ELEC 204 Digital Design Project Report

SAFE LOCK

01/07/21

İrem Karaca - 71879
Berker Berk - 64913

1. Introduction and Objectives

The aim of this project is to design a password system that is representing a safe lock while implementing concepts that we have learned in ELEC204 class such as binary to hexadecimal conversions, combinational and sequential circuits. The Safe Lock Project gets a password in hexadecimal from the user at the beginning and then saves the password. After that, users are able to try to open the safe lock by entering a password using switches and buttons. If users enter the password correctly, the implementation corresponds to “open”; if they enter the password incorrectly, the implementation corresponds to “fail” on the seven-segment display. Furthermore, if users enter the password incorrectly 3 times in total, the system is locked, and “lock” is written on the 7-segment display. We also created a master key that can open every lock.

2. Methods

There are 6 inputs in the SafeLockCode:

- **MCLK**: 1-bit standard logic for the master clock
- **B0**: 1-bit standard logic that stands for the enter button
- **B1**: 1-bit standard logic that stands for the submit button
- **reset**: 1-bit standard logic that stands for the reset button
- **inputNum**: 4-bit standard logic vector that stands for the number entered by the user
- **digit**: 2-bit standard logic vector that stands for the selected digit of the number

There are 8 inouts in the SafeLockCode:

- **password0**: 4-bit standard logic vector that keeps the left-most digit of the password
- **password1**: 4-bit standard logic vector that keeps the second left-most digit of the password
- **password2**: 4-bit standard logic vector that keeps the third left-most digit of the password
- **password3**: 4-bit standard logic vector that keeps the right-most digit of the password
- **try0**: 4-bit standard logic vector that keeps the left-most digit of the number entered
- **try1**: 4-bit standard logic vector that keeps the second left-most digit of the number entered
- **try2**: 4-bit standard logic vector that keeps the third left-most digit of the number entered
- **try3**: 4-bit standard logic vector that keeps the right-most digit of the number entered

There are 2 outputs in the SafeLockCode:

- **SEVSEG_DATA**: 7-bit standard logic vector for the implementation of the 7-segment display
- **SEVSEG_CONTROL**: 8-bit standard logic vector for the implementation of the 7-segment display

There are 8 intermediate signals in the SafeLockCode:

- **CLK_DIV**: 1-bit standard logic for the divided clock
- exist: 1-bit standard logic for checking a password exist in the system or not
- **out0**: 5-bit standard logic vector that keeps the fourth right-most digit displaying on the 7-segment display.
- **out1**: 5-bit standard logic vector that keeps the third right-most digit displaying on the 7-segment display.
- **out2**: 5-bit standard logic vector that keeps the second right-most digit displaying on the 7-segment display.
- **out3**: 5-bit standard logic vector that keeps the right-most digit displaying on the 7-segment display.
- **tcount**: 2-bit standard logic vector that keeps the number of the failed attempts
- **newTry**: 1-bit standard logic that becomes 1 when the user tries to make another attempt after “open” or “fail” is displayed on the 7-segment display.

The SafeLockCode consists of 2 processes. The first process is the clock divider that aims to change the frequency of the master clock and make the clock suitable for the other process. The second process is the main process that takes inputs and generates outputs. At the beginning of the main process, the code is checking whether the “reset" signal is 1 or not. If the reset signal is 1, all variables are set to their initial value which is 0. After that, the code checks whether there is a password saved in the system or not using the “exist” signal. If there is no password in the system -when the user is using the system for the first time or the system is reset, the user needs to enter a password. To enter a password, switches and buttons are used. The right-most 2 switches are used to determine which digit is entered at that time.

00	Left-most digit
01	Second left-most digit
10	Second right-most digit
11	Right-most digit

Table 1: Switch configurations to select digits on 7-segment display

The left-most 4 digits are used to enter a 4-bit binary number that corresponds to a digit of the password. Since users are able to enter 0 to 15 for the input number, the password will be in hexadecimal.

0001	1
1010	A
0110	6
1111	F

Table 2: Sample switch configurations to enter a number in binary and corresponding values in hexadecimal

After every bit, the user needs to click the enter button to save that bit. After all bits are entered, the user needs to click the submit button so that the code saves that password. Password is saved digit by digit in password0, password1, password2, and password3 vectors. While saving the password, out0, out1, out2, out3 vectors are also changed with the corresponding values.

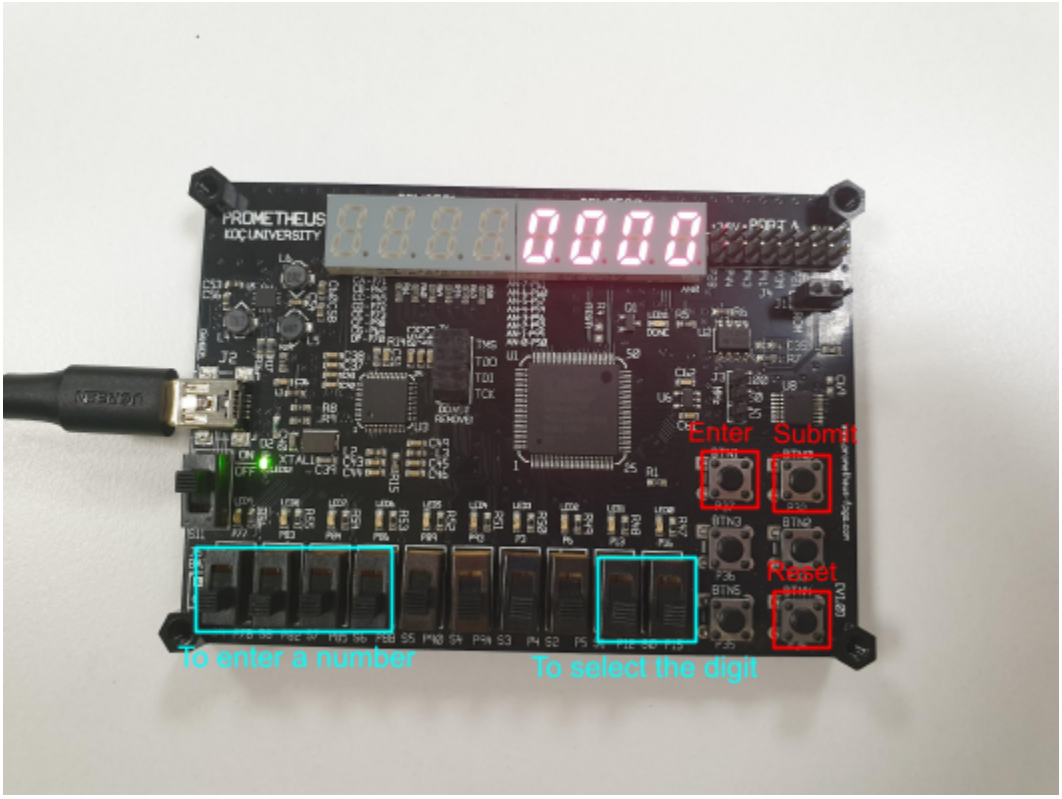


Figure 1: Switches and buttons used on the FPGA board

After the password is saved, users are able to try passwords to open the lock. Making a password attempt is the same as defining an initial password. The only difference is that, when the user clicks the submit button, the code checks whether the input number is the same as the

