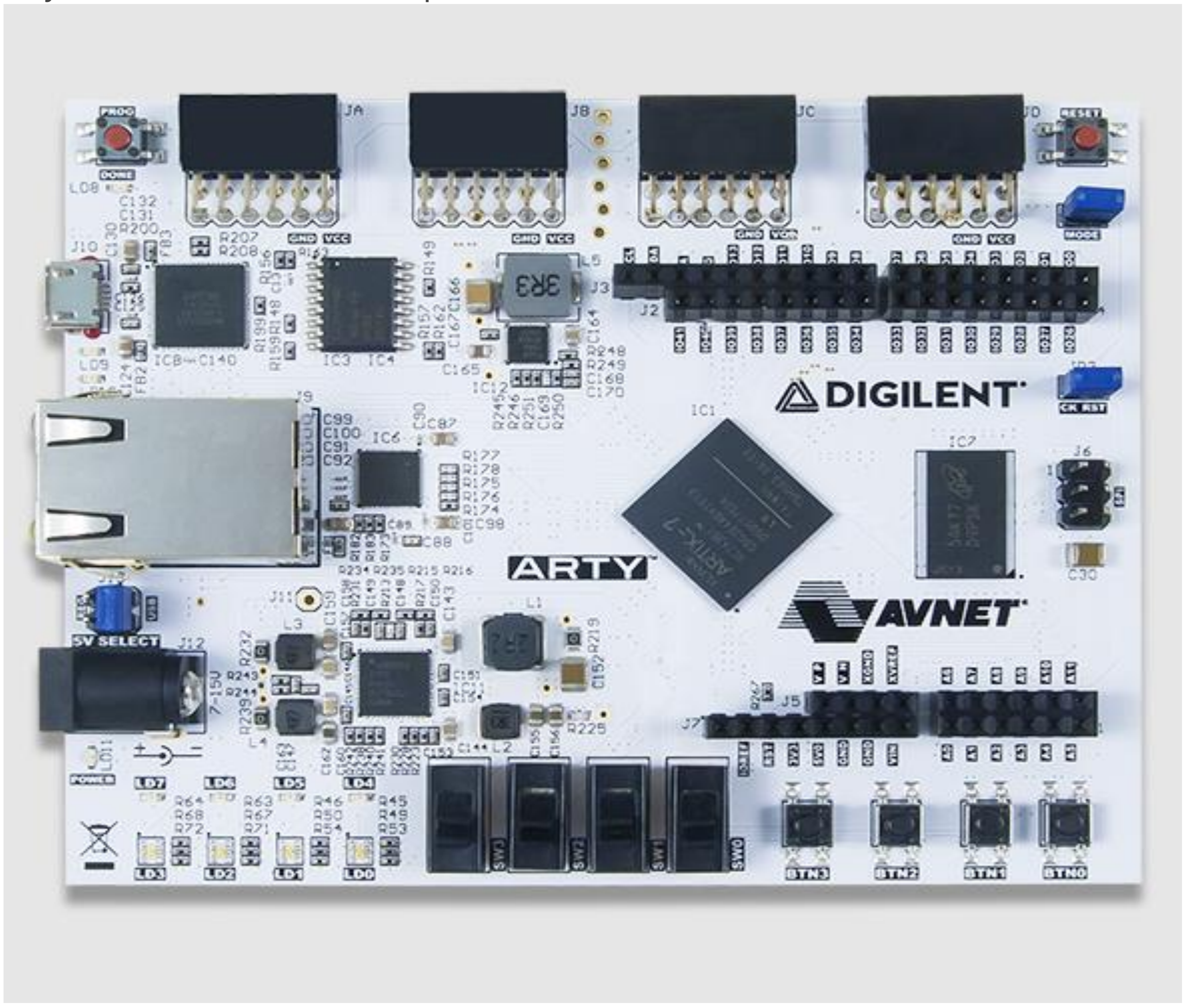
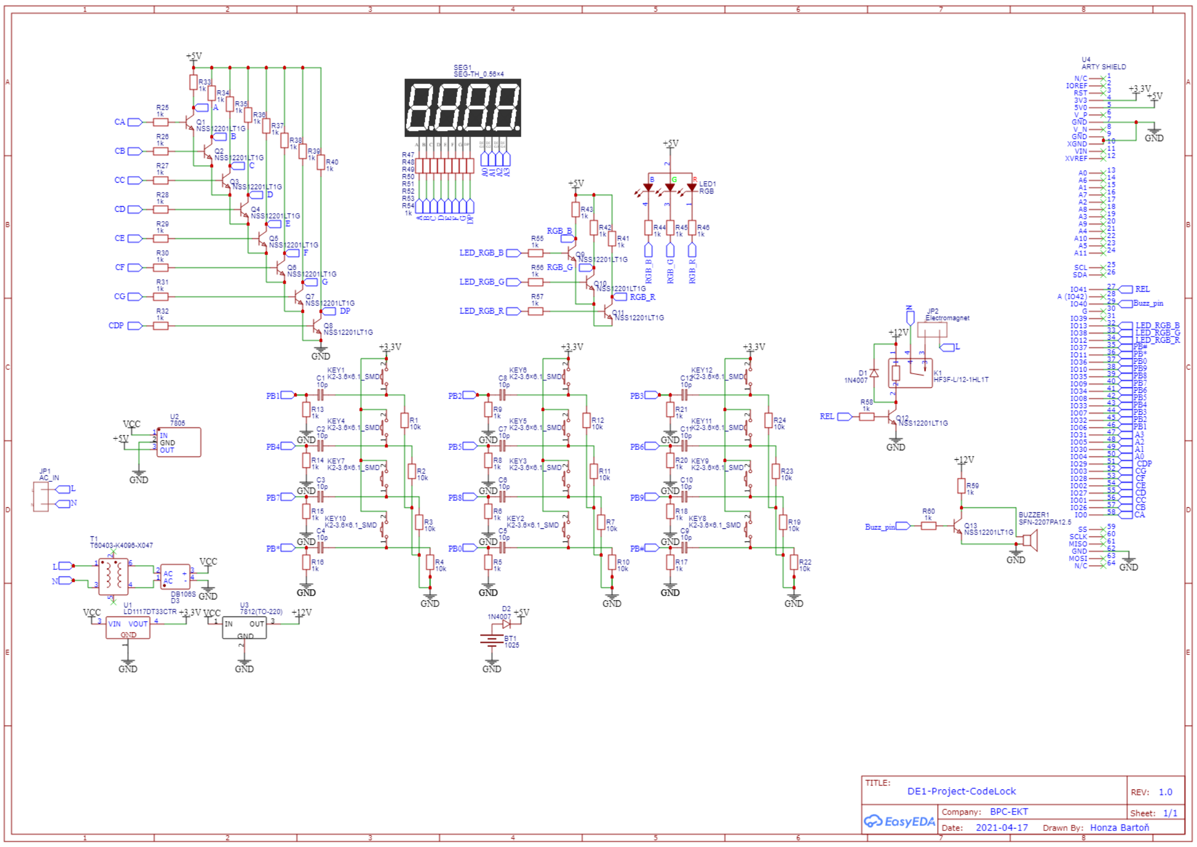
Door lock system

