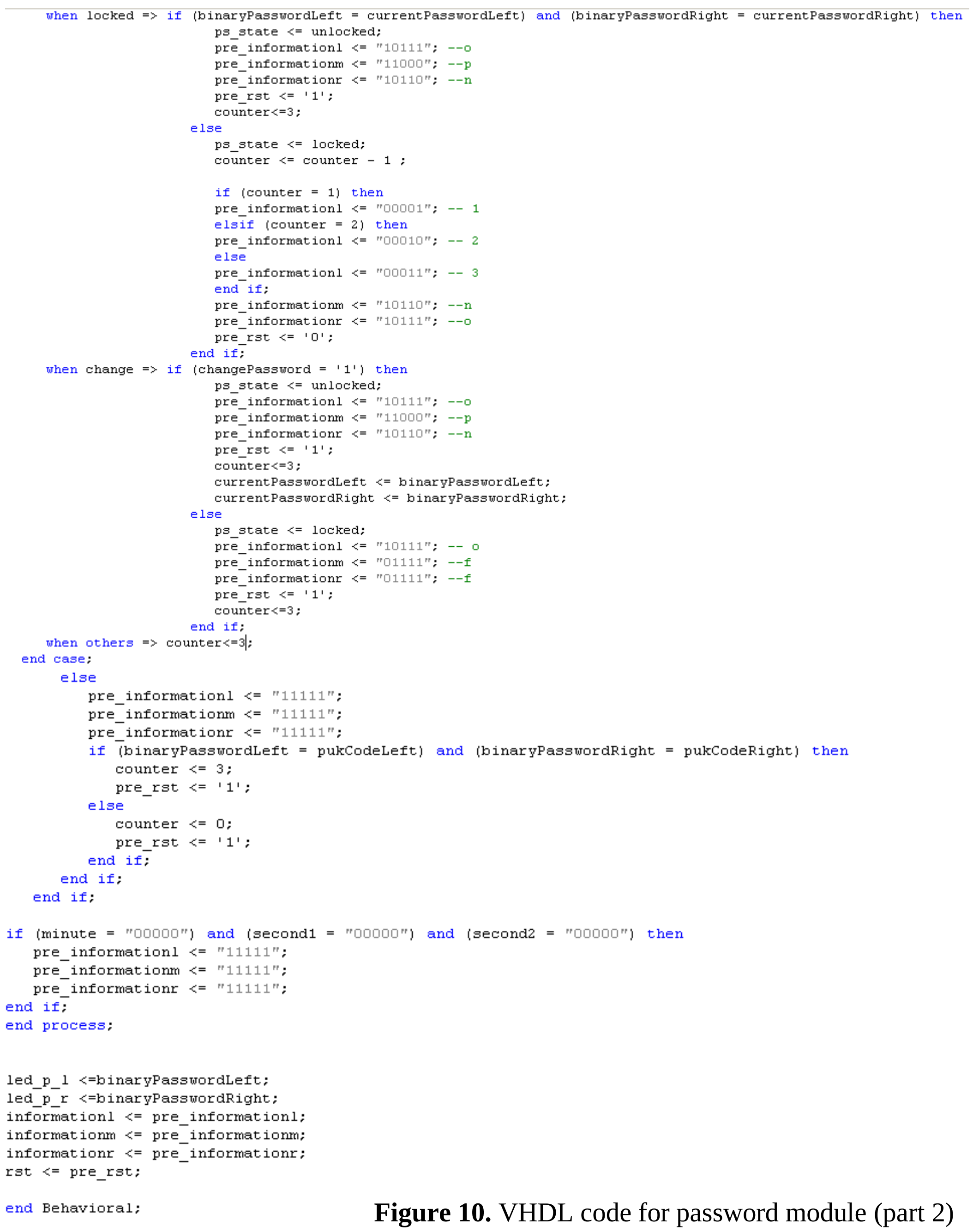


```
entity passwordModule is
   port (
         do_it : IN STD_LOGIC;
         binaryPasswordLeft : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
         binaryPasswordRight : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
         minute: IN STD_LOGIC_VECTOR(4 DOWNTO 0);
         second2: IN STD_LOGIC_VECTOR(4 DOWNTO 0);
         second1: IN STD_LOGIC_VECTOR(4 DOWNTO 0);
         changePassword : IN STD_LOGIC;
         rst : OUT STD_LOGIC;
         led_p_l: OUT STD_LOGIC_VECTOR(4 downto 0);
         led_p_r: OUT STD_LOGIC_VECTOR(4 downto 0);
         informationl: OUT STD_LOGIC_VECTOR (4 DOWNTO 0);
         informationm: OUT STD_LOGIC_VECTOR (4 DOWNTO 0);
         informationr: OUT STD_LOGIC_VECTOR (4 DOWNTO 0)
         );
end passwordModule;

architecture Behavioral of passwordModule is


type state is(locked,unlocked,change);
signal  ps_state : state := unlocked;
signal currentPasswordLeft : std_logic_vector(4 downto 0)  := "01010";
signal currentPasswordRight : std_logic_vector(4 downto 0):= "01010";
signal counter : integer := 3;
signal pukCodeLeft : std_logic_vector(4 downto 0)  := "10101";
signal pukCodeRight : std_logic_vector(4 downto 0)  := "11101";
signal pre_informationl : STD_LOGIC_VECTOR (4 DOWNTO 0)  := "11111";
signal pre_informationm : STD_LOGIC_VECTOR (4 DOWNTO 0)  := "11111";
signal pre_informationr : STD_LOGIC_VECTOR (4 DOWNTO 0)  := "11111";
signal pre_rst : STD_LOGIC := '0';
begin
```



```
testprocess : process(binaryPasswordLeft,binaryPasswordRight,do_it,changepassword,minute,second1,second2)

begin
   if(rising_edge(do_it)) then
      if (counter > 0) and (not((minute = "00000") and (second1 = "00000") and (second2 = "00000"))) then
         case ps_state is
            when unlocked =>  if (binaryPasswordLeft = currentPasswordLeft) and (binaryPasswordRight = currentPasswordRight) then
                                 if (changePassword = '1') then
                                    ps_state <= change;
                                    pre_informationl <= "01100"; --c
                                    pre_informationm <= "10001"; --h
                                    pre_informationr <= "10110"; --n
                                 else
                                    ps_state <= locked;
                                    pre_informationl <= "10111"; -- o
                                    pre_informationm <= "01111"; --f
                                    pre_informationr <= "01111"; --f
                                 end if;
                                 pre_rst <= '1';
                                 counter<=3;
                              else
                                 ps_state <= unlocked;
                                 counter <= counter - 1 ;

                                 if (counter = 1) then
                                 pre_informationl <= "00001"; -- 1
                                 elsif (counter = 2) then
                                 pre_informationl <= "00010"; -- 2
                                 else
                                 pre_informationl <= "00011"; -- 3
                                 end if;
                                 pre_informationm <= "10110"; --n
                                 pre_informationr <= "10111"; --o
                                 pre_rst <= '0';
                              end if;
```