password saved or the master key. If it is, “open” is printed on the 7-segment display; otherwise “fail”. If the attempts are failed 3 times in total -when the tcount is 3, the system will be locked. It means that users will not be able to make a password attempt after that time. The system needs to be reset to use it again with another password.

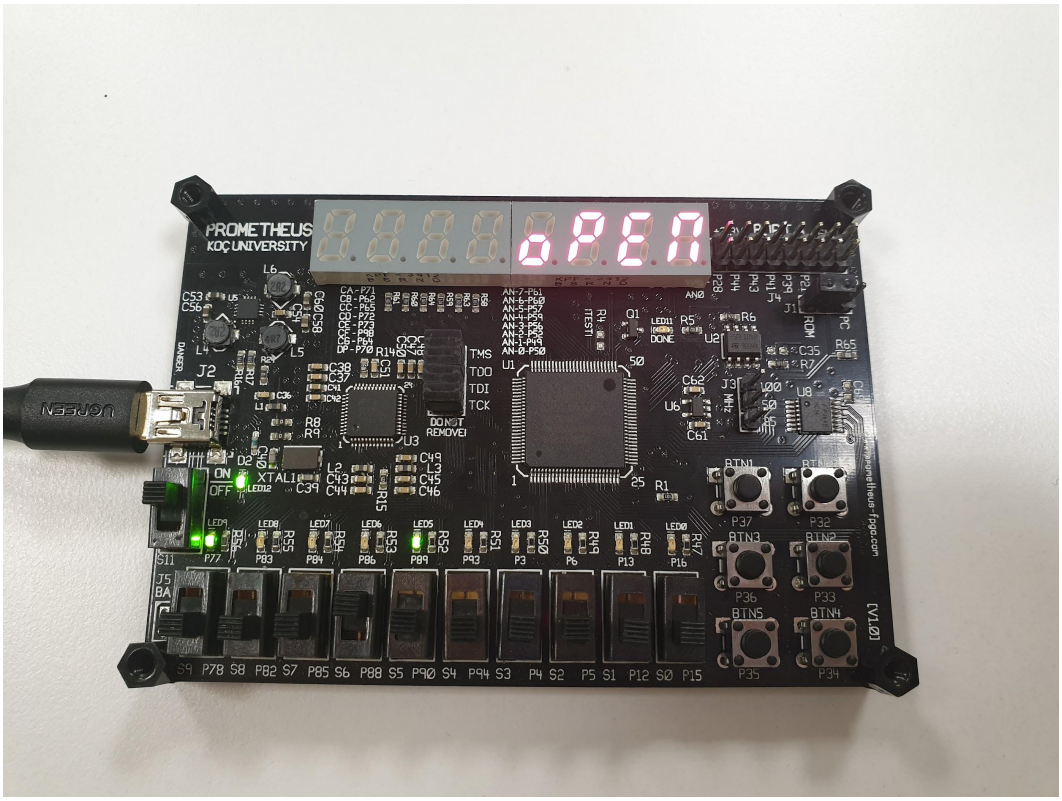


Figure 2: 7-segment display when the user enters password correctly

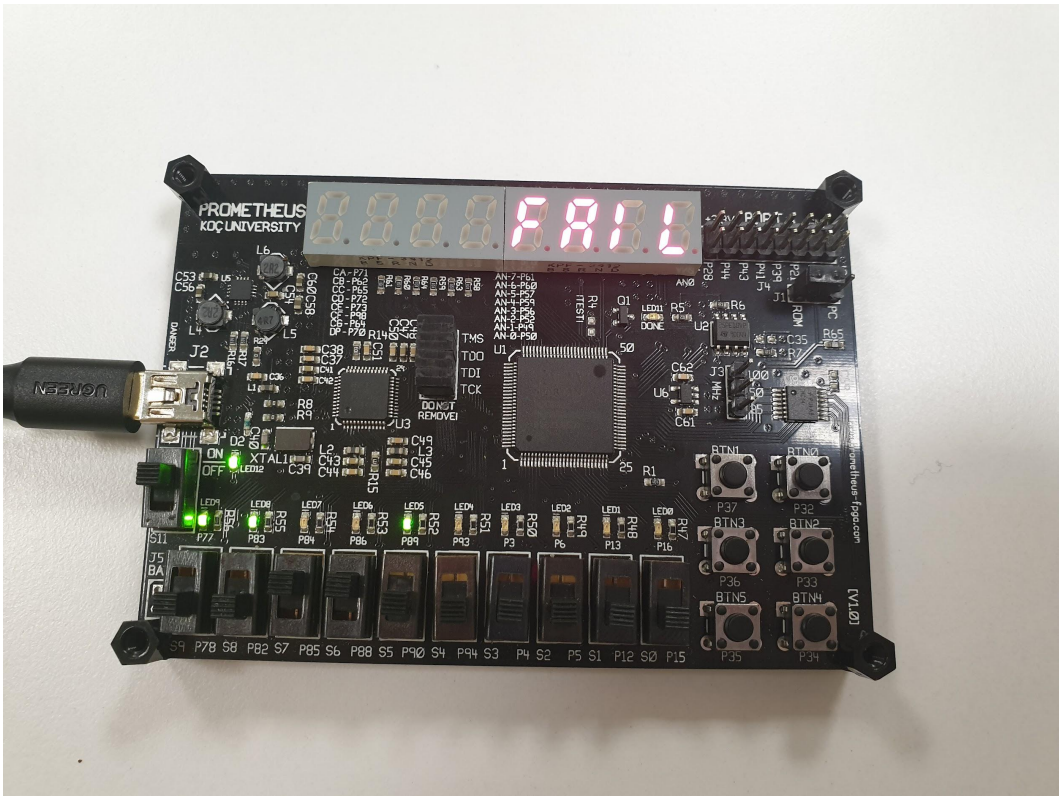


Figure 3: 7-segment display when the user enters password incorrectly

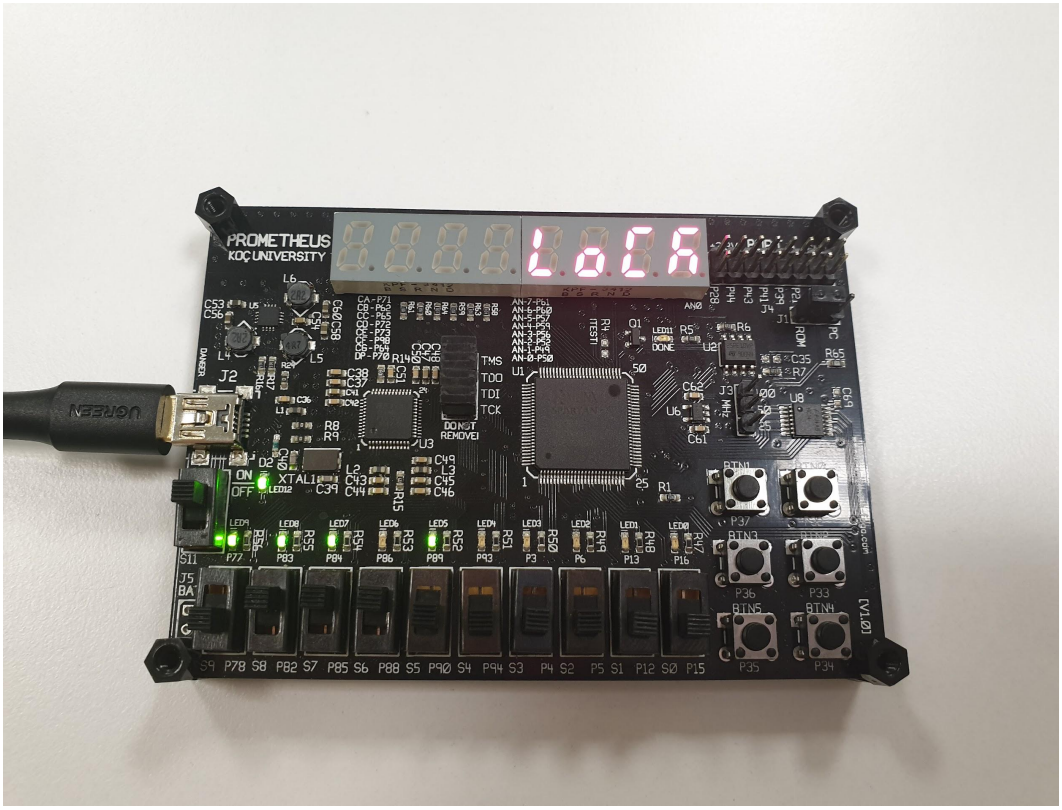


Figure 4: 7-segment display when the user enters password incorrectly 3 times.

There is SSLIB component in SafeLockCode to use the 7-segment display. SSLIB consists of SEVSEG_DRIVER, SEVSEG_DECODER, and HUDREDHZ_CLOCK_GENERATOR.

SEVSEG_DRIVER is used to display the password digits on separate digits on 7-segment display.

SEVSEG_DECODER gets 5-bit inputs and decodes them into 7-bit outputs that correspond to the letters in “open”, “fail”, “lock”, numbers from 0 to 9, and letters from A to F.

HUDREDHZ_CLOCK_GENERATOR gets the master clock as an input and generates another clock for the 7-segment display. The working principle of the new clock is that the counter is increasing at each rising edge of the master clock up to 250000. When the counter is between 0 and 125000, the new clock becomes 1; when the counter is between 125000 and 250000, the new clock becomes 0.

