Car parking system

Lab 1

Matteo Golinelli, Nicolò Vantini, Marcus Gregory

12 November 2025

1 Introduction

In this project a car parking system is designed from finite state machine to a VHDL implementation. The system is designed to control the traffic entering a car park with a password and sensor input along with LED and 8 segment display output.

2 Assumptions

For this project many assumptions had to be made, given the project specifications, from the inputs to the function of the device.

2.1 Ideal Drivers

In most car parks a gate stops cars from entering and exiting without authorization. However, here we assume that there is no gate and drivers respect the sign telling them not to drive without further question.

2.2 Password Input

We are given the following strict entity declaration to follow:

```
entity Car_Parking_System_VHDL is
    port(
        clk : in std_logic;
        reset_n : in std_logic; -- Active low asynchronus
        front_sensor : in std_logic;
        back_sensor : in std_logic;
        password_1 : in std_logic_vector(1 downto 0); -- Correct "01"
        password_2 : in std_logic_vector(1 downto 0); -- Correct "10"
        pswd_in : in std_logic; -- Flag (1 when the password is set by the user)
        GREEN_LED : out std_logic;
        RED_LED : out std_logic;
        HEX_1 : out std_logic_vector(6 downto 0); -- Segments gfedcba
        HEX_2 : out std_logic_vector(6 downto 0); -- Segments gfedcba
        car_count : out std_logic_vector(6 downto 0) -- Car counter
    );
end Car_Parking_System_VHDL;
```

2.3 External Input Circuit

The signals password_1 and password_2 are one of 4 2-bit inputs sourced from 4 buttons; ”A”, ”B”, ”C” & ”D”. However, the specifications say that there are only 4 buttons. This begs the question, how does the input port know that there are 2 password words if there are only 4 buttons. Thus we must assume that there is another circuit which collects button inputs and outputs them sequentially. We also assume that this device resets whenever the system is in idle. This would stop the third press of the last car becoming the first of the next. Moreover, this device outputs a signal pswd_in which is equal to 1 for a small amount of time when the password is set by the user.

2.4 Waiting time

To make the project more realistic, we introduced timing delays so the system can buffer and display outputs for a noticeable duration. Without these delays, components like the 7-segment displays would only show results for a single clock cycle. Specifically, we implemented:

- A 10-clock-cycle waiting time for password entry, as required by the specification.
- Additional 3 clock cycles to display the "wrong password" message on the LCDs.
- A 10-clock-cycle timeout state if no password has been entered during the waiting time.

2.5 IDLE state

In order to reset the system after a password is set, the system is moved to the IDLE state. So, when the password is correct and the car is detected at the back sensor, the system goes back to the IDLE state. If multiple cars are detected, the system transitions to the STOP state and wait for the car inside the park to move before changing to IDLE and then WAIT PASSWORD.
In this way, the system always loses a clock cycle to reset.

2.6 LED blinking

In order to meet the requirements of a blinking LED with same frequency as the clock, we added to the output process a condition on the falling edge of the clock that updates the LED blinking. Without this condition, the LED would have been updated only at the rising edge of the clock, thus making it blinks at half the clock frequency.

2.7 LED prescaler

The specifications also required to implement a LED prescaler to slow the LED blink to 1 Hz instead of at clock frequency. While the implementation is straightforward, we did not inserted it into our code because of the huge difference between the testbench clock frequency of 50 MHz and the required prescaler frequency of 1 Hz.
Below, we report the steps to implement the prescaler for the LED blink rate.

