Terminál pro odemčení / zamčení dveří pomocí 4místného PIN kódu

Členové týmu

Adam Hora, Daniel Haluška, Marek Halaš, Jan Honkyš

https://github.com/janhonkys/Digital-electronics-1/tree/main/Labs/project

Cíle projektu

Cílem projektu je vytvořit terminál pro odemčení / zamčení dveří pomocí 4místného PIN kódu, s použitím 4x3 tlačítek, 4 sedmisegmentových displejů, relé pro ovládání zámku dveří.

Návrh vizualizace řešení

Screenshot

Stavový diagram process p_result controller

Screenshot

Stavový diagram process p_keypad watcher

Screenshot

Popis hardwaru

Základní deska Arty A7-100T

Parametry: 4 přepínače, 4 tlačítka, 1 tlačítko reset, 4 LEDs, 4 RGB LEDs, interní hodinový signál, 4 PMOD rozhraní, USB-UART rozhraní

Screenshot

Klávesnice

Klávesnice je vzhledem k počtu vstupů na základní desce navržena maticově, 4 řádky, 3 sloupce.

Schéma zapojení klávesnice

Screenshot

Plošný spoj klávesnice

Screenshot

Osazovací plán plošného spoje klávesnice

Screenshot

Schéma zapojení displeje

Z důvodu omezených možností základní desky jsme navrhli použití 4 7mi segmentových displejů K121, zapojených se společnou katodou. V obrázku na horní straně schéma zapojení zámku a externí sirény.

Screenshot

Plošný spoj displeje

Screenshot

Osazovací plán plošného spoje displeje

Screenshot

Popis VHDL modulů a simulací

Klávesnice

Klávesnice je navržena tak, že po stisknutí tlačítka se nestane nic, dokud se tlačítko nepustí. Po puštění tlačítka je hodnota zapsaná na 1 impuls hodinového signálu do paměti. Při stlačení několika kláves naráz se nic nestane, při jejich puštění se do paměti zapíše nedefinovaná hodnota.

Převodní tabulka vstupů na výstup

Tlačítko	Výstup	hor_1	hor_2	hor_3	hor_4	ver_1	ver_2	ver_3
0	0000	0	0	0	1	0	1	0
1	0001	1	0	1	0	1	0	0
2	0010	0	0	1	0	0	1	0
3	0011	0	0	1	0	0	0	1
4	0100	0	1	0	0	1	0	0
5	0101	0	1	0	0	0	1	0
6	0110	0	1	0	0	0	0	1
7	0111	1	0	0	0	1	0	0
8	1000	1	0	0	0	0	1	0
9	1001	1	0	0	0	0	0	1
enter	1010	0	0	0	1	1	0	0
cancel	1011	0	0	0	1	0	0	1

Vstupní porty

```
entity keypad_4x3 is
    port(
        clk   : in  std_logic;                              -- input for clock
        hor_1 : in  std_logic;                              -- input for first row
        hor_2 : in  std_logic;                              -- input for second
row
        hor_3 : in  std_logic;                              -- input for third row
        hor_4 : in  std_logic;                              -- input for fourth
row
        ver_1 : in  std_logic;                              -- input for first
cloumn
        ver_2 : in  std_logic;                              -- input for second
column
        ver_3 : in  std_logic;                              -- input for third
column

        number_o : out std_logic_vector(4 - 1 downto 0)     -- output for number

      );
end keypad_4x3;
```