Figure 9. VHLD code for password module (part 1)

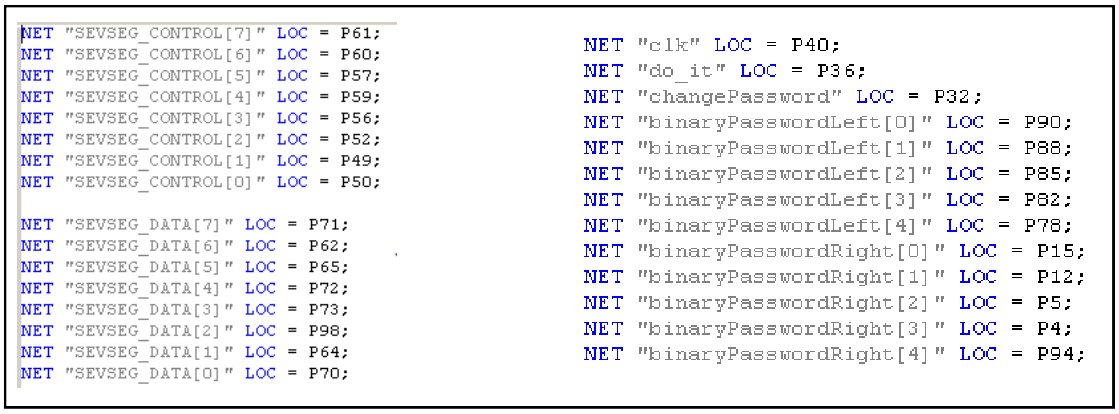


```
     when locked => if (binaryPasswordLeft = currentPasswordLeft) and (binaryPasswordRight = currentPasswordRight) then
                           ps_state <= unlocked;
                           pre_informationl <= "10111"; --o
                           pre_informationm <= "11000"; --p
                           pre_informationr <= "10110"; --n
                           pre_rst <= '1';
                           counter<=3;
                        else
                           ps_state <= locked;
                           counter <= counter - 1 ;

                           if (counter = 1) then
                           pre_informationl <= "00001"; -- 1
                           elsif (counter = 2) then
                           pre_informationl <= "00010"; -- 2
                           else
                           pre_informationl <= "00011"; -- 3
                           end if;
                           pre_informationm <= "10110"; --n
                           pre_informationr <= "10111"; --o
                           pre_rst <= '0';
                        end if;
     when change => if (changePassword = '1') then
                           ps_state <= unlocked;
                           pre_informationl <= "10111"; --o
                           pre_informationm <= "11000"; --p
                           pre_informationr <= "10110"; --n
                           pre_rst <= '1';
                           counter<=3;
                           currentPasswordLeft <= binaryPasswordLeft;
                           currentPasswordRight <= binaryPasswordRight;
                        else
                           ps_state <= locked;
                           pre_informationl <= "10111"; -- o
                           pre_informationm <= "01111"; --f
                           pre_informationr <= "01111"; --f
                           pre_rst <= '1';
                           counter<=3;
                        end if;
     when others => counter<=3;
  end case;
      else
         pre_informationl <= "11111";
         pre_informationm <= "11111";
         pre_informationr <= "11111";
         if (binaryPasswordLeft = pukCodeLeft) and (binaryPasswordRight = pukCodeRight) then
            counter <= 3;
            pre_rst <= '1';
         else
            counter <= 0;
            pre_rst <= '1';
         end if;
      end if;
   end if;

if (minute = "00000") and (second1 = "00000") and (second2 = "00000") then
   pre_informationl <= "11111";
   pre_informationm <= "11111";
   pre_informationr <= "11111";
end if;
end process;


led_p_l <=binaryPasswordLeft;
led_p_r <=binaryPasswordRight;
informationl <= pre_informationl;
informationm <= pre_informationm;
informationr <= pre_informationr;
rst <= pre_rst;

end Behavioral;
```

Figure 10. VHDL code for password module (part 2)

Implementation Files

The Ports are used in this project as below



```
NET "SEVSEG_CONTROL[7]" LOC = P61;
NET "SEVSEG_CONTROL[6]" LOC = P60;
NET "SEVSEG_CONTROL[5]" LOC = P57;
NET "SEVSEG_CONTROL[4]" LOC = P59;
NET "SEVSEG_CONTROL[3]" LOC = P56;
NET "SEVSEG_CONTROL[2]" LOC = P52;
NET "SEVSEG_CONTROL[1]" LOC = P49;
NET "SEVSEG_CONTROL[0]" LOC = P50;

NET "SEVSEG_DATA[7]" LOC = P71;
NET "SEVSEG_DATA[6]" LOC = P62;
NET "SEVSEG_DATA[5]" LOC = P65;
NET "SEVSEG_DATA[4]" LOC = P72;
NET "SEVSEG_DATA[3]" LOC = P73;
NET "SEVSEG_DATA[2]" LOC = P98;
NET "SEVSEG_DATA[1]" LOC = P64;
NET "SEVSEG_DATA[0]" LOC = P70;
```

```
NET "clk" LOC = P40;
NET "do_it" LOC = P36;
NET "changePassword" LOC = P32;
NET "binaryPasswordLeft[0]" LOC = P90;
NET "binaryPasswordLeft[1]" LOC = P88;
NET "binaryPasswordLeft[2]" LOC = P85;
NET "binaryPasswordLeft[3]" LOC = P82;
NET "binaryPasswordLeft[4]" LOC = P78;
NET "binaryPasswordRight[0]" LOC = P15;
NET "binaryPasswordRight[1]" LOC = P12;
NET "binaryPasswordRight[2]" LOC = P5;
NET "binaryPasswordRight[3]" LOC = P4;
NET "binaryPasswordRight[4]" LOC = P94;
```