- Add the following signals to the architecture:

```
signal scaled_clock : std_logic := '0';
signal clock_counter : integer := 0;
```

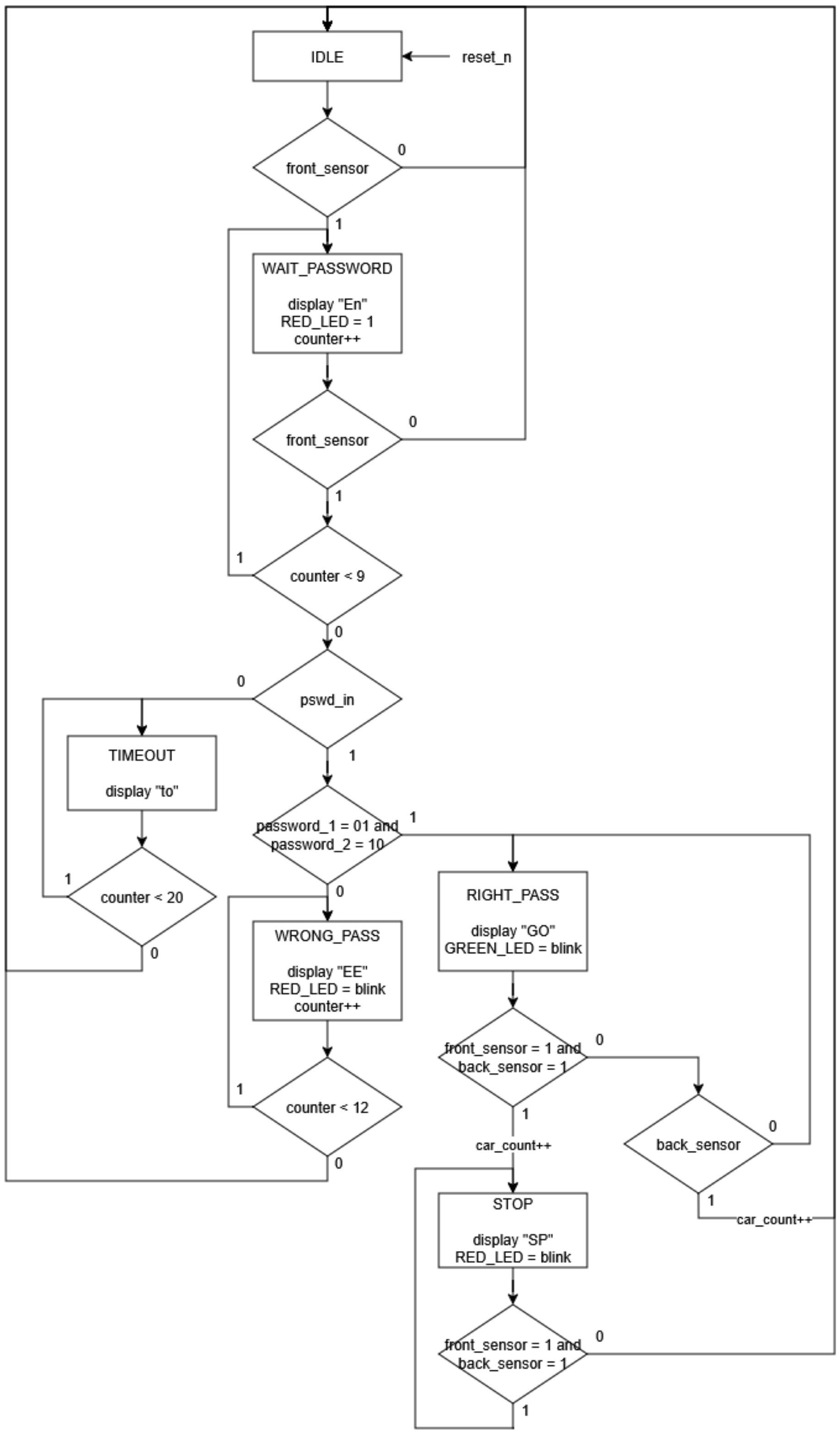
- Add the following code inside the architecture:

```
-- Prescale the clock from 50 MHz to 1 Hz
clock_prescale : process(clk)
begin
    if rising_edge(clk) then
        if clock_counter < 25000000 then
            clock_counter <= clock_counter + 1;
        else
            clock_counter <= 0;
            scaled_clock <= not scaled_clock;
        end if;
    end if;
end process clock_prescale;
```

- Remove the conditions on the falling edge of the clock in the output_values process.
- Inside the output_values process, change the LED signals which are set to clk and set them to scaled_clock:

```
RED_LED <= scaled_clock; -- Before it was RED_LED <= clk
GREEN_LED <= scaled_clock; -- Before it was GREEN_LED <= clk
```