Architektura

```
architecture Behavioral of keypad_4x3 is

     signal s_number    : std_logic_vector(4 - 1 downto 0);

     constant number_0  : std_logic_vector(4 - 1 downto 0) := b"0000";      --
setting constants for numbel 0-9 in binary
     constant number_1  : std_logic_vector(4 - 1 downto 0) := b"0001";
     constant number_2  : std_logic_vector(4 - 1 downto 0) := b"0010";
     constant number_3  : std_logic_vector(4 - 1 downto 0) := b"0011";
     constant number_4  : std_logic_vector(4 - 1 downto 0) := b"0100";
     constant number_5  : std_logic_vector(4 - 1 downto 0) := b"0101";
     constant number_6  : std_logic_vector(4 - 1 downto 0) := b"0110";
     constant number_7  : std_logic_vector(4 - 1 downto 0) := b"0111";
     constant number_8  : std_logic_vector(4 - 1 downto 0) := b"1000";
     constant number_9  : std_logic_vector(4 - 1 downto 0) := b"1001";
     constant ENTER     : std_logic_vector(4 - 1 downto 0) := b"1010";      -- for
enter
     constant CANCEL    : std_logic_vector(4 - 1 downto 0) := b"1011";      -- for
cancel
     constant UNDEFINED : std_logic_vector(4 - 1 downto 0) := b"1111";      -- and
for undefined

begin


p_output_keypad : process(clk)
    begin
```

```
        if rising_edge(clk) then                        -- defining numbers based
on inputs

            number_o <= UNDEFINED;

            if(hor_1 = '0' AND hor_2 = '0' AND hor_3 = '0' AND hor_4 = '1' AND
ver_1 = '0' AND ver_2 = '1' AND ver_3 = '0')then         -- fourth row and second
column for value 0
                s_number <= number_0;
            elsif(hor_1 = '0' AND hor_2 = '0' AND hor_3 = '1' AND hor_4 = '0' AND
ver_1 = '1' AND ver_2 = '0' AND ver_3 = '0')then      -- third row and first
column for value 1
                s_number <= number_1;
            elsif(hor_1 = '0' AND hor_2 = '0' AND hor_3 = '1' AND hor_4 = '0' AND
ver_1 = '0' AND ver_2 = '1' AND ver_3 = '0')then      -- third row and second
column for value 2
                s_number <= number_2;
            elsif(hor_1 = '0' AND hor_2 = '0' AND hor_3 = '1' AND hor_4 = '0' AND
ver_1 = '0' AND ver_2 = '0' AND ver_3 = '1')then      -- third row and third
column for value 3
                s_number <= number_3;
            elsif(hor_1 = '0' AND hor_2 = '1' AND hor_3 = '0' AND hor_4 = '0' AND
ver_1 = '1' AND ver_2 = '0' AND ver_3 = '0')then      -- second row and first
column for value 4
                s_number <= number_4;
            elsif(hor_1 = '0' AND hor_2 = '1' AND hor_3 = '0' AND hor_4 = '0' AND
ver_1 = '0' AND ver_2 = '1' AND ver_3 = '0')then      -- second row and second
column for value 5
                s_number <= number_5;
            elsif(hor_1 = '0' AND hor_2 = '1' AND hor_3 = '0' AND hor_4 = '0' AND
ver_1 = '0' AND ver_2 = '0' AND ver_3 = '1')then      -- second row and third
column for value 6
                s_number <= number_6;
            elsif(hor_1 = '1' AND hor_2 = '0' AND hor_3 = '0' AND hor_4 = '0' AND
ver_1 = '1' AND ver_2 = '0' AND ver_3 = '0')then      -- first row and first
column for value 7
                s_number <= number_7;
            elsif(hor_1 = '1' AND hor_2 = '0' AND hor_3 = '0' AND hor_4 = '0' AND
ver_1 = '0' AND ver_2 = '1' AND ver_3 = '0')then      -- first row and second
column for value 8
                s_number <= number_8;
            elsif(hor_1 = '1' AND hor_2 = '0' AND hor_3 = '0' AND hor_4 = '0' AND
ver_1 = '0' AND ver_2 = '0' AND ver_3 = '1')then      -- first row and third
column for value 9
                s_number <= number_9;
            elsif(hor_1 = '0' AND hor_2 = '0' AND hor_3 = '0' AND hor_4 = '1' AND
ver_1 = '1' AND ver_2 = '0' AND ver_3 = '0')then      -- fourth row and first
column for ENTER
                s_number <= ENTER;
            elsif(hor_1 = '0' AND hor_2 = '0' AND hor_3 = '0' AND hor_4 = '1' AND
ver_1 = '0' AND ver_2 = '0' AND ver_3 = '1')then      -- fourth row and third
column for CANCEL
                s_number <= CANCEL;
            elsif(hor_1 = '0' AND hor_2 = '0' AND hor_3 = '0' AND hor_4 = '0' AND
ver_1 = '0' AND ver_2 = '0' AND ver_3 = '0')then      -- if we don't have inputs,
then number is undefined
                number_o <= s_number;    --
                s_number <= UNDEFINED;
            else                                -- if have been pressed 2 or more
buttons in same time, then number is undefined
                number_o <= UNDEFINED;
            end if;


        end if; -- Synchronous reset

    end process p_output_keypad;

end Behavioral;
```

Testbench klávesnice

```
architecture Behavioral of tb_keypad_4x3 is

    constant c_CLK_100MHZ_PERIOD : time := 10 ns;
    signal  s_clk_100MHz     : std_logic;
    signal  s_hor_1 :   std_logic;
    signal  s_hor_2 :   std_logic;
    signal  s_hor_3 :   std_logic;
    signal  s_hor_4 :   std_logic;
    signal  s_ver_1 :   std_logic;
    signal  s_ver_2 :   std_logic;
    signal  s_ver_3 :   std_logic;
    signal  s_number_i  : std_logic_vector(4 - 1 downto 0);

begin
     uut_keypad_4x3 : entity work.keypad_4x3
        port map(
            clk     => s_clk_100MHz,
            hor_1   => s_hor_1,
            hor_2   => s_hor_2,
            hor_3   => s_hor_3,
            hor_4   => s_hor_4,
            ver_1   => s_ver_1,
            ver_2   => s_ver_2,
            ver_3   => s_ver_3,
            number_o => s_number_i
        );

p_clk_gen : process
    begin
        while now < 100000000 ns loop   -- 10 usec of simulation
            s_clk_100MHz <= '0';
            wait for c_CLK_100MHZ_PERIOD / 2;
            s_clk_100MHz <= '1';
            wait for c_CLK_100MHZ_PERIOD / 2;
        end loop;
        wait;
    end process p_clk_gen;

     p_stimulus : process
    begin


        s_hor_1 <= '0';
        s_hor_2 <= '0';
        s_hor_3 <= '0';
        s_hor_4 <= '0';
        s_ver_1 <= '0';
        s_ver_2 <= '0';
        s_ver_3 <= '0';

        s_hor_4 <= '1';      --0
        s_ver_2 <= '1';
        wait for 40 ns;
        s_hor_4 <= '0';      --0
        s_ver_2 <= '0';
        wait for 40 ns;

        s_hor_1 <= '1';       --1
        s_hor_3 <= '1';
        s_ver_1 <= '1';
        wait for 40 ns;
        s_hor_1 <= '0';       --1
        s_hor_3 <= '0';
        s_ver_1 <= '0';
        wait for 40 ns;

        s_hor_3 <= '1';      --2
        s_ver_2 <= '1';
        wait for 40 ns;
        s_hor_3 <= '0';      --2
        s_ver_2 <= '0';
        wait for 40 ns;

        s_hor_3 <= '1';      --3
        s_ver_3 <= '1';
        wait for 40 ns;
        s_hor_3 <= '0';      --3
        s_ver_3 <= '0';
        wait for 40 ns;

        s_hor_2 <= '1';      --4
        s_ver_1 <= '1';
        wait for 40 ns;
        s_hor_2 <= '0';
        s_ver_1 <= '0';
        wait for 40 ns;
        s_hor_2 <= '1';     --5
        s_ver_2 <= '1';
        wait for 40 ns;
        s_hor_2 <= '0';
        s_ver_2 <= '0';
        wait for 40 ns;

        s_hor_2 <= '1';     --6
        s_ver_3 <= '1';
        wait for 40 ns;
        s_hor_2 <= '0';
        s_ver_3 <= '0';
        wait for 40 ns;

        s_hor_1 <= '1';     --7
        s_ver_1 <= '1';
        wait for 40 ns;
        s_hor_1 <= '0';
        s_ver_1 <= '0';
        wait for 40 ns;

        s_hor_1 <= '1';     --8
        s_ver_2 <= '1';
        wait for 40 ns;
        s_hor_1 <= '0';
        s_ver_2 <= '0';
        wait for 40 ns;

        s_hor_1 <= '1';     --9
        s_ver_3 <= '1';
        wait for 40 ns;
        s_hor_1 <= '0';
        s_ver_3 <= '0';
        wait for 40 ns;

        s_hor_4 <= '1';     --enter
        s_ver_1 <= '1';
        wait for 40 ns;
        s_hor_4 <= '0';
        s_ver_1 <= '0';
        wait for 40 ns;

        s_hor_4 <= '1';     --cancel
        s_ver_3 <= '1';
        wait for 40 ns;
        s_hor_4 <= '0';
        s_ver_3 <= '0';
        wait for 40 ns;

        wait;
        wait;
    end process p_stimulus;
```

```
end Behavioral;
```