user manual

Arty A7: Artix-7 FPGA Development Board



Detailed connection



Input for Artix-7:

Keyboard 3x4

Press the asterisk to enter the password. After pressing you have a limited time to enter the password. Press the cross to confirm the password. If you want to change the password, press the cross on the keyboard, the password will be checked and if the password is entered correctly the door will be open for a while. It may happen that the password is entered incorrectly, then enter your password again, but if you enter the wrong password three times, an alarm will be triggered. The pin code is a four-digit number that includes numbers from 0 to 9.

Outputs for Artix-7:

LED Display

The display is blank in default state. When a value is written to it, at the leading edge it checks at which position on the display is entered, then it is checked whether the numeric key is pressed, if so, it writes the given digit at the position, if not, it remains in the default state until the integer is entered. The password is checked and the door is opened, an alarm is triggered, an incorrect password or a password is set. That is about the function. The display shows the numbers you entered using the keypad. If the password is entered incorrectly, the display returns to its original state and you can enter the password again.

Beeper

To trigger an alarm (red and duration for five minutes)

Constant tone

RGB LED diods

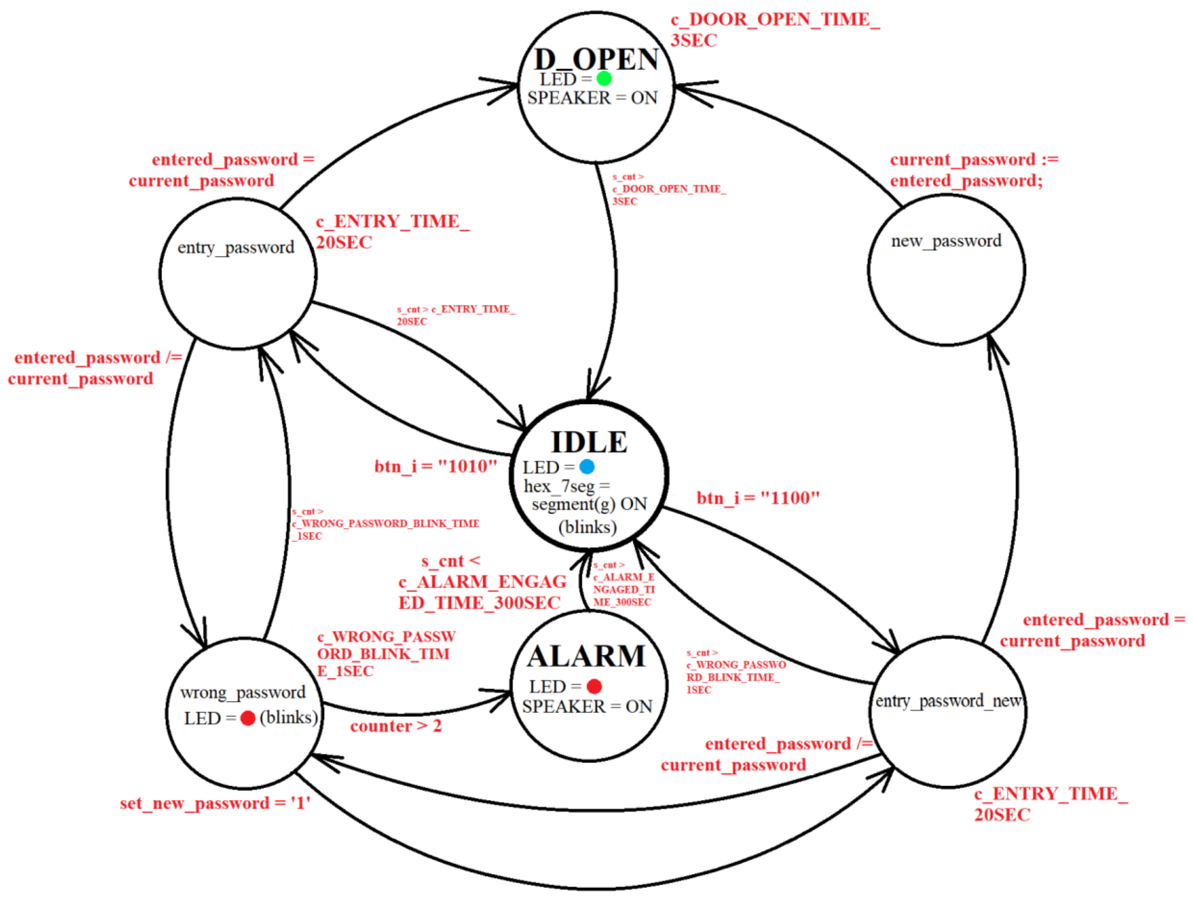
Green - You have entered the correct password

Red - You have entered an incorrect password

Blue color - for normal operation of the terminal (indicates power failures)

Switch relay

Door lock control, activated for a certain time. When the color on the LED is green, the relay is deactivated.


c_DOOR_OPEN_TIME_ 3SEC
D_OPEN
LED =
SPEAKER = ON
entered_password = current_password
s_cnt > c_DOOR_OPEN_TIME_ 3SEC
current_password := entered_password;
c_ENTRY_TIME_ 20SEC
entry_password
new_password
s_cnt > c_ENTRY_TIME_ 20SEC
entered_password /= current_password
IDLE
btn_i = "1010"
LED =
hex_7seg = segment(g) ON (blinks)
btn_i = "1100"
s_cnt > c_WRONG_PASSWORD_BLINK_TIME _1SEC
s_cnt < c_ALARM_ENGAGED_TIME_300SEC
s_cnt > c_ALARM_ENGAGED_TIME_300SEC
entered_password = current_password
c_WRONG_PASSWORD_BLINK_TIME_1SEC
ALARM
LED =
SPEAKER = ON
s_cnt > c_WRONG_PASSWORD_BLINK_TIME_ 1SEC
wrong_password
LED = (blinks)
counter > 2
entry_password_new
entered_password /= current_password
set_new_password = '1'
c_ENTRY_TIME_ 20SEC