3 FSM design


IDLE
reset_n
front_sensor
0
1
WAIT_PASSWORD
display "En"
RED_LED = 1
counter++
front_sensor
0
1
counter < 9
1
0
pswd_in
0
1
TIMEOUT
display "to"
counter < 20
1
0
password_1 = 01 and password_2 = 10
1
0
RIGHT_PASS
display "GO"
GREEN_LED = blink
WRONG_PASS
display "EE"
RED_LED = blink
counter++
counter < 12
1
0
front_sensor = 1 and back_sensor = 1
0
1
car_count++
back_sensor
0
1
car_count++
STOP
display "SP"
RED_LED = blink
front_sensor = 1 and back_sensor = 1
0
1

4 Results

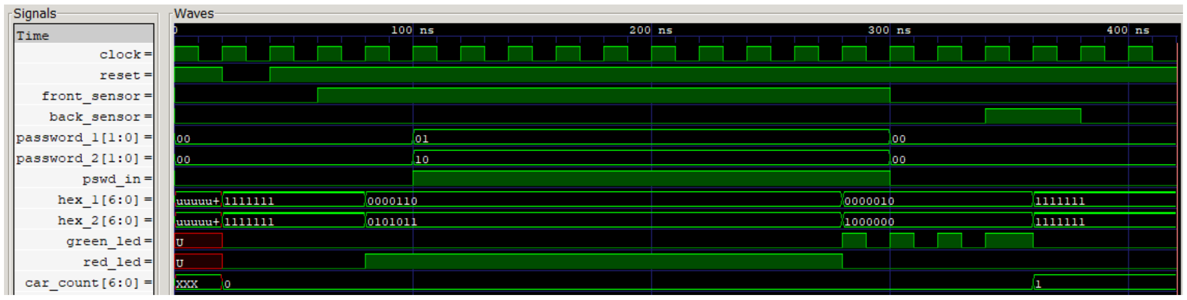


Figure 1: Normal Entry

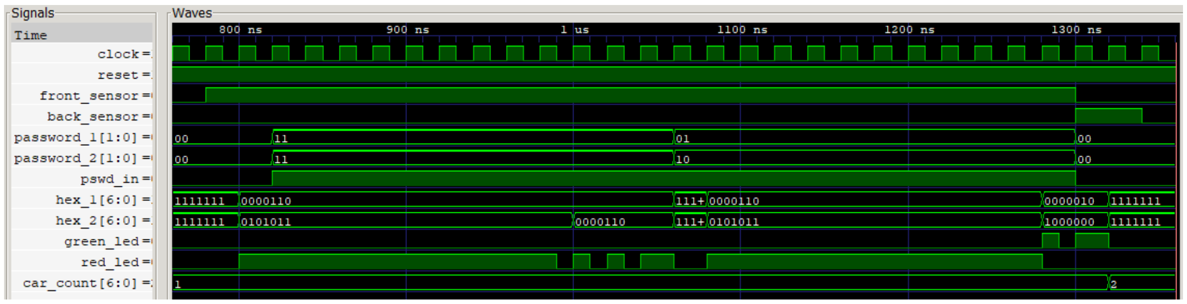


Figure 2: Wrong Password

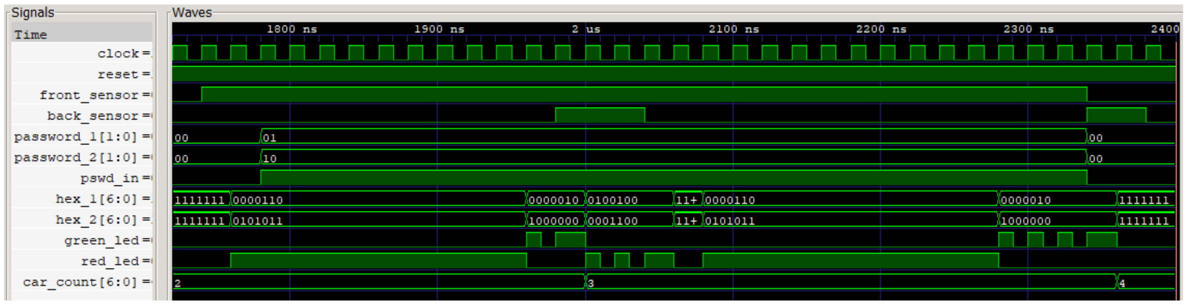


Figure 3: Multiple Cars

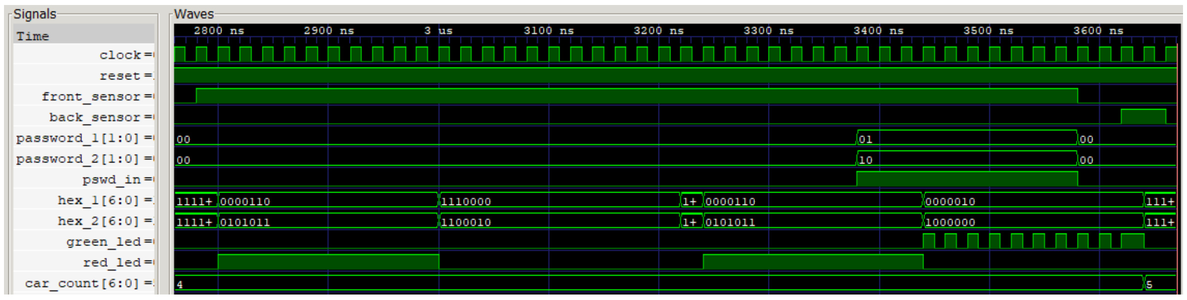


Figure 4: Timeout

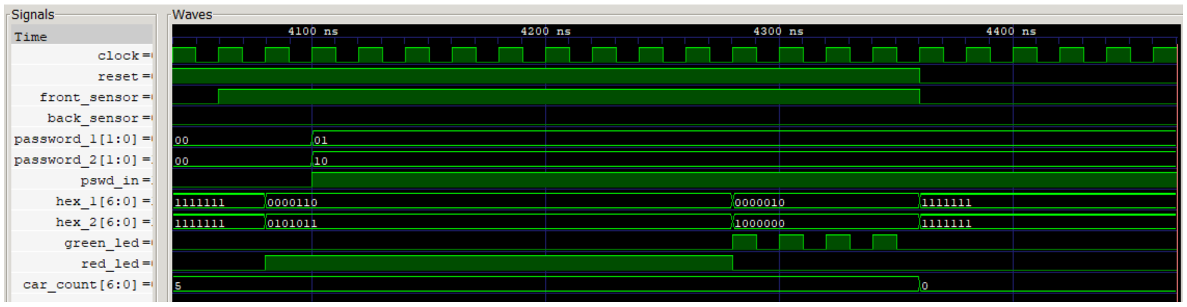


Figure 5: Reset Input

5 Car Parking System VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Car_Parking_System_VHDL is
    port(
        clk : in std_logic;
        reset_n : in std_logic; -- Active low asynchronus
        front_sensor : in std_logic;
        back_sensor : in std_logic;
        password_1 : in std_logic_vector(1 downto 0); -- Correct "01"
        password_2 : in std_logic_vector(1 downto 0); -- Correct "10"
        pswd_in : in std_logic; -- Flag (1 when the password is set by the user)
        GREEN_LED : out std_logic;
        RED_LED : out std_logic;
        HEX_1 : out std_logic_vector(6 downto 0); -- Segments gfedcba
        HEX_2 : out std_logic_vector(6 downto 0); -- Segments gfedcba
        car_count : out std_logic_vector(6 downto 0) -- Car counter
    );
end Car_Parking_System_VHDL;

architecture behaviour of Car_Parking_System_VHDL is
    type FSM_States is (IDLE, WAIT_PASSWORD, WRONG_PASS, RIGHT_PASS, STOP, TIMEOUT);
    ↪   -- States for the FSM machine
    signal current_state : FSM_States;

    signal counter : std_logic_vector(4 downto 0) := "00000";
    signal internal_car_count : std_logic_vector(6 downto 0) := (others => '0'); --
    ↪   Counter for the cars