3. Problems encountered, errors and warnings resolved

Although we wrote the simulation code properly, the simulation did not work at the beginning. To solve this problem, we decreased the N value from 50*10**6 to 2 in SafeLockCode before we made the simulation. We also faced another problem with the FPGA board. When we click buttons, some leds become on randomly even though our code has nothing about leds. When we asked the instructor about this issue, he told us that it will not be a problem so we did not try to solve that.

4. Conclusion

In this project, we designed and implemented a password system that represents a safe lock. We used switches to take 4-bit binary values from the user, then we convert them into hexadecimal values and display them on a 7-segment display. We used buttons to save and reset the password. In order to do that, we used our knowledge about binary to hexadecimal conversions, decoders, multiplexers, and sequential circuits. We learned to display different outputs for different situations on a 7-segment display. Also, we gained more experience with VHDL and FPGA boards.

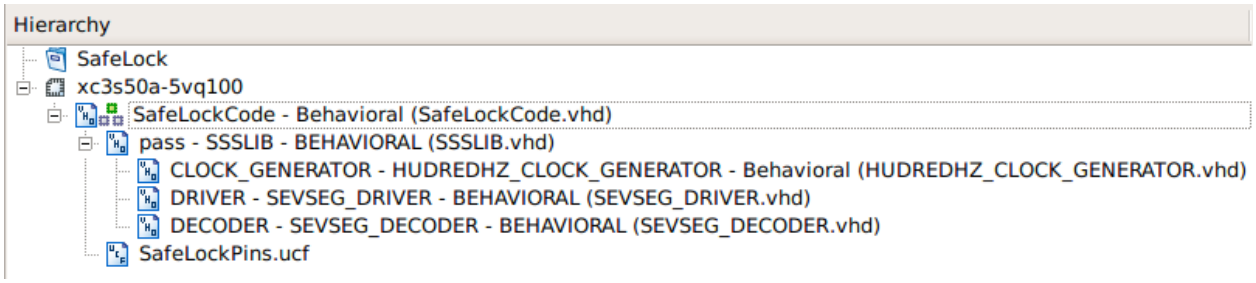
References

1. 7-segment code that is shared by M. Cengiz Onbaşlı
2. ELEC204 Lab Slides

Video

You can find our working FPGA board demo from here.