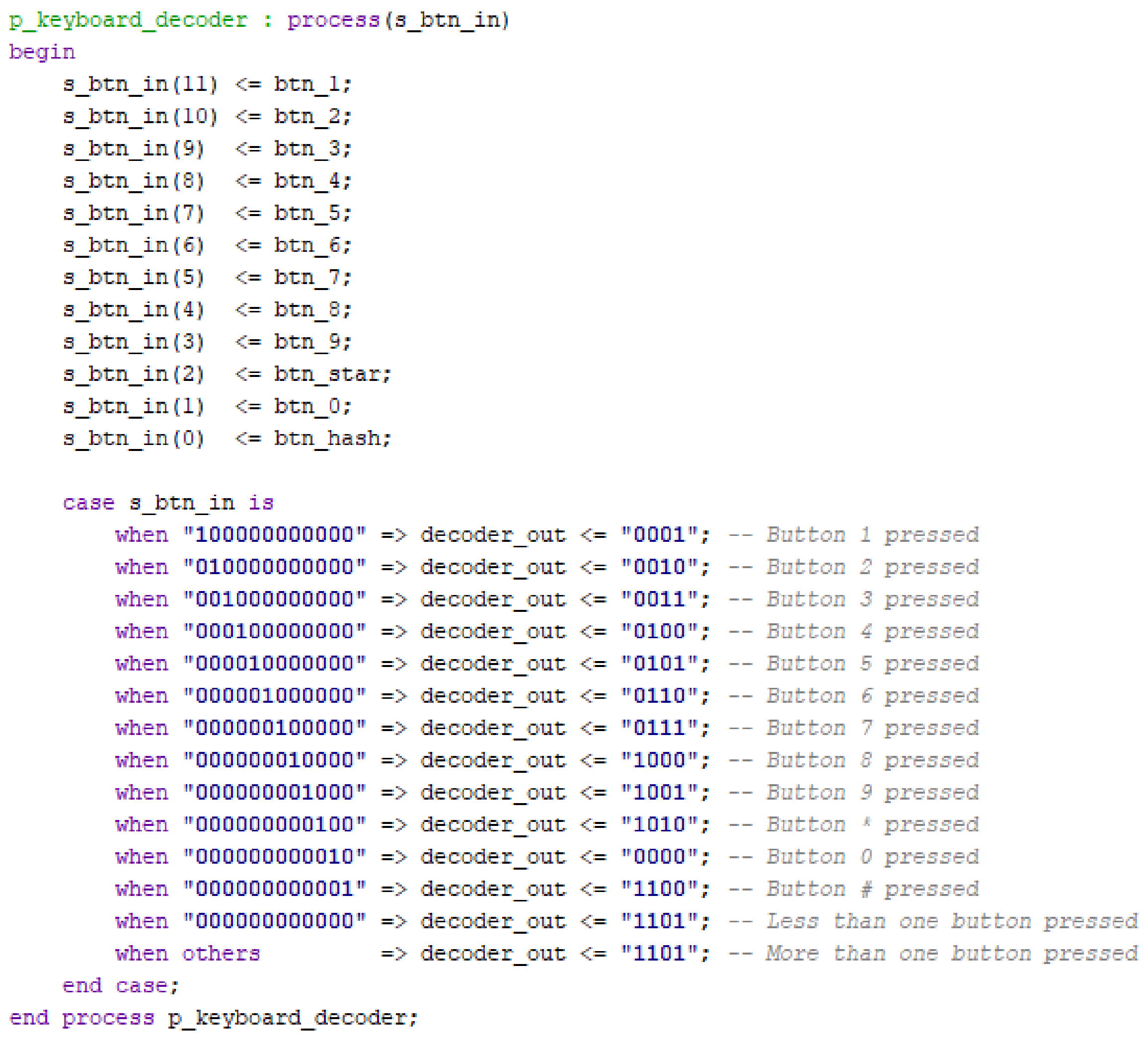
Description of states according to the state diagram:

IDLE is the default state and will change state depending on future use. The LED will flash blue.

If the asterisk is tapped, you will be able to enter a four-digit code that describes the four-bit binary code at the btn_i (input for each button). Next, it enters the entry_password state, where the password for opening the door is entered. If you enter the wrong password, it will move you back to the IDLE state, the same happen if you do not have time to enter the password within twenty seconds. If you enter the wrong password, it will go into the wrong_password state, the LED will flash red. You have three attempts to enter the correct password, so it will always go back to the entry_password state. Three incorrect attempts mean that it will go into the ALARM state and then the red LED will light up. After 300 seconds, it returns to the default IDLE state. It is also possible to change your password from the IDLE state by pressing the cross, it goes to the entry_password_new state, which takes 20 seconds, during this time you have time to enter a new four-digit password, which again has a four-digit binary value, but only after entering your password. If you do not enter your unchanged password, it will return to the wrong_password state. After successfully entering the password, you can change your password in the new_password state. When you have finished entering the new password, the door will open.

Variants that complement our door lock system:

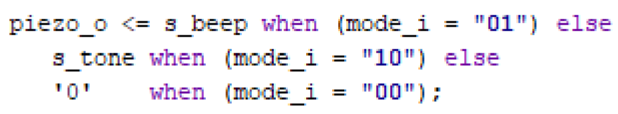
Keyboard decoder



```
p_keyboard_decoder : process(s_btn_in)