begin

    -- Update the current state based on the inputs
    compute_current_state : process(clk, reset_n)
    begin
        if reset_n = '0' then -- Reset condition
            current_state <= IDLE;
            internal_car_count <=  (others => '0');

        elsif rising_edge(clk) then
            current_state <= IDLE; -- Set default state

            case current_state is
                when IDLE =>
                    if front_sensor = '1' then
                        current_state <= WAIT_PASSWORD;
                    end if;

                when WAIT_PASSWORD =>
                    if front_sensor = '1' then
                        if counter < "01001" then -- Counter < 9
                        current_state <= WAIT_PASSWORD;

                    elsif pswd_in = '0' then -- The password has been inputted
                        current_state <= TIMEOUT;

                    elsif (password_1 & password_2) = "0110" then -- Right
                    ↪   password
                        current_state <= RIGHT_PASS;

                    else
                        current_state <= WRONG_PASS;
                    end if;
                end if;

            when WRONG_PASS =>
                if counter < "01100" then -- Counter < 12, used to display a bit
                ↪   longer the state
                    current_state <= WRONG_PASS;
                end if;

            when RIGHT_PASS =>
                if front_sensor = '1' and back_sensor = '1' then
                    current_state <= STOP;
                    internal_car_count <=
                    ↪   std_logic_vector(unsigned(internal_car_count) + 1); --
                    ↪   Increase car counter by 1

                elsif back_sensor = '0' then -- Wait when the car is passed
                    current_state <= RIGHT_PASS;

                else
                    internal_car_count <=
                    ↪   std_logic_vector(unsigned(internal_car_count) + 1); --
                    ↪   Increase car counter by 1
                end if;

            when STOP =>
                if front_sensor = '1' and back_sensor = '1' then
                    current_state <= STOP;
                end if;

            when TIMEOUT =>
                if counter < "10100" then -- Counter < 20
                    current_state <= TIMEOUT;
                end if;
        end case;

    end if;
end process compute_current_state;


-- Update the output values at the following clock
output_values : process(clk, reset_n)
```

```
begin
    if reset_n = '0' then -- Reset status
        GREEN_LED <= '0';
        RED_LED <= '0';
        HEX_1 <= (others => '1');
        HEX_2 <= (others => '1');
        car_count <= (others => '0');
        counter <= "00000";

    elsif rising_edge(clk) then
        -- Default values
        GREEN_LED <= '0';
        RED_LED <= '0';
        HEX_1 <= (others => '1');
        HEX_2 <= (others => '1');
        counter <= "00000";
        car_count <= internal_car_count;

        case current_state is
            when WAIT_PASSWORD =>
                counter <= std_logic_vector(unsigned(counter) + 1); -- Increase
                 ↪  counter by 1
                RED_LED <= '1';
                HEX_1 <= "0000110"; -- E
                HEX_2 <= "0101011"; -- n

            when WRONG_PASS =>
                counter <= std_logic_vector(unsigned(counter) + 1); -- Increase
                 ↪  counter by 1
                RED_LED <= clk;
                HEX_1 <= "0000110"; -- E
                HEX_2 <= "0000110"; -- E

            when RIGHT_PASS =>
                GREEN_LED <= clk;
                HEX_1 <= "0000010"; -- G
                HEX_2 <= "1000000"; -- O

            when STOP =>
                RED_LED <= clk;
                HEX_1 <= "0100100"; -- S
                HEX_2 <= "0001100"; -- P

            when TIMEOUT =>
                counter <= std_logic_vector(unsigned(counter) + 1); -- Increase
                 ↪  counter by 1
                HEX_1 <= "1110000"; -- t
                HEX_2 <= "1100010"; -- o

            when others =>
                null;
        end case;
```

```vhdl
        elsif falling_edge(clk) then
            -- Continues the LED blinking outside the rising edges of the clock
            case current_state is
                when WRONG_PASS =>
                    RED_LED <= clk;

                when RIGHT_PASS =>
                    GREEN_LED <= clk;