Screenshot simulace tb_keypad

Postupně zkoušíme stisknutí jednotlivých tlačítek, simulace proběhla v pořádku, výstupní signál odpovídá převodní tabulce.

Screenshot

Hlavní řídící jednotka

Slouží ke zpracování vstupního signálu z klávesnice, který se podle rozhodovacího kritéria posoudí, jestli odpovídá správnému heslu. Když odpovídá správnému heslu, které je nastaveno na kombinaci čísel 2222, zámek dveři se otevře. Po dobu 10s budou dveře otevřeny (v simulaci nastaveno 1000ns), poté se zámek dveří zavře a čeká se na zavření dveří 10s (v simulaci 1000 ns), pokud se dveře do časového intervalu nezavřou (signál door_i), spustí se alarm, který je možný resetovat master heslem 1111. Při otevření dveří použitím master hesla je zámek otevřený na 10s (v simulaci 1000ns), ale následně se nečeká na zavření dveří, nespustí se alarm. Vstupní signál z klávesnice je zpracován na výstupní signály, které jsou předány ovladači 4 7mi segmentových displejů.

Vstupní porty

```
entity controler is
 port(
        clk       : in  std_logic;                                    -- input for
clock
        reset     : in  std_logic;                                    -- input for
reset
        number_i  : in  std_logic_vector(4 - 1 downto 0);             -- input for
number
        door_i    : in  std_logic;                                    -- input for
door
        alarm_o   : out  std_logic;                                   -- output for
alarm
        locker_o  : out  std_logic;                                   -- output for
"locker"

        rgb_o     : out  std_logic_vector(3 - 1 downto 0);            -- output for
RGB diode

        data0_o   : out  std_logic_vector(4 - 1 downto 0);            -- output for
first number of password
        data1_o   : out  std_logic_vector(4 - 1 downto 0);            -- output for
second number of password
        data2_o   : out  std_logic_vector(4 - 1 downto 0);            -- output for
third number of password
        data3_o   : out  std_logic_vector(4 - 1 downto 0)             -- output for
fourth number of password
```

```
    );
end controler;
```