begin
    s_btn_in(11) <= btn_1;
    s_btn_in(10) <= btn_2;
    s_btn_in(9)  <= btn_3;
    s_btn_in(8)  <= btn_4;
    s_btn_in(7)  <= btn_5;
    s_btn_in(6)  <= btn_6;
    s_btn_in(5)  <= btn_7;
    s_btn_in(4)  <= btn_8;
    s_btn_in(3)  <= btn_9;
    s_btn_in(2)  <= btn_star;
    s_btn_in(1)  <= btn_0;
    s_btn_in(0)  <= btn_hash;

    case s_btn_in is
        when "100000000000" => decoder_out <= "0001"; -- Button 1 pressed
        when "010000000000" => decoder_out <= "0010"; -- Button 2 pressed
        when "001000000000" => decoder_out <= "0011"; -- Button 3 pressed
        when "000100000000" => decoder_out <= "0100"; -- Button 4 pressed
        when "000010000000" => decoder_out <= "0101"; -- Button 5 pressed
        when "000001000000" => decoder_out <= "0110"; -- Button 6 pressed
        when "000000100000" => decoder_out <= "0111"; -- Button 7 pressed
        when "000000010000" => decoder_out <= "1000"; -- Button 8 pressed
        when "000000001000" => decoder_out <= "1001"; -- Button 9 pressed
        when "000000000100" => decoder_out <= "1010"; -- Button * pressed
        when "000000000010" => decoder_out <= "0000"; -- Button 0 pressed
        when "000000000001" => decoder_out <= "1100"; -- Button # pressed
        when "000000000000" => decoder_out <= "1101"; -- Less than one button pressed
        when others         => decoder_out <= "1101"; -- More than one button pressed
    end case;
end process p_keyboard_decoder;
```

The decoder works on the principle that from a 12-bit binary number, which comes as a signal to the decoder input, it sends a 4-bit binary number from the decoder output.