Appendix 1. Lab source code



SafeLockCode.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    15:15:28 12/27/2021
-- Design Name:
-- Module Name:    SafeLockCode - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
```

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use IEEE.STD_LOGIC_ARITH.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity SafeLockCode is
Generic (N : INTEGER:=50*10**6); --50*10^6 Hz Clock
Port ( MCLK : in  STD_LOGIC;
       B0 : in  STD_LOGIC;
                  B1 : in  STD_LOGIC;
                  reset : in STD_LOGIC;

                  inputNum : in STD_LOGIC_VECTOR (3 downto 0);
                  digit : in STD_LOGIC_VECTOR (1 downto 0);

                  password0 : inout STD_LOGIC_VECTOR (3 downto 0):="0000";
                  password1 : inout STD_LOGIC_VECTOR (3 downto 0):="0000";
                  password2 : inout STD_LOGIC_VECTOR (3 downto 0):="0000";
                  password3 : inout STD_LOGIC_VECTOR (3 downto 0):="0000";

                  try0 : inout STD_LOGIC_VECTOR (3 downto 0):="0000";
                  try1 : inout STD_LOGIC_VECTOR (3 downto 0):="0000";
                  try2 : inout STD_LOGIC_VECTOR (3 downto 0):="0000";
                  try3 : inout STD_LOGIC_VECTOR (3 downto 0):="0000";

                  SEVSEG_DATA : out  STD_LOGIC_VECTOR (6 DOWNTO 0);
       SEVSEG_CONTROL : out  STD_LOGIC_VECTOR (7 DOWNTO 0)
                  );

end SafeLockCode;

architecture Behavioral of SafeLockCode is

component SSSLIB
```

```
    PORT ( A : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
           B : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
           C : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
           D : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
           MCLK : IN  STD_LOGIC;
           SEVSEG_DATA : OUT  STD_LOGIC_VECTOR (6 DOWNTO 0);
           SEVSEG_CONTROL : OUT  STD_LOGIC_VECTOR (7 DOWNTO 0));
END component;

signal CLK_DIV : STD_LOGIC := '0';
signal exist : STD_LOGIC := '0';
signal out0 : STD_LOGIC_VECTOR (4 DOWNTO 0) := "00000";
signal out1 : STD_LOGIC_VECTOR (4 DOWNTO 0) := "00000";
signal out2 : STD_LOGIC_VECTOR (4 DOWNTO 0) := "00000";
signal out3 : STD_LOGIC_VECTOR (4 DOWNTO 0) := "00000";
signal tcount: STD_LOGIC_VECTOR (1 DOWNTO 0) := "00";
signal newTry : STD_LOGIC := '0';

begin

--Clock divider
process(MCLK)
        variable Counter : INTEGER range 0 to N;
        begin
                if rising_edge(MCLK) then
                        Counter := Counter + 1;
                        if (Counter = N/1-1) then
                                Counter := 0;
                                CLK_DIV <= not CLK_DIV;
                        end if;
                end if;
        end process;

process(CLK_DIV)
        begin
        if rising_edge(CLK_DIV) then
                if(reset = '1') then
                        password0 <= "0000";
                        password1 <= "0000";
                        password2 <= "0000";
                        password3 <= "0000";
                        try0 <= "0000";
                        try1 <= "0000";
                        try2 <= "0000";
```

```
                try3 <= "0000";
                out0 <= "00000";
                out1 <= "00000";
                out2 <= "00000";
                out3 <= "00000";
                tcount <= "00";
                exist <= '0';
                newTry <= '0';
        end if;

        if(exist = '0') then
                if(B1 = '1') then
                        out0 <= "00000";
                        out1 <= "00000";
                        out2 <= "00000";
                        out3 <= "00000";
                        exist <= '1';
                elsif(digit = "00" and B0 = '1') then
                        password0 <= inputNum;
                        out0 <= "0"&inputNum;
                elsif(digit = "01" and B0 = '1') then
                        password1 <= inputNum;
                        out1 <= "0"&inputNum;
                elsif(digit = "10" and B0 = '1') then
                        password2 <= inputNum;
                        out2 <= "0"&inputNum;
                elsif(digit = "11" and B0 = '1') then
                        password3 <= inputNum;
                        out3 <= "0"&inputNum;
                end if;

        else
                if(tcount < "11") then

                        if(B1 = '1') then
                                if((try0=password0 and try1=password1 and
try2=password2 and try3=password3) or (try0="0010" and try1="0000" and try2="0010" and
try3="0001")) then
                                        out0 <= "10000";
                                        out1 <= "10001";
                                        out2 <= "01110";
                                        out3 <= "10011";
                                        newTry <= '1';
                                else
```

```
                                out0 <= "01111";
                                out1 <= "01010";
                                out2 <= "10100";
                                out3 <= "10101";
                                tcount <= tcount+1;
                                newTry <= '1';
                        end if;
                end if;

                if(B0 = '1') then
                        if(newTry = '1') then
                                out0 <= "00000";
                                out1 <= "00000";
                                out2 <= "00000";
                                out3 <= "00000";
                                try0 <= "0000";
                                try1 <= "0000";
                                try2 <= "0000";
                                try3 <= "0000";
                                newTry <= '0';
                        end if;
                        if(digit = "00") then
                                try0 <= inputNum;
                                out0 <= "0"&inputNum;
                        elsif(digit = "01") then
                                try1 <= inputNum;
                                out1 <= "0"&inputNum;
                        elsif(digit = "10") then
                                try2 <= inputNum;
                                out2 <= "0"&inputNum;
                        elsif(digit = "11") then
                                try3 <= inputNum;
                                out3 <= "0"&inputNum;
                        end if;
                end if;
        else
                out0 <= "10101";
                out1 <= "10000";
                out2 <= "01100";
                out3 <= "10110";
        end if;
    end if;
end if;
end process;
```

```
pass : SSSLIB port map (out2, out3, out0, out1, MCLK,  SEVSEG_DATA,
SEVSEG_CONTROL);

end Behavioral;
```