Architektura

```
architecture Behavioral of controler is

    type   t_state is (CLOSE, OPENED, WAITH, AFTERTIME, ALARM, MASTER);
-- types of states
    signal s_state  : t_state;
-- assign state to signal

    type   t_state_pass is (POS1, POS2, POS3, POS4, ENT);
-- types of states by pressing of buttons
    signal s_state_pass  : t_state_pass;
-- assign state_pass to signal

    signal   s_clk  : std_logic;
-- signal for clock

    signal   s_cnt  : unsigned(32 - 1 downto 0):=
b"0000_0000_0000_0000_0000_0000_0000_0000"; -- nejak neviem zisti? ?o je to s_cnt

    signal   s_alarm  : std_logic;                          -- signal for
alarm
    signal   s_reset_pass : std_logic;                      -- signal for
reset password

    signal   s_pass_1  : std_logic_vector(4 - 1 downto 0);      -- signal for
first password position
    signal   s_pass_2  : std_logic_vector(4 - 1 downto 0);      -- for second
    signal   s_pass_3  : std_logic_vector(4 - 1 downto 0);      -- for third
    signal   s_pass_4  : std_logic_vector(4 - 1 downto 0);      -- for last
    signal   s_pass : std_logic_vector(16 - 1 downto 0);        -- signal for
whole password


    constant c_DELAY_10SEC : unsigned(32 - 1 downto 0) :=
b"0000_0000_0000_0000_0000_0000_0110_0100";      -- time constant for 10s (but
changed to 100ns, becose we want see that in our simulation)
    constant c_ZERO       : unsigned(32 - 1 downto 0) :=
b"0000_0000_0000_0000_0000_0000_0000_0000";       -- time constant for 0s

    constant c_MASTER_pass : std_logic_vector(16 - 1 downto 0) :=
b"0001_0001_0001_0001";                    -- master password is 1111
    constant c_SLAVE_pass : std_logic_vector(16 - 1 downto 0) :=
b"0010_0010_0010_0010";                     -- normal password is 2222
    constant c_UNDEFINED_pass : std_logic_vector(16 - 1 downto 0) :=
b"1111_1111_1111_1111";                 -- undefined password
```

```
    constant c_UNDEFINED : std_logic_vector(4 - 1 downto 0) := b"1111";
-- constant for unexistent button
    constant c_CANCEL : std_logic_vector(4 - 1 downto 0) := b"1011";
-- constant for cancel
    constant c_ENTER : std_logic_vector(4 - 1 downto 0) := b"1010";
-- constant for enter

    constant c_RED : std_logic_vector(3 - 1 downto 0) := b"100";
-- constants for rgb diode   - for red
    constant c_GREEN : std_logic_vector(3 - 1 downto 0) := b"010";
--                           - for green
    constant c_YELOW : std_logic_vector(3 - 1 downto 0) := b"110";
--                           - for yellow

    begin


    p_keypad_watcher : process(clk, s_reset_pass, reset)
    begin

    if reset = '1' then
        s_pass <= c_UNDEFINED_pass;                          -- password will be
undefined
        s_pass_1 <= c_UNDEFINED;                             -- first position will
be undefined
        s_pass_2 <= c_UNDEFINED;                             -- second
        s_pass_3 <= c_UNDEFINED;                             -- third
        s_pass_4 <= c_UNDEFINED;
        s_state_pass <= POS1;

    else
        if s_reset_pass = '1' then -- treba dako vyhutat        -- if s_reset_pass
= 1 then
            s_pass <= c_UNDEFINED_pass;                             -- password will
be undefined
            s_pass_1 <= c_UNDEFINED;                                -- first position
will be undefined
            s_pass_2 <= c_UNDEFINED;                                -- second
            s_pass_3 <= c_UNDEFINED;                                -- third
            s_pass_4 <= c_UNDEFINED;                                -- and last
        end if;

        if falling_edge(clk) then

                if(number_i = c_UNDEFINED)then

                                                                        -- proces for
enter password
                else

                    case (s_state_pass) is
                        when POS1 =>
-- assign first position
                            if (number_i = c_CANCEL OR number_i = c_ENTER)then
```

```
-- if is pressed cancel or enter
                                s_pass_1 <= c_UNDEFINED;
-- then first position will be undefined
                                s_pass_2 <= c_UNDEFINED;
-- second
                                s_pass_3 <= c_UNDEFINED;
-- third
                                s_pass_4 <= c_UNDEFINED;
-- fourth
                                s_state_pass <= POS1;
-- and whole password
                            else
                                s_pass_1 <= number_i;
-- if it is number 1-9 then we have first position
                                s_state_pass <= POS2;
-- and change state to POS2
                            end if;

                        when POS2 =>

-- pos2
                            if (number_i = c_CANCEL OR number_i = c_ENTER)then
-- same as pos1
                                s_pass_1 <= c_UNDEFINED;
                                s_pass_2 <= c_UNDEFINED;
                                s_pass_3 <= c_UNDEFINED;
                                s_pass_4 <= c_UNDEFINED;
                                s_state_pass <= POS1;
                            else
                                s_pass_2 <= number_i;
-- now we have second position
                                s_state_pass <= POS3;
-- and change state to POS3
                            end if;

                        when POS3 =>
-- same as pos2
                            if (number_i = c_CANCEL OR number_i = c_ENTER)then
                                s_pass_1 <= c_UNDEFINED;
                                s_pass_2 <= c_UNDEFINED;
                                s_pass_3 <= c_UNDEFINED;
                                s_pass_4 <= c_UNDEFINED;
                                s_state_pass <= POS1;
                            else
                                s_pass_3 <= number_i;
                                s_state_pass <= POS4;
                            end if;

                        when POS4 =>
-- same as pos2
                            if (number_i = c_CANCEL)then
                                s_pass_1 <= c_UNDEFINED;
                                s_pass_2 <= c_UNDEFINED;
                                s_pass_3 <= c_UNDEFINED;
                                s_pass_4 <= c_UNDEFINED;
                                s_state_pass <= POS1;
                            else
                                s_pass_4 <= number_i;
                                s_state_pass <= ENT;
-- but state will be changed to ENT
                            end if;

                         when ENT =>
-- state ENT
                            if (number_i = c_CANCEL)then
-- if is pressed CANCEL
                                s_pass_1 <= c_UNDEFINED;
-- then first position will be undefined
                                s_pass_2 <= c_UNDEFINED;
-- second
                                s_pass_3 <= c_UNDEFINED;
-- third