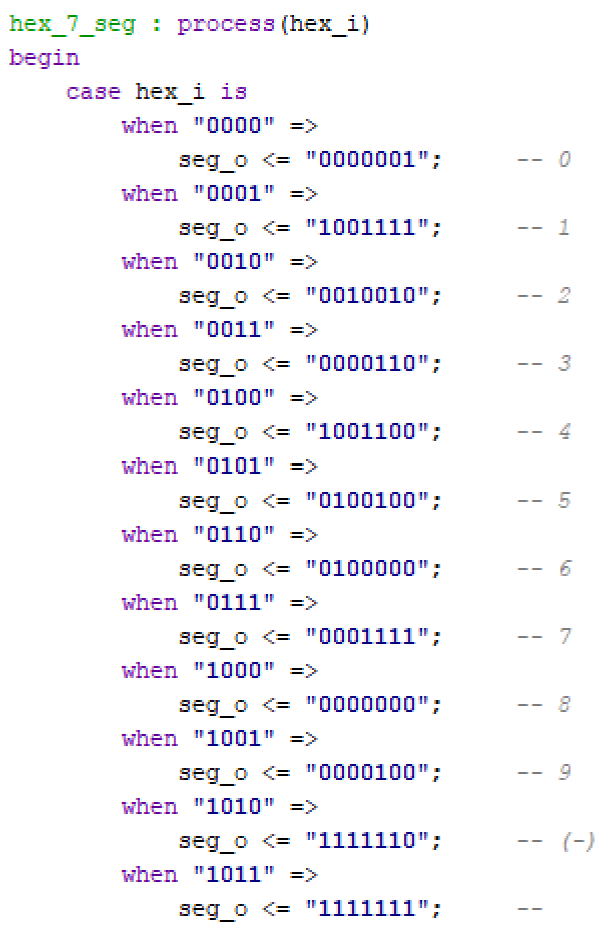
Piezzo beeper



```
piezo_o <= s_beep when (mode_i = "01") else
   s_tone when (mode_i = "10") else
   '0'    when (mode_i = "00");
```

A two-bit number describes whether the beeper will beep, emit a constant tone, or be off.

Display



```
hex_7_seg : 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 <= "1111110";     -- (-)
        when "1011" =>
            seg_o <= "1111111";     --
```

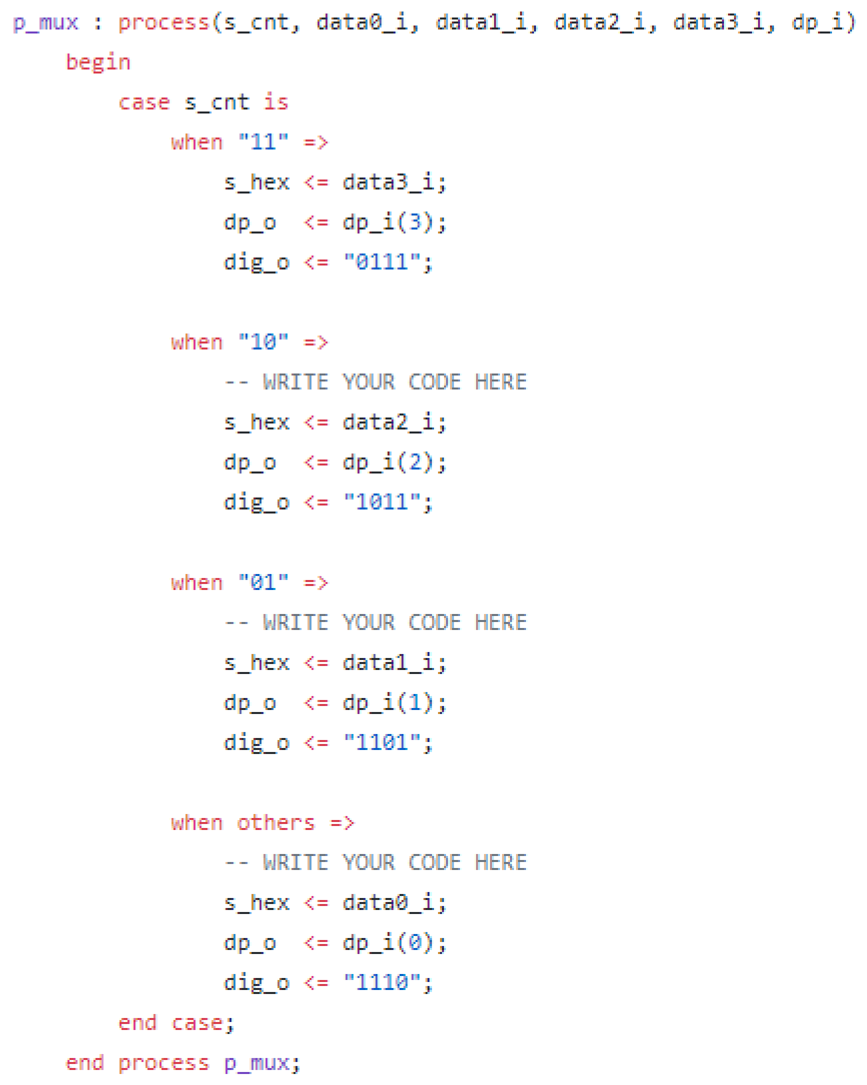
According to the four-bit input value, the values corresponding to this input will appear on the display.