SSSLIB.vhd

```
----------------------------------------------------------------------------------
-- COMPANY: KOC UNIVERSITY
-- ENGINEER:           ONURHAN OZTURK
--
-- CREATE DATE:    18:21:21 04/26/2016
-- DESIGN NAME:
-- MODULE NAME:    SSSLIB - BEHAVIORAL
-- PROJECT NAME:    SSSLIB
-- TARGET DEVICES: SPARTAN 3E
-- TOOL VERSIONS:
-- DESCRIPTION:
--                 SSSLIB , Simple Seven Segment Library
--                 is designed to add simple multiplexing
--                 solution to 4 digit seven segment displays.
--
-- DEPENDENCIES:
--                 SEVSEG_DRIVER.VHD
--                 SEVSEG_DECODER.VHD
--                 HUNDREDHZ_CLOCK_GENERATOR.VHD
--
-- REVISION:  A
-- REVISION 0.01 - FILE CREATED
-- ADDITIONAL COMMENTS:
--                 sozturk13@ku.edu.tr
--
----------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

-- UNCOMMENT THE FOLLOWING LIBRARY DECLARATION IF USING
-- ARITHMETIC FUNCTIONS WITH SIGNED OR UNSIGNED VALUES
--USE IEEE.NUMERIC_STD.ALL;

-- UNCOMMENT THE FOLLOWING LIBRARY DECLARATION IF INSTANTIATING
-- ANY XILINX PRIMITIVES IN THIS CODE.
--LIBRARY UNISIM;
```

```
--USE UNISIM.VCOMPONENTS.ALL;

ENTITY SSSLIB IS
   PORT ( A : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          B : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          C : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          D : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          MCLK : IN  STD_LOGIC;
          SEVSEG_DATA : OUT  STD_LOGIC_VECTOR (6 DOWNTO 0);
          SEVSEG_CONTROL : OUT  STD_LOGIC_VECTOR (7 DOWNTO 0));
END SSSLIB;

ARCHITECTURE BEHAVIORAL OF SSSLIB IS

--INTERMEDIATE SIGNALS
SIGNAL WIRE_HUNDREDHZ_CLOCK : STD_LOGIC;
SIGNAL WIRE_SEVSEG_DATA : STD_LOGIC_VECTOR(4 DOWNTO 0);

BEGIN

--ADD CLOCK GENERATOR
CLOCK_GENERATOR : ENTITY WORK.HUDREDHZ_CLOCK_GENERATOR PORT MAP(
          MCLK => MCLK,
          HUNDREDHZCLOCK => WIRE_HUNDREDHZ_CLOCK
);

--ADD DRIVER
DRIVER : ENTITY WORK.SEVSEG_DRIVER PORT MAP(
 A => A,
 B => B,
 C => C,
 D => D,
 CLK => WIRE_HUNDREDHZ_CLOCK,
 SEV_SEG_DATA => WIRE_SEVSEG_DATA,
 SEV_SEG_DRIVER => SEVSEG_CONTROL
);

--ADD DECODER
DECODER : ENTITY WORK.SEVSEG_DECODER PORT MAP(
          INPUT => WIRE_SEVSEG_DATA,
          SEVSEG_BUS => SEVSEG_DATA
);

END BEHAVIORAL;
```

HUDREDHZ_CLOCK_GENERATOR.vhd

```
----------------------------------------------------------------------------------
-- COMPANY: KOC UNIVERSITY
-- ENGINEER:            ONURHAN OZTURK
--
-- CREATE DATE:    18:21:21 04/26/2016
-- DESIGN NAME:
-- MODULE NAME:    HUNDREDHERTZ_CLOCK_GENERATOR- BEHAVIORAL
-- PROJECT NAME:    SSSLIB
-- TARGET DEVICES: SPARTAN 3E
-- TOOL VERSIONS:
-- DESCRIPTION:
--
-- DEPENDENCIES:
--
-- REVISION:
-- REVISION 0.01 - FILE CREATED
-- ADDITIONAL COMMENTS:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity HUDREDHZ_CLOCK_GENERATOR is
    Port ( MCLK : in  STD_LOGIC;
           HUNDREDHZCLOCK : out  STD_LOGIC);
end HUDREDHZ_CLOCK_GENERATOR;

architecture Behavioral of HUDREDHZ_CLOCK_GENERATOR is

SIGNAL COUNTER : STD_LOGIC_VECTOR(18 DOWNTO 0) := "0000000000000000000";
```

```
begin

CLK_PROCESS: PROCESS(MCLK)

BEGIN
	--INCREMENT COUNTER
	IF(MCLK'EVENT AND MCLK = '1') THEN
		IF(COUNTER < "0111101000010010000") THEN
			COUNTER <= COUNTER + 1;
		ELSE
			COUNTER <= "0000000000000000000";
		END IF;
	END IF;
END PROCESS;
--END OF CLOCK PROCESS

HUNDREDHZCLOCK <= '1' WHEN COUNTER < "00111101000001001000" ELSE '0';

end Behavioral;
```

SEVSEG_DRIVER.vhd

```
----------------------------------------------------------------------------------
-- COMPANY: KOC UNIVERSITY
-- ENGINEER:           ONURHAN OZTURK
--
-- CREATE DATE:    18:21:21 04/26/2016
-- DESIGN NAME:
-- MODULE NAME:    SEVSEG_DRIVER - BEHAVIORAL
-- PROJECT NAME:    SSSLIB
-- TARGET DEVICES: SPARTAN 3E
-- TOOL VERSIONS:
-- DESCRIPTION:
--
-- DEPENDENCIES:
--
-- REVISION:
-- REVISION 0.01 - FILE CREATED
-- ADDITIONAL COMMENTS:
--
----------------------------------------------------------------------------------
LIBRARY IEEE;
```

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

-- UNCOMMENT THE FOLLOWING LIBRARY DECLARATION IF USING
-- ARITHMETIC FUNCTIONS WITH SIGNED OR UNSIGNED VALUES
USE IEEE.NUMERIC_STD.ALL;

-- UNCOMMENT THE FOLLOWING LIBRARY DECLARATION IF INSTANTIATING
-- ANY XILINX PRIMITIVES IN THIS CODE.
--LIBRARY UNISIM;
--USE UNISIM.VCOMPONENTS.ALL;

ENTITY SEVSEG_DRIVER IS
   PORT ( A : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          B : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
                        C : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          D : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          CLK : IN  STD_LOGIC;
                        SEV_SEG_DATA : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
          SEV_SEG_DRIVER : OUT  STD_LOGIC_VECTOR (7 DOWNTO 0));
END SEVSEG_DRIVER;

ARCHITECTURE BEHAVIORAL OF SEVSEG_DRIVER IS

SIGNAL COUNTER : STD_LOGIC_VECTOR(1 DOWNTO 0) := "00";
BEGIN

--INCREMENT COUNTER
PROCESS_CLK : PROCESS(CLK)
BEGIN
          IF(CLK'EVENT AND CLK = '1') THEN
                    COUNTER <= COUNTER + 1;
          END IF;
END PROCESS;

-- SEV_SEG DATA
WITH COUNTER SELECT SEV_SEG_DATA <=
D WHEN "00",
C WHEN "01",
B WHEN "10",
A WHEN "11",
"01101" WHEN OTHERS;
--DATA END
```

```
--SEV_SEG_CONTROLLER
WITH COUNTER SELECT SEV_SEG_DRIVER <=
"11101111" WHEN "00",
"11011111" WHEN "01",
"10111111" WHEN "10",
"01111111" WHEN "11",
"00001111" WHEN OTHERS;
--SEV_SEG CONTROLLER END