Figure 11. UCF file for pin locations

RTL Schematics

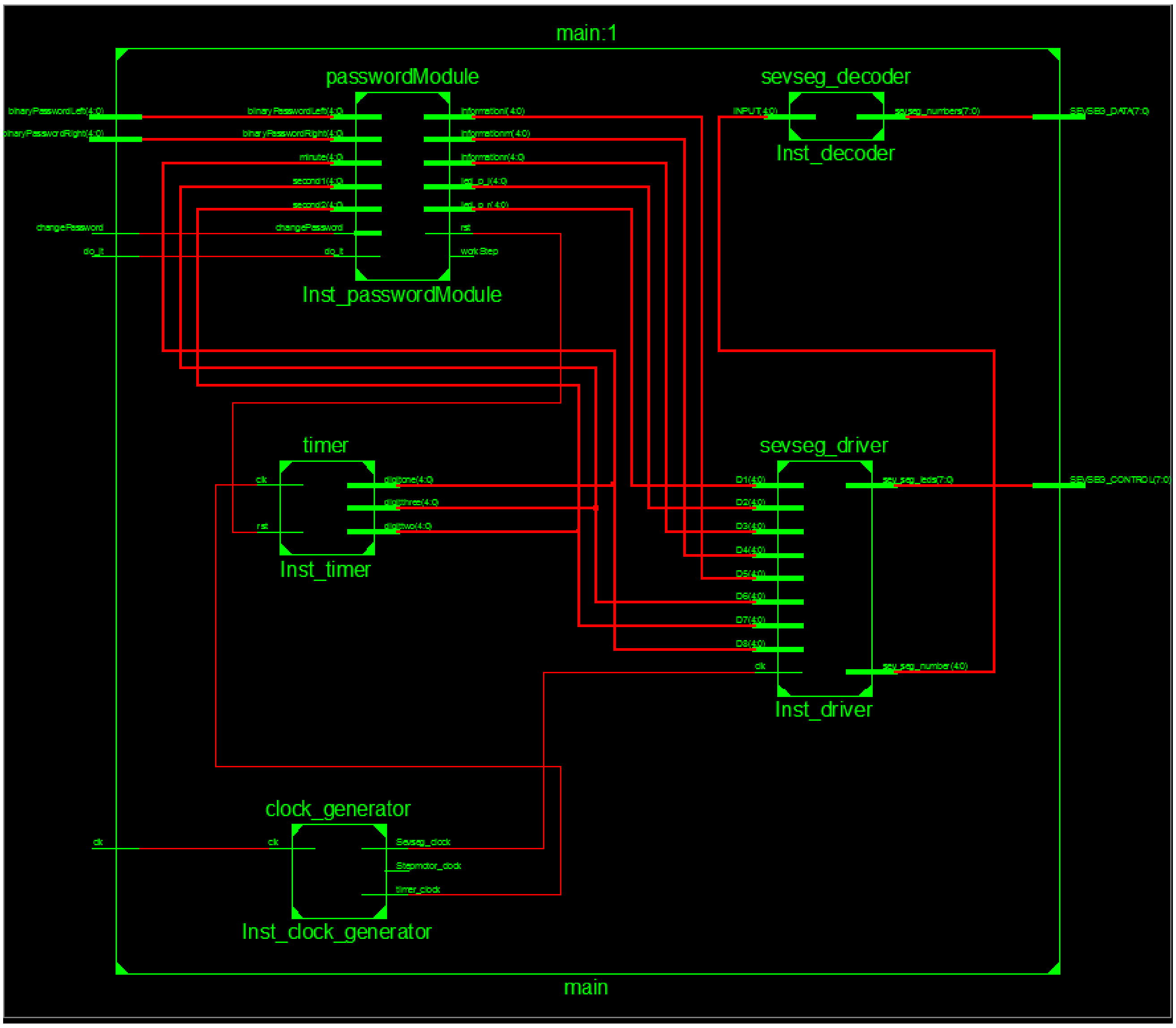


Figure 12. RTL schematics of project

What are the similarities with the labs?

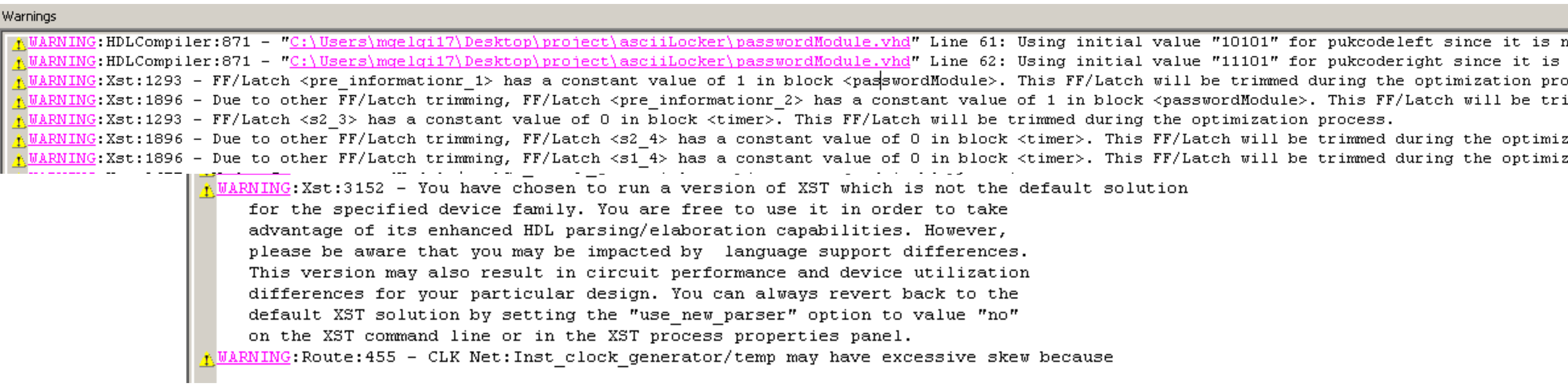
- Taking input from the switches
- Moving to next state according to button pressed
- Using LEDS to display input entered through switches
- Using Clock generator with different frequencies
- Keep previous states in memory