Clock Divider

```vhdl
architecture Behavioral of clock_divider is
    signal cnt : integer := 1;
    signal tmp : std_logic := '0';
begin
    p_clock_divider : process(clk,rst)
    begin
        if rising_edge(clk) then
            if (rst = '1') then
                cnt <= 1;
                tmp <= '0';
            else
                cnt <= cnt + 1;
                if (cnt = g_CYCLES) then
                    tmp <= NOT tmp;
                    cnt <= 1;
                end if;
            end if;
        end if;
        clk_out <= tmp;
    end process;
```

The clock divider has the task of adding "1" at each leading edge, if the reset in "1" cnt remains at the value in which it was.

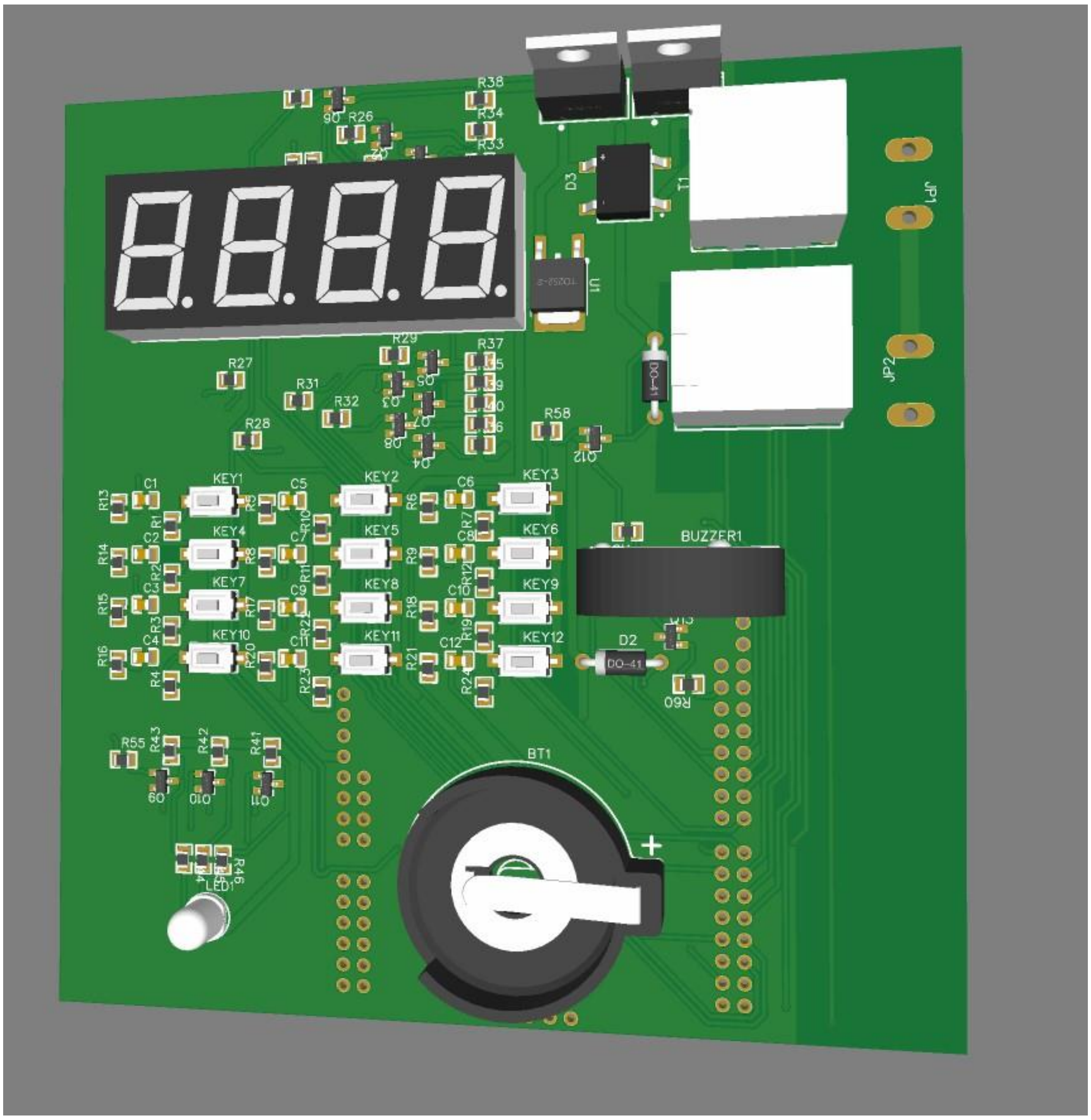
Driver of 7-segment / 4 digits display