                                s_pass_4 <= c_UNDEFINED;
-- forth
                                s_state_pass <= POS1;
-- and whole password
                            elsif (number_i = c_ENTER)then
-- if is pressed ENTER
                                s_state_pass <= POS1;
-- state will change to POS1
                                s_pass <= s_pass_1 & s_pass_2 & s_pass_3 &
s_pass_4 ;   -- pass_1, pass_2, pass_3 nad pass_4 will be merged
                                s_pass_1 <= c_UNDEFINED; -- abo nejaky znak iny
-- then all password position will be undefined
                                s_pass_2 <= c_UNDEFINED;
                                s_pass_3 <= c_UNDEFINED;
                                s_pass_4 <= c_UNDEFINED;
                            else
                                s_state_pass <= ENT;
-- if we dont press enter or cancel then we are again in ENT
                            end if;
                         when others =>
                            s_state_pass <= POS1;
-- else state is again pos1
                     end case;

                 end if;

             end if;
    end if;
    end process p_keypad_watcher;


    p_result_controler : process(clk, reset)                             --
proces for result
    begin

    if reset = '1' then
```

```
           s_state <= CLOSE;
           locker_o <= '0';
           rgb_o <= c_RED;
           s_alarm <= '0';

       else

           if rising_edge(clk) then

                   case s_state is

                       when CLOSE =>
-- state close

                           if ((s_pass = c_MASTER_pass) OR (s_pass =
c_SLAVE_pass))then         -- if password is rigth (1111 or 2222)
                               s_state <= OPENED; -- mozna led
-- then state is changed to opened
                           else
                               s_state <= CLOSE;  -- mozna led
-- else
                               locker_o <= '0';
-- door is locked
                               rgb_o <= c_RED;
-- and color of diode is red
                           end if;

                       when OPENED =>                                      --
state opened

                           if (s_cnt < c_DELAY_10SEC) then                 -- if
the time is less than 10s
                               s_cnt <= s_cnt + 1;
                               locker_o <= '1';                            --
then door will be opened
                               rgb_o <= c_GREEN;                           -- and
diode is green
                           else
                               s_state <= WAITH;                           -- if
is time more than 10s
                               locker_o <= '0';                            --
door will be closed
                               s_cnt   <= c_ZERO;                          -- and
time is reset (0)
                           end if;

                       when WAITH =>                                       --
state waith

                           if (s_cnt < c_DELAY_10SEC) then                 -- if
the time is less than 10s
                               s_cnt <= s_cnt + 1;
                               rgb_o <= c_YELOW;                           --
then color of diode will be yellow
```

```
                        elsif (door_i = '1') then                                   -- if
the door will be closed
                            s_state <= CLOSE;                                       --
then we go to state close
                            s_reset_pass <= '1';                                    --
password is reseted
                            s_cnt   <= c_ZERO;                                      -- and
time will be reset too
                        else
                            s_state <= AFTERTIME;                                   -- it
the time is more than 10s then we go to state AFTERTIME
                            s_cnt   <= c_ZERO;                                      --
time is 0

                        end if;

                    when AFTERTIME =>                                               --
state aftertime
                        if (s_pass = c_MASTER_pass) then                            -- if
was password 1111 (master password)
                            s_state <= MASTER;                                      --
then state is changed to MASTER
                            s_reset_pass <= '1';                                    -- and
password is reseted
                        else
                        s_reset_pass <= '1';                                        --
else password is reseted
                        s_state <= ALARM;                                           -- and
state is changed to ALARM
                        end if;

                    when ALARM =>                                                   --
state alarm
                        s_alarm <= '1';                                             -- if
the alarm is on
                        s_state <= CLOSE;                                           --
then we go to state CLOSE

                    when MASTER =>                                                  --
state master
                        if (door_i = '1') then                                      -- if
door is closed
                            s_state <= CLOSE;                                       --
then we go to state CLOSE
                        else
                            s_state <= MASTER;                                      -- if
the door is open
                            rgb_o <= c_GREEN;                                       --
then we stay in state MASTER and diode is green
                        end if;
                    when others =>
                        s_state <= CLOSE;                                           --
ielse state is close
```

```
                end case;
              end if;

            if falling_edge(clk) then

            s_reset_pass <= '0';

            if (s_alarm = '1') then                         -- alarm is on
                if (s_pass = c_MASTER_pass)then             -- if the password was
master password
                    s_alarm <= '0';                         -- then alarm is off
                    s_reset_pass <= '1';                    -- and password is
reseted
                     s_state <= CLOSE;
                else
                    s_alarm <='1';                          -- else alarm is on
                end if;
            end if; -- Alarm
        end if; -- Falling edge
    end if;
    end process p_result_controler;

    alarm_o <= s_alarm ;                  -- signal shift from architecture to
entity (alarm)

    data0_o <= s_pass_1;                  -- signal shift from architecture to
entity (all parts of password)
    data1_o <= s_pass_2;
    data2_o <= s_pass_3;
    data3_o <= s_pass_4;

end Behavioral;
```