What are the differences we added?

- Creating random number in a specific range using linear feedback
- Evaluating the input entered by the user
- Adding a restart button, returning to initial state

3. Problems encountered, errors and warnings resolved

In this project we got two main warning during project. One of them is about clock warning due to lots of different clock at the same time. The second one is more significant because for this problem we couldn’t synthesize code. By some research we found that changing some properties inside of project tab. By this way our code is working however has warning which can be ignored.



```
Warnings
WARNING:HDLCompiler:871 - "C:\Users\mgelgi17\Desktop\project\asciiLocker\passwordModule.vhd" Line 61: Using initial value "10101" for pukcodeleft since it is n
WARNING:HDLCompiler:871 - "C:\Users\mgelgi17\Desktop\project\asciiLocker\passwordModule.vhd" Line 62: Using initial value "11101" for pukcoderight since it is
WARNING:Xst:1293 - FF/Latch <pre_informationr_1> has a constant value of 1 in block <passwordModule>. This FF/Latch will be trimmed during the optimization pro
WARNING:Xst:1896 - Due to other FF/Latch trimming, FF/Latch <pre_informationr_2> has a constant value of 1 in block <passwordModule>. This FF/Latch will be tri
WARNING:Xst:1293 - FF/Latch <s2_3> has a constant value of 0 in block <timer>. This FF/Latch will be trimmed during the optimization process.
WARNING:Xst:1896 - Due to other FF/Latch trimming, FF/Latch <s2_4> has a constant value of 0 in block <timer>. This FF/Latch will be trimmed during the optimiz
WARNING:Xst:1896 - Due to other FF/Latch trimming, FF/Latch <s1_4> has a constant value of 0 in block <timer>. This FF/Latch will be trimmed during the optimiz
```

```
WARNING:Xst:3152 - You have chosen to run a version of XST which is not the default solution
   for the specified device family. You are free to use it in order to take
   advantage of its enhanced HDL parsing/elaboration capabilities. However,
   please be aware that you may be impacted by  language support differences.
   This version may also result in circuit performance and device utilization
   differences for your particular design. You can always revert back to the
   default XST solution by setting the "use_new_parser" option to value "no"
   on the XST command line or in the XST process properties panel.
WARNING:Route:455 - CLK Net:Inst_clock_generator/temp may have excessive skew because
```

Figure 13. warnings encountered

4. Conclusion

Main objective in this project is that creating a lock system which allows user to change password. In this project we could get familiar using states, clock and numeric libraries. First we coded simple ASCII code then by using this code we implemented passwords on FPGA board. This allows us lots of different passwords possibilities. Throughout the project we experienced the design and implementation of a project from scratch, tested our ideas, modified them according to our capabilities and knowledge. After performing this project, we both had an idea about how to perform system design and test.

In overall, it was an educative experience for us. We reviewed the entire course material while testing and improving our project. Also, it was so useful for us to get familiarize using Xilinx.

5. References

1. Marro, Ciletti (2018). Digital Design. New York, NY; Pearson
2. https://electronics.stackexchange.com/questions/204540/verilog-slow-clock-geneator-module-1hz-from-50mhz
3. https://github.com/arash-codes/Elec204-labs/
4. LAB-4, ELEC-204 Digital Design Course
5. https://www.idt.com/us/en/products/clocks-timing/clock-distribution/clock-dividers
6. https://www.electronics-tutorials.ws/counter/count_1.html
7. https://www.nandland.com/vhdl/examples/example-case-statement.html