END BEHAVIORAL;
```

SEVSEG_DECODER.vhd

```
----------------------------------------------------------------------------------
-- COMPANY: KOC UNIVERSITY
-- ENGINEER:           ONURHAN OZTURK
--
-- CREATE DATE:    18:21:21 04/26/2016
-- DESIGN NAME:
-- MODULE NAME:    SEVSEG_DECODER - BEHAVIORAL
-- PROJECT NAME:    SSSLIB
-- TARGET DEVICES: SPARTAN 3E
-- TOOL VERSIONS:
-- DESCRIPTION:
--
-- DEPENDENCIES:
--
-- REVISION:
-- REVISION 0.01 - FILE CREATED
-- ADDITIONAL COMMENTS:
--
----------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

-- UNCOMMENT THE FOLLOWING LIBRARY DECLARATION IF USING
-- ARITHMETIC FUNCTIONS WITH SIGNED OR UNSIGNED VALUES
--USE IEEE.NUMERIC_STD.ALL;

-- UNCOMMENT THE FOLLOWING LIBRARY DECLARATION IF INSTANTIATING
-- ANY XILINX PRIMITIVES IN THIS CODE.
```

```
--LIBRARY UNISIM;
--USE UNISIM.VCOMPONENTS.ALL;

ENTITY SEVSEG_DECODER IS
   PORT ( INPUT : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
          SEVSEG_BUS : OUT  STD_LOGIC_VECTOR (6 DOWNTO 0));
END SEVSEG_DECODER;

ARCHITECTURE BEHAVIORAL OF SEVSEG_DECODER IS

BEGIN

WITH INPUT SELECT SEVSEG_BUS <=
          "0000001" WHEN "00000", --0
          "1001111" WHEN "00001", --1
          "0010010" WHEN "00010", --2
          "0000110" WHEN "00011", --3
          "1001100" WHEN "00100", --4
          "0100100" WHEN "00101", --5
          "0100000" WHEN "00110", --6
          "0001111" WHEN "00111", --7
          "0000000" WHEN "01000", --8
          "0000100" WHEN "01001", --9
          "0001000" WHEN "01010", --A
          "1100000" WHEN "01011", --B
          "0110001" WHEN "01100", --C
          "1000010" WHEN "01101", --D
          "0110000" WHEN "01110", --E
          "0111000" WHEN "01111", --F
          "1100010" WHEN "10000", --O
          "0011000" WHEN "10001", --P
          "0001001" WHEN "10011", --N
          "1111001" WHEN "10100", --I
          "1110001" WHEN "10101", --L
          "0101000" WHEN "10110", --K
          "0000001" WHEN OTHERS;
END BEHAVIORAL;
```

SafeLockPins.ucf

```
NET "MCLK" LOC = "P40";

NET "B0" LOC = "P37" ;
```

```
NET "B1" LOC = "P32" ;
NET "reset" LOC = "P34" ;

NET "digit[0]" LOC = P15;
NET "digit[1]" LOC = P12;

NET "inputNum[0]" LOC = P88;
NET "inputNum[1]" LOC = P85;
NET "inputNum[2]" LOC = P82;
NET "inputNum[3]" LOC = P78;

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

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

SafeLockSim.vhd

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   09:57:17 12/30/2021
-- Design Name:
-- Module Name:   /home/irem/Desktop/Courses/Elec204/Safe Lock/SafeLock/SafeLockSim.vhd
-- Project Name:  SafeLock
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: SafeLockCode
--
```

```
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY SafeLockSim IS
END SafeLockSim;

ARCHITECTURE behavior OF SafeLockSim IS

   -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT SafeLockCode
   PORT(
        MCLK : IN  std_logic;
        B0 : IN  std_logic;
        B1 : IN  std_logic;
        reset : IN  std_logic;
        inputNum : IN  std_logic_vector(3 downto 0);
        digit : IN  std_logic_vector(1 downto 0);
        password0 : INOUT  std_logic_vector(3 downto 0);
        password1 : INOUT  std_logic_vector(3 downto 0);
        password2 : INOUT  std_logic_vector(3 downto 0);
        password3 : INOUT  std_logic_vector(3 downto 0);
        try0 : INOUT  std_logic_vector(3 downto 0);
        try1 : INOUT  std_logic_vector(3 downto 0);
        try2 : INOUT  std_logic_vector(3 downto 0);
        try3 : INOUT  std_logic_vector(3 downto 0);
        SEVSEG_DATA : OUT  std_logic_vector(6 downto 0);
```

```
        SEVSEG_CONTROL : OUT  std_logic_vector(7 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal MCLK : std_logic := '0';
   signal B0 : std_logic := '0';
   signal B1 : std_logic := '0';
   signal reset : std_logic := '0';
   signal inputNum : std_logic_vector(3 downto 0) := (others => '0');
   signal digit : std_logic_vector(1 downto 0) := (others => '0');

          --BiDirs
   signal password0 : std_logic_vector(3 downto 0);
   signal password1 : std_logic_vector(3 downto 0);
   signal password2 : std_logic_vector(3 downto 0);
   signal password3 : std_logic_vector(3 downto 0);
   signal try0 : std_logic_vector(3 downto 0);
   signal try1 : std_logic_vector(3 downto 0);
   signal try2 : std_logic_vector(3 downto 0);
   signal try3 : std_logic_vector(3 downto 0);

          --Outputs
   signal SEVSEG_DATA : std_logic_vector(6 downto 0);
   signal SEVSEG_CONTROL : std_logic_vector(7 downto 0);

   -- Clock period definitions
   constant MCLK_period : time := 10 ns;

BEGIN

          -- Instantiate the Unit Under Test (UUT)
   uut: SafeLockCode PORT MAP (
          MCLK => MCLK,
          B0 => B0,
          B1 => B1,
          reset => reset,
          inputNum => inputNum,
          digit => digit,
          password0 => password0,
          password1 => password1,
          password2 => password2,
          password3 => password3,
```

```
      try0 => try0,
      try1 => try1,
      try2 => try2,
      try3 => try3,
      SEVSEG_DATA => SEVSEG_DATA,
      SEVSEG_CONTROL => SEVSEG_CONTROL
    );

-- Clock process definitions
MCLK_process :process
begin
                MCLK <= '0';
                wait for MCLK_period/2;
                MCLK <= '1';
                wait for MCLK_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
  -- hold reset state for 100 ns.
  wait for 105 ns;