Testbench hlavní řídící jednotky

```
architecture testbench of tb_controler is

    constant c_CLK_100MHZ_PERIOD : time := 10 ns;

    --Local signals
    signal s_clk_100MHz     : std_logic;
    signal s_reset          : std_logic;
    signal s_number_i       : std_logic_vector(4 - 1 downto 0);
    signal s_door_i         : std_logic;
    signal s_alarm_o        : std_logic;
    signal s_locker_o       : std_logic;

    signal  s_hor_1 :   std_logic;
    signal  s_hor_2 :   std_logic;
    signal  s_hor_3 :   std_logic;
    signal  s_hor_4 :   std_logic;
    signal  s_ver_1 :   std_logic;
    signal  s_ver_2 :   std_logic;
    signal  s_ver_3 :   std_logic;

    signal  s_data0_o :   std_logic_vector(4-1 downto 0);
    signal  s_data1_o :   std_logic_vector(4-1 downto 0);
    signal  s_data2_o :   std_logic_vector(4-1 downto 0);
    signal  s_data3_o :   std_logic_vector(4-1 downto 0);

    signal  s_rgb_o :   std_logic_vector(3-1 downto 0);



begin
            -- signals shift from controler to signals in this tb
    uut_controler : entity work.controler
        port map(
            clk      => s_clk_100MHz,
            reset    => s_reset,
            number_i => s_number_i,
            door_i    => s_door_i ,
            alarm_o    => s_alarm_o ,
            locker_o   => s_locker_o,

            rgb_o => s_rgb_o,

            data0_o => s_data0_o,
            data1_o => s_data1_o,
            data2_o => s_data2_o,
            data3_o => s_data3_o

        );
            -- -- signals shift from keypad to signals in this tb
    uut_keypad_4x3 : entity work.keypad_4x3
        port map(
            clk      => s_clk_100MHz,
            hor_1    => s_hor_1,
            hor_2    => s_hor_2,
            hor_3    => s_hor_3,
            hor_4    => s_hor_4,
            ver_1    => s_ver_1,
            ver_2    => s_ver_2,
            ver_3    => s_ver_3,
            number_o => s_number_i
        );
         -- proces for clock
    p_clk_gen : process
    begin
        while now < 100000000 ns loop       -- while is time less than 10s
            s_clk_100MHz <= '0';            -- then clk_100MHz is 0
            wait for c_CLK_100MHZ_PERIOD / 2;
            s_clk_100MHz <= '1';            -- after 5 ns is 1
            wait for c_CLK_100MHZ_PERIOD / 2;
                                            -- and loop until time will be 10s
        end loop;
        wait;
    end process p_clk_gen;

    p_stimulus : process
    begin


        s_hor_1 <= '0';
        s_hor_2 <= '0';              -- set to 0
        s_hor_3 <= '0';
        s_hor_4 <= '0';
        s_ver_1 <= '0';
        s_ver_2 <= '0';
        s_ver_3 <= '0';
        s_door_i <= '0';

        s_reset <= '1';               -- set reset to 1
        wait for 100 ns;
        s_reset <= '0';               -- set reset to 0
        wait for 10 ns;

        s_hor_3 <= '1';               -- press 1
        s_ver_1 <= '1';
        wait for 40 ns;
        s_hor_3 <= '0';               -- set signals to zero
        s_ver_1 <= '0';
        wait for 40 ns;

        s_hor_3 <= '1';               -- press 1
        s_ver_1 <= '1';
        wait for 40 ns;
        s_hor_3 <= '0';               -- set signals to zero
        s_ver_1 <= '0';
        wait for 40 ns;

        s_hor_4 <= '1';
        s_ver_3 <= '1';               -- press cancel
        wait for 40 ns;
        s_hor_4 <= '0';               -- set signal to zero
        s_ver_3 <= '0';
        wait for 40 ns;


        s_hor_3 <= '1';               -- press 2
        s_ver_2 <= '1';
        wait for 40 ns;
        s_hor_3 <= '0';               -- set signals to zero
        s_ver_2 <= '0';
        wait for 40 ns;

        s_hor_3 <= '1';               -- press 2
        s_ver_2 <= '1';
        wait for 40 ns;
```

```
            s_hor_3 <= '0';             -- set signals to zero
            s_ver_2 <= '0';
            wait for 40 ns;

            s_hor_3 <= '1';             -- press 2
            s_ver_2 <= '1';
            wait for 40 ns;
            s_hor_3 <= '0';             -- set signals to zero
            s_ver_2 <= '0';
            wait for 40 ns;

            s_hor_3 <= '1';             -- press 2
            s_ver_2 <= '1';
            wait for 40 ns;
            s_hor_3 <= '0';             -- set signals to zero
            s_ver_2 <= '0';
            wait for 40 ns;


            s_hor_4 <= '1';             -- press enter
            s_ver_1 <= '1';
            wait for 40 ns;
            s_hor_4 <= '0';             -- set signals to zero
            s_ver_1 <= '0';

            wait for 1.6 us;
            s_door_i <= '1';            -- door will be closed
             wait for  40 ns;

            wait for 1 us;

            s_hor_3 <= '1';             -- press 2
            s_ver_2 <= '1';
            wait for 40 ns;
            s_hor_3 <= '0';             -- set signals to zero
            s_ver_2 <= '0';
            wait for 40 ns;

            s_hor_3 <= '1';             -- press 2
            s_ver_2 <= '1';
            wait for 40 ns;
            s_hor_3 <= '0';             -- set signals to zero
            s_ver_2 <= '0';
            wait for 40 ns;

            s_hor_3 <= '1';             -- press 2
            s_ver_2 <= '1';
            wait for 40 ns;
            s_hor_3 <= '0';             -- set signals to zero
            s_ver_2 <= '0';
            wait for 40 ns;

            s_hor_3 <= '1';             -- press 2
            s_ver_2 <= '1';
```

```
        wait for 40 ns;
        s_hor_3 <= '0';             -- set signals to zero
        s_ver_2 <= '0';
        wait for 40 ns;

        s_hor_4 <= '1';
        s_ver_1 <= '1';             -- press enter
        wait for 40 ns;
        s_hor_4 <= '0';             -- set signal to zero
        s_ver_1 <= '0';
        wait for 40 ns;
        s_door_i <= '0';            -- door will be closed
        wait for 2 us;

        s_hor_3 <= '1';             -- press 1
        s_ver_1 <= '1';
        wait for 40 ns;
        s_hor_3 <= '0';             -- set signals to zero
        s_ver_1 <= '0';
        wait for 40 ns;

        s_hor_3 <= '1';             -- press 1
        s_ver_1 <= '1';
        wait for 40 ns;
        s_hor_3 <= '0';             -- set signals to zero
        s_ver_1 <= '0';
        wait for 40 ns;

        s_hor_3 <= '1';             -- press 1
        s_ver_1 <= '1';
        wait for 40 ns;
        s_hor_3 <= '0';             -- set signals to zero
        s_ver_1 <= '0';
        wait for 40 ns;

        s_hor_3 <= '1';             -- press 1
        s_ver_1 <= '1';
        wait for 40 ns;
        s_hor_3 <= '0';             -- set signals to zero
        s_ver_1 <= '0';
        wait for 40 ns;

        s_hor_4 <= '1';
        s_ver_1 <= '1';             -- press enter
        wait for 40 ns;
        s_hor_4 <= '0';
        s_ver_1 <= '0';

        wait;

        wait;
    end process p_stimulus;
end testbench;
```