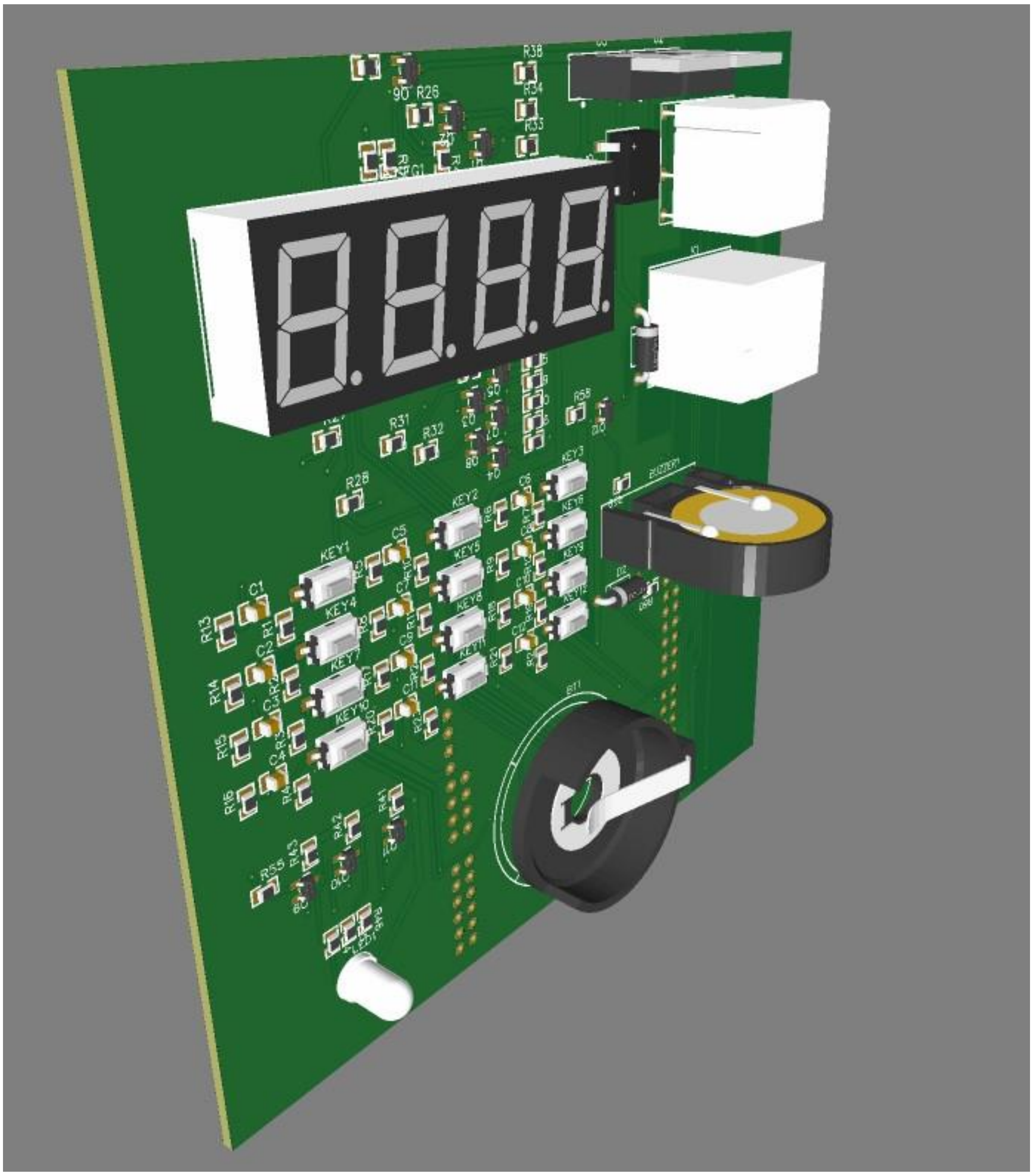


```
p_mux : process(s_cnt, data0_i, data1_i, data2_i, data3_i, dp_i)
    begin
        case s_cnt is
            when "11" =>
                s_hex <= data3_i;
                dp_o  <= dp_i(3);
                dig_o <= "0111";

            when "10" =>
                -- WRITE YOUR CODE HERE
                s_hex <= data2_i;
                dp_o  <= dp_i(2);
                dig_o <= "1011";

            when "01" =>
                -- WRITE YOUR CODE HERE
                s_hex <= data1_i;
                dp_o  <= dp_i(1);
                dig_o <= "1101";

            when others =>
                -- WRITE YOUR CODE HERE
                s_hex <= data0_i;
                dp_o  <= dp_i(0);
                dig_o <= "1110";
        end case;
    end process p_mux;
```