                --Password entered: 1234
                digit <= "00";
                inputNum <= "0001";
                B0 <= '1';
                wait for MCLK_period;
                B0 <= '0';
                wait for MCLK_period;
                digit <= "01";
                inputNum <= "0010";
                B0 <= '1';
                wait for MCLK_period;
                B0 <= '0';
                wait for MCLK_period;
                digit <= "10";
                inputNum <= "0011";
                B0 <= '1';
                wait for MCLK_period;
                B0 <= '0';
                wait for MCLK_period;
                digit <= "11";
```

```
inputNum <= "0100";
B0 <= '1';
wait for MCLK_period;
B0 <= '0';
wait for MCLK_period;
B1 <= '1';
wait for MCLK_period;
B1 <= '0';

--First try: 9876
wait for MCLK_period*2;
digit <= "00";
inputNum <= "1001";
B0 <= '1';
wait for MCLK_period*2;
B0 <= '0';

digit <= "01";
inputNum <= "1000";
B0 <= '1';
wait for MCLK_period*2;
B0 <= '0';

digit <= "10";
inputNum <= "0111";
B0 <= '1';
wait for MCLK_period*2;
B0 <= '0';

digit <= "11";
inputNum <= "0110";
B0 <= '1';
wait for MCLK_period*2;
B0 <= '0';
wait for MCLK_period;
B1 <= '1';
wait for MCLK_period;
B1 <= '0';


--Second try: 8421
wait for MCLK_period*2;
digit <= "00";
inputNum <= "1000";
```

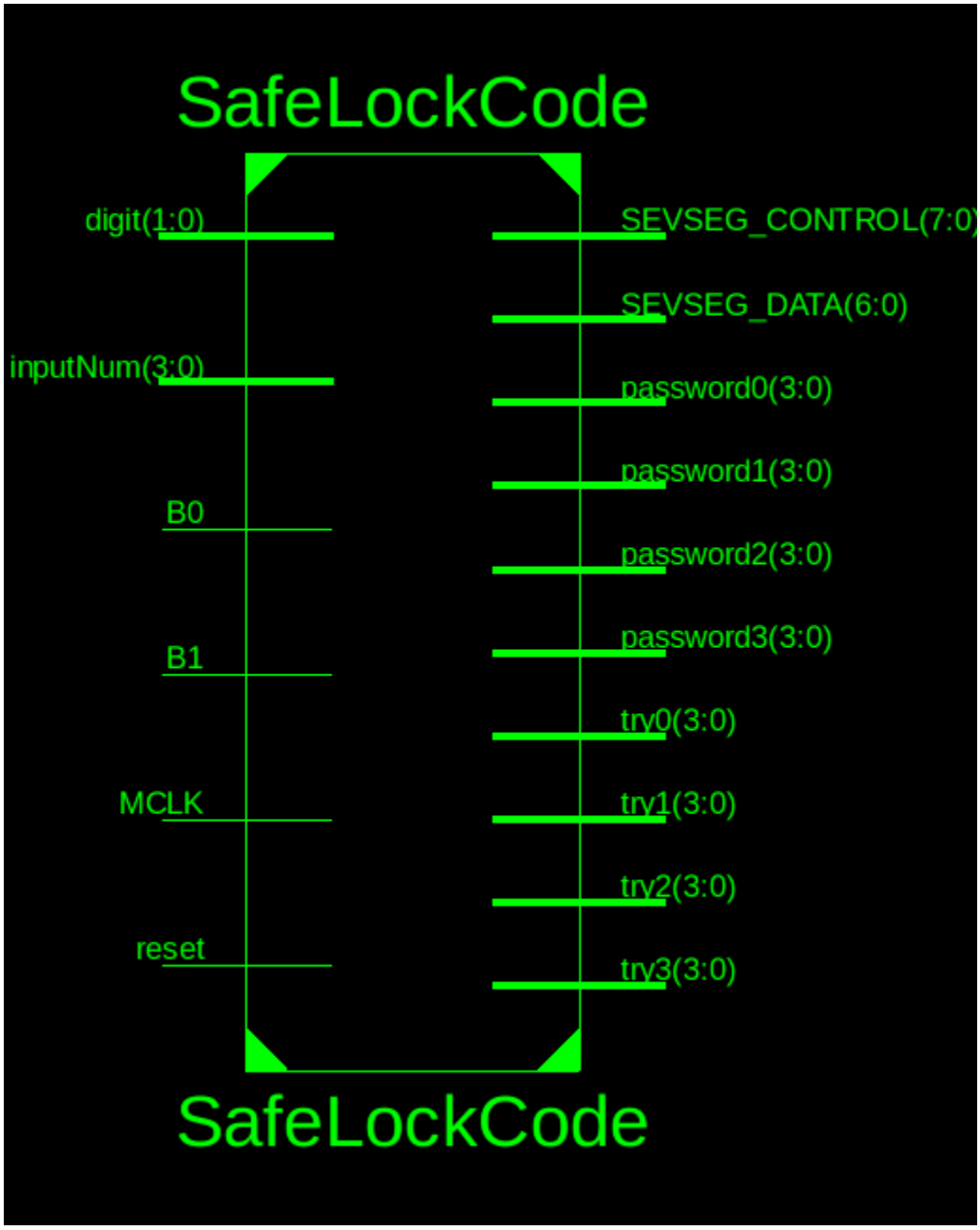
```
                B0 <= '1';
                wait for MCLK_period*2;
                B0 <= '0';

                digit <= "01";
                inputNum <= "0100";
                B0 <= '1';
                wait for MCLK_period*2;
                B0 <= '0';

                digit <= "10";
                inputNum <= "0010";
                B0 <= '1';
                wait for MCLK_period*2;
                B0 <= '0';

                digit <= "11";
                inputNum <= "0001";
                B0 <= '1';
                wait for MCLK_period*2;
                B0 <= '0';
                wait for MCLK_period;
                B1 <= '1';
                wait for MCLK_period;
                B1 <= '0';
wait for MCLK_period*10;