Screenshot simulace tb_controller

Na začátku zkoušíme zadat 2 čísla, poté stlačíme cancel. S_state_pass se vrátil na POS1. Následně zadáme správné heslo, potvrdíme enter, dveře se otevřou na dobu 1000 ns, následně jsou dveře zavřené, nedojde ke spuštění alarmu. Po chvíli zadáme správné heslo, ale dveře nezavřeme, zapne se alarm. Alarm se zruší zadáním master hesla 1111, při tomto zadání nedojde k otevření dveří, toto heslo se používá pro zrušení alarmu.

Screenshot

Ovladač 4 7mi segmentových displejů

Vstupní porty

```
entity driver_7seg_4digits is
    port(
        clk     : in  std_logic;        -- Main clock
        reset   : in  std_logic;        -- Synchronous reset
        -- 4-bit input values for individual digits
        data0_i : in  std_logic_vector(4 - 1 downto 0);
        data1_i : in  std_logic_vector(4 - 1 downto 0);
        data2_i : in  std_logic_vector(4 - 1 downto 0);
        data3_i : in  std_logic_vector(4 - 1 downto 0);

        -- Cathode values for individual segments
        seg_o   : out std_logic_vector(7 - 1 downto 0);
        -- Common anode signals to individual displays
        dig_o   : out std_logic_vector(4 - 1 downto 0)
    );
end entity driver_7seg_4digits;
```

Architektura

```
architecture Behavioral of driver_7seg_4digits is

    -- Internal clock enable
    signal s_en  : std_logic;
    -- Internal 2-bit counter for multiplexing 4 digits
    signal s_cnt : std_logic_vector(2 - 1 downto 0);
    -- Internal 4-bit value for 7-segment decoder
    signal s_hex : std_logic_vector(4 - 1 downto 0);

begin
    --------------------------------------------------------------------
    -- Instance (copy) of clock_enable entity generates an enable pulse
    -- every 4 ms
    clk_en0 : entity work.clock_enable
        generic map(
            --- WRITE YOUR CODE HERE
             g_MAX => 4 --400000

        )
        port map(
            --- WRITE YOUR CODE HERE
            clk =>clk,      -- Main clock
            reset =>reset,      -- Synchronous reset
            ce_o  =>s_en       -- Clock enable pulse signal
        );

    --------------------------------------------------------------------
    -- Instance (copy) of cnt_up_down entity performs a 2-bit down
    -- counter
    bin_cnt0 : entity work.cnt_up_down
        generic map(
            --- WRITE YOUR CODE HERE
            g_CNT_WIDTH => 2
        )
        port map(
            --- WRITE YOUR CODE HERE
            clk  => clk,          -- Main clock
            reset =>reset,       -- Synchronous reset
            en_i  =>s_en,       -- Enable input
            cnt_up_i => '0',       -- Direction of the counter
            cnt_o=>s_cnt
        );

    --------------------------------------------------------------------
    -- Instance (copy) of hex_7seg entity performs a 7-segment display
    -- decoder
    hex2seg : entity work.hex_7seg
        port map(
            hex_i => s_hex,
            seg_o => seg_o
        );

    --------------------------------------------------------------------
    -- p_mux:
    -- A combinational process that implements a multiplexer for
    -- selecting data for a single digit, a decimal point signal, and
    -- switches the common anodes of each display.
    --------------------------------------------------------------------
    p_mux : process(s_cnt, data0_i, data1_i, data2_i, data3_i)
    begin
        case s_cnt is
            when "11" =>
                s_hex <= data3_i;
                dig_o <= "0111";

            when "10" =>
                -- WRITE YOUR CODE HERE
                s_hex <= data2_i;
                dig_o <= "1011";
            when "01" =>
                -- WRITE YOUR CODE HERE
                s_hex <= data1_i;
                dig_o <= "1101";

            when others =>
                -- WRITE YOUR CODE HERE
                s_hex <= data0_i;
                dig_o <= "1110";
        end case;
    end process p_mux;

end architecture Behavioral;
```

Hodinový signál

Vstupní porty

```
entity clock_enable is
    generic(
        g_MAX : natural := 10       -- Number of clk pulses to generate
                                    -- one enable signal period
    );  -- Note that there IS a semicolon between generic and port
        -- sections
    port(
        clk   : in  std_logic;      -- Main clock
        reset : in  std_logic;      -- Synchronous reset
        ce_o  : out std_logic       -- Clock enable pulse signal
    );
end entity clock_enable;
```

Architektura

```
architecture Behavioral of clock_enable is

    -- Local counter
    signal s_cnt_local : natural;

begin
    --------------------------------------------------------------------
    -- p_clk_ena:
    -- Generate clock enable signal. By default, enable signal is low
    -- and generated pulse is always one clock long.
    --------------------------------------------------------------------
    p_clk_ena : process(clk)
    begin
        if rising_edge(clk) then        -- Synchronous process
```

```vhdl
            if (reset = '1') then        -- High active reset
                s_cnt_local <= 0;        -- Clear local counter
                ce_o        <= '0';      -- Set output to low

            -- Test number of clock periods
            elsif (s_cnt_local >= (g_MAX - 1)) then
                s_cnt_local <= 0;        -- Clear local counter
                ce_o        <= '1';      -- Generate clock enable pulse

            else
                s_cnt_local <= s_cnt_local + 1;
                ce_o        <= '0';
            end if;
        end if;
    end process p_clk_ena;

end architecture Behavioral;
```

Obousměrný čítač

Vstupní porty

```vhdl
entity cnt_up_down is
    generic(
        g_CNT_WIDTH : natural := 4      -- Number of bits for counter
    );
    port(
        clk      : in  std_logic;       -- Main clock
        reset    : in  std_logic;       -- Synchronous reset
        en_i     : in  std_logic;       -- Enable input
        cnt_up_i : in  std_logic;       -- Direction of the counter
        cnt_o    : out std_logic_vector(g_CNT_WIDTH - 1 downto 0)
    );
end entity cnt_up_down;
```

Architektura

```vhdl
architecture behavioral of cnt_up_down is

    -- Local counter
    signal s_cnt_local : unsigned(g_CNT_WIDTH - 1 downto 0);

begin
    --------------------------------------------------------------------
    -- p_cnt_up_down:
    -- Clocked process with synchronous reset which implements n-bit
    -- up/down counter.
    --------------------------------------------------------------------
    p_cnt_up_down : process(clk)
    begin
        if rising_edge(clk) then

            if (reset = '1') then               -- Synchronous reset
                s_cnt_local <= (others => '0'); -- Clear all bits

            elsif (en_i = '1') then       -- Test if counter is enabled

                -- TEST COUNTER DIRECTION HERE
                if (cnt_up_i = '1') then
                    s_cnt_local <= s_cnt_local + 1;
                elsif (cnt_up_i = '0') then
                    s_cnt_local <= s_cnt_local - 1;
                end if;
            end if;
        end if;
    end process p_cnt_up_down;

    -- Output must be retyped from "unsigned" to "std_logic_vector"
    cnt_o <= std_logic_vector(s_cnt_local);

end architecture behavioral;
```

Dekodér na 7mi segmentový displej

Slouží k převodu vstupního 4 bitového signálu na výstupní 7mi bitový signál, který se zobrazuje na displeji.