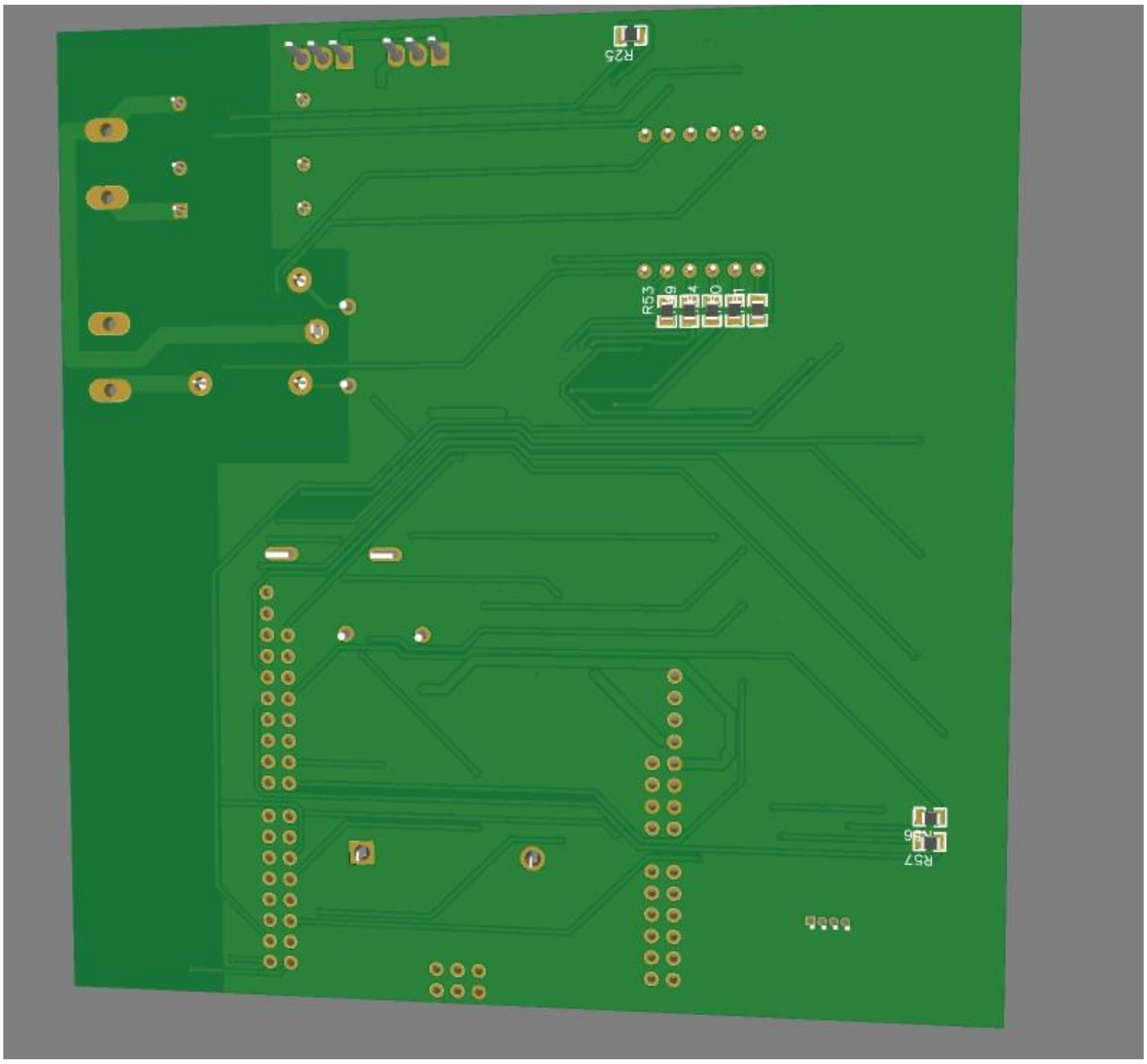
The display here recognizes in which position it will write the value we entered.

View of the final connection of components to the board





KEY1
KEY2
KEY3
KEY4
KEY5
KEY6
KEY7
KEY8
KEY9
KEY10
KEY11
KEY12
BUZZER
BT1


R25
R53
R56
R57

Electronic Door Lock 12 V DC

The relay is active for a certain time, when the door is unlocked, the relay is deactivated and the LED lights up green. The beep will emit a constant tone. This is a sample of the component that is used on our version of the project.

Code location

https://github.com/xbarto0c/Digital-electronics-1/tree/main/Labs/Project_code_lock

Readme file