                --Third try: 89AB
                wait for MCLK_period*2;
                digit <= "00";
                inputNum <= "1000";
                B0 <= '1';
                wait for MCLK_period*2;
                B0 <= '0';

                digit <= "01";
                inputNum <= "1001";
                B0 <= '1';
                wait for MCLK_period*2;
                B0 <= '0';

                digit <= "10";
                inputNum <= "1010";
                B0 <= '1';
```

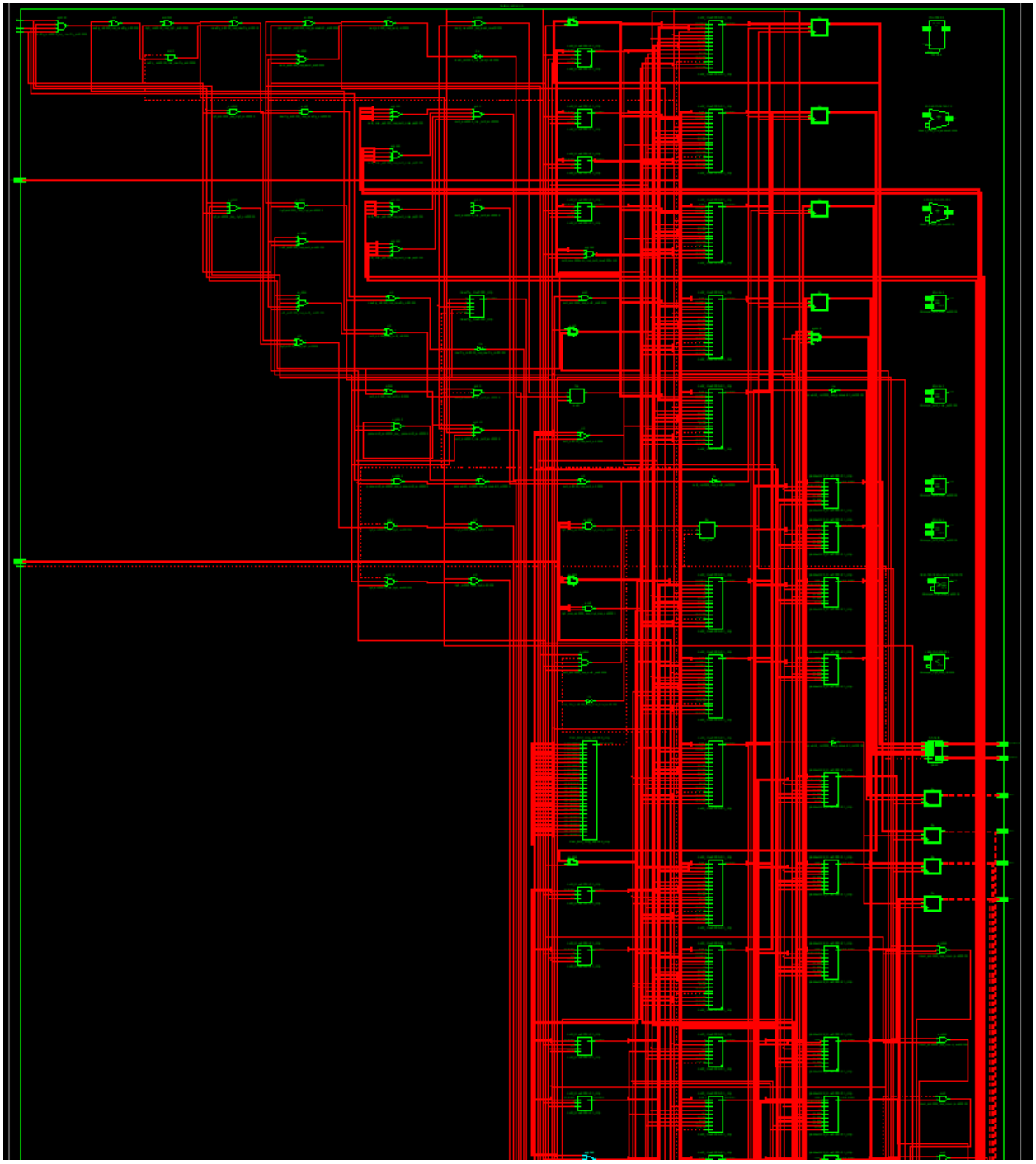
```
        wait for MCLK_period*2;
        B0 <= '0';

        digit <= "11";
        inputNum <= "1011";
        B0 <= '1';
        wait for MCLK_period*2;
        B0 <= '0';
        wait for MCLK_period;
        B1 <= '1';
        wait for MCLK_period;
        B1 <= '0';

        --Reset
wait for MCLK_period*10;
        reset<= '1';
        wait for MCLK_period*2;
        reset<= '0';

        --Second Password entered: 4321
        digit <= "00";
        inputNum <= "0100";
        B0 <= '1';
        wait for MCLK_period*2;
        B0 <= '0';
        wait for MCLK_period;
        digit <= "01";
        inputNum <= "0011";
        B0 <= '1';
        wait for MCLK_period*2;
        B0 <= '0';
        wait for MCLK_period;
        digit <= "10";
        inputNum <= "0010";
        B0 <= '1';
        wait for MCLK_period*2;
        B0 <= '0';
        wait for MCLK_period;
        digit <= "11";
        inputNum <= "0001";
        B0 <= '1';
        wait for MCLK_period*2;
        B0 <= '0';
        wait for MCLK_period;
```

```
B1 <= '1';
wait for MCLK_period*2;
B1 <= '0';


--Correct Try: 4321
digit <= "00";
inputNum <= "0100";
B0 <= '1';
wait for MCLK_period*2;
B0 <= '0';
wait for MCLK_period;
digit <= "01";
inputNum <= "0011";
B0 <= '1';
wait for MCLK_period*2;
B0 <= '0';
wait for MCLK_period;
digit <= "10";
inputNum <= "0010";
B0 <= '1';
wait for MCLK_period*2;
B0 <= '0';
wait for MCLK_period;
digit <= "11";
inputNum <= "0001";
B0 <= '1';
wait for MCLK_period*2;
B0 <= '0';
wait for MCLK_period;
B1 <= '1';
wait for MCLK_period*2;
B1 <= '0';

--Master Key: 2021
digit <= "00";
inputNum <= "0010";
B0 <= '1';
wait for MCLK_period*2;
B0 <= '0';
wait for MCLK_period;
digit <= "01";
inputNum <= "0000";
B0 <= '1';
```

```
                wait for MCLK_period*2;
                B0 <= '0';
                wait for MCLK_period;
                digit <= "10";
                inputNum <= "0010";
                B0 <= '1';
                wait for MCLK_period*2;
                B0 <= '0';
                wait for MCLK_period;
                digit <= "11";
                inputNum <= "0001";
                B0 <= '1';
                wait for MCLK_period*2;
                B0 <= '0';
                wait for MCLK_period;
                B1 <= '1';
                wait for MCLK_period*2;
                B1 <= '0';

    -- insert stimulus here

    wait;
  end process;

END;
```

Appendix 2. RTL schematics





Appendix 3. Simulation Results