Převodní tabulka dekodéru na 7mi segmentový displej

Hex	Vstup	A	B	C	D	E	F	G
0	0000	0	0	0	0	0	0	1
1	0001	1	0	0	1	1	1	1
2	0010	0	0	1	0	0	1	0
3	0011	0	0	0	0	1	1	0
4	0100	1	0	0	1	1	0	0
5	0101	0	1	0	0	1	0	0
6	0110	0	1	0	0	0	0	0
7	0111	0	0	1	1	1	1	1
8	1000	0	0	0	0	0	0	0
9	1001	0	0	0	0	1	0	0
A	1010	0	0	0	1	0	0	0

Hex	Vstup	A	B	C	D	E	F	G
B	1011	1	1	0	0	0	0	0
C	1100	0	1	1	0	0	0	1
D	1101	1	0	0	0	0	1	0
E	1110	0	1	1	0	0	0	0
F	1111	0	1	1	1	0	0	0

Architektura

```
architecture Behavioral of hex_7seg is
begin

p_7seg_decoder : process(hex_i)
    begin
        case hex_i is
            when "0000" =>
                seg_o <= "0000001";     -- 0
            when "0001" =>
                seg_o <= "1001111";     -- 1
            when "0010" =>
                seg_o <= "0010010";     -- 2
            when "0011" =>
                seg_o <= "0000110";     -- 3
            when "0100" =>
                seg_o <= "1001100";     -- 4
            when "0101" =>
                seg_o <= "0100100";     -- 5
            when "0110" =>
                seg_o <= "0100000";     -- 6
            when "0111" =>
                seg_o <= "0001111";     -- 7
            when "1000" =>
                seg_o <= "0000000";     -- 8
            when "1001" =>
                seg_o <= "0000100";     -- 9
            when "1010" =>
                seg_o <= "0001000";     -- A
            when "1011" =>
                seg_o <= "1100000";     -- B
            when "1100" =>
                seg_o <= "0110001";     -- C
            when "1101" =>
                seg_o <= "1000010";     -- D
            when "1110" =>
                seg_o <= "0110000";     -- E
            when others =>
                seg_o <= "1111111";     -- NOTHING
        end case;
    end process p_7seg_decoder;

end architecture Behavioral;
```

TOP modul a simulace

Schéma TOP modulu

Screenshot

Vstupní porty

```
entity top is
    Port (
           CLK100MHZ : in STD_LOGIC;

           btn : in STD_LOGIC_VECTOR (4-1 downto 0);    --reset

           ja : in STD_LOGIC_VECTOR (8-1 downto 0);     -- door imput for keypad
           jb : out STD_LOGIC_VECTOR (8-1 downto 0);    -- cathodes for 7-segment
display
           jc : out STD_LOGIC_VECTOR (8-1 downto 0);    -- anodes for alarm, door,
locker

           led0_b : out STD_LOGIC;                      -- led rgb
           led0_g : out STD_LOGIC;
           led0_r : out STD_LOGIC

           );
end top;
```

Architektura TOP modulu

```
architecture Behavioral of top is
    -- No internal signals

    signal   s_number   : std_logic_vector(4 - 1 downto 0);

    signal   s_data0  : std_logic_vector(4 - 1 downto 0);
    signal   s_data1  : std_logic_vector(4 - 1 downto 0);
    signal   s_data2  : std_logic_vector(4 - 1 downto 0);
    signal   s_data3  : std_logic_vector(4 - 1 downto 0);


begin

    --------------------------------------------------------------------
    -- Instance (copy) of driver_7seg_4digits entity
    uut_keypad_4x3 : entity work.keypad_4x3
        port map(
            clk      => CLK100MHZ,
            hor_1    => ja(0),
            hor_2    => ja(1),
            hor_3    => ja(2),
            hor_4    => ja(3),
            ver_1    => ja(4),
            ver_2    => ja(5),
            ver_3    => ja(6),
            number_o => s_number
        );

     uut_controler : entity work.controler
        port map(
            clk      => CLK100MHZ,
            reset    => btn(0),
            number_i => s_number,
            door_i    => ja(7),
            alarm_o    => jc(7),
            locker_o   => jc(6),
            data0_o => s_data0,
            data1_o => s_data1,
            data2_o => s_data2,
            data3_o => s_data3
    );

    driver_seg_4 : entity work.driver_7seg_4digits
        port map(
            clk         => CLK100MHZ,
            reset       => btn(0),

            data0_i => s_data0,
            data1_i => s_data1,
            data2_i => s_data2,
            data3_i => s_data3,


            seg_o(6) => jb(6),
            seg_o(5) => jb(5),
            seg_o(4) => jb(4),
            seg_o(3) => jb(3),
            seg_o(2) => jb(2),
            seg_o(1) => jb(1),
            seg_o(0) => jb(0)
     );

end architecture Behavioral;
```

Diskuse o výsledcích

Zadání projektu jsme rozšířili o LED RGB diodu, která mění barvu podle aktuálního stavu v process controller a o alarm, který může být realizovaný sirénou. Výsledky projektu se shodují s naší představou a očekáváním. Program by bylo vhodné otestoval na hardwaru a ověřit funkčnost.

Video

https://vutbr-my.sharepoint.com/✌
/g/personal/xhonky00_vutbr_cz/EStso6UwhyFKhILq08kfJZEBm5fL0fUlk9VYA59FjubmrA?e=ymaxjz

Reference

https://github.com/shahjui2000/Push-Button-Door-VHDL-

https://www.kth.se/social/files/5458faeef276544021bdf437/codelockVHDL_eng.pdf

https://www.fpga4student.com/2017/09/vhdl-code-for-seven-segment-display.html