                when STOP =>
                    RED_LED <= clk;

                when others =>
                    null;
            end case;

        end if;
    end process output_values;

end architecture behaviour;
```

6 Testbench

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb is
end entity tb;

architecture behaviour of tb is

    component Car_Parking_System_VHDL is
        port(
            clk : in std_logic;
            reset_n : in std_logic;
            front_sensor : in std_logic;
            back_sensor : in std_logic;
            password_1 : in std_logic_vector(1 downto 0);
            password_2 : in std_logic_vector(1 downto 0);
            pswd_in : in std_logic;
            GREEN_LED : out std_logic;
            RED_LED : out std_logic;
            HEX_1 : out std_logic_vector(6 downto 0);
            HEX_2 : out std_logic_vector(6 downto 0);
            car_count : out std_logic_vector(6 downto 0)
        );
    end component Car_Parking_System_VHDL;

    constant clock_period : time := 20 ns;

    signal clock : std_logic := '1';
    signal reset : std_logic := '1';
    signal front_sensor : std_logic := '0';
    signal back_sensor : std_logic := '0';
    signal password_1 : std_logic_vector(1 downto 0) := "00";
    signal password_2 : std_logic_vector(1 downto 0) := "00";
    signal pswd_in : std_logic := '0';
    signal GREEN_LED : std_logic;
    signal RED_LED : std_logic;
    signal HEX_1 : std_logic_vector(6 downto 0);
    signal HEX_2 : std_logic_vector(6 downto 0);
    signal car_count : std_logic_vector(6 downto 0);

begin

    DUT : Car_Parking_System_VHDL port map( -- Device under test
        clk => clock,
        reset_n => reset,
        front_sensor => front_sensor,
        back_sensor => back_sensor,
        password_1 => password_1,
        password_2 => password_2,
        pswd_in => pswd_in,
        GREEN_LED => GREEN_LED,
        RED_LED => RED_LED,
        HEX_1 => HEX_1,
        HEX_2 => HEX_2,
        car_count => car_count
    );

    -- Define the signal clock
    clock_process : process
    begin
        wait for clock_period / 2;
        clock <= not clock;
    end process clock_process;

    -- Testbench process
    main_tb : process
    begin
        -- Initial reset
        wait for clock_period;
        reset <= '0';
        wait for clock_period;
        reset <=  '1';
        wait for clock_period;

        -- Normal entry
        front_sensor <=  '1'; -- Car approaches
        wait for 2*clock_period;
        password_1 <= "01";
        password_2 <= "10";
        pswd_in <= '1'; -- The password is set
        wait for 10*clock_period;
        front_sensor <= '0'; -- The car starts moving
        password_1 <= "00";
        password_2 <= "00";
        pswd_in <= '0'; -- The password is reset
        wait for 2*clock_period;
        back_sensor <= '1'; -- The car is now inside the parking
        wait for 2*clock_period;
        back_sensor <= '0'; -- The car leaves the parking entrance

        wait for 20*clock_period;

        -- Wrong password
        front_sensor <= '1'; -- Car approaches
        wait for 2*clock_period;
        password_1 <= "11";
        password_2 <= "11";
        pswd_in <= '1'; -- The wrong password is set
        wait for 12*clock_period;
        password_1 <= "01";
        password_2 <= "10";
        pswd_in <= '1'; -- The right password is set
        wait for 12*clock_period;
```

```vhdl
front_sensor <= '0';
back_sensor <= '1'; -- The car moves inside
password_1 <= "00";
password_2 <= "00";
pswd_in <= '0'; -- The password is reset
wait for 2*clock_period;
back_sensor <= '0'; -- The car leaves the parking entrance

wait for 20*clock_period;

-- Multiple cars
front_sensor <=  '1'; -- Car approaches
wait for 2*clock_period;
password_1 <= "01";
password_2 <= "10";
pswd_in <= '1'; -- The password is set
wait for 10*clock_period;
back_sensor <= '1'; -- The first car enters, but another one follows
wait for 3*clock_period;
back_sensor <= '0'; -- The first car leaves the parking entrance
wait for 15*clock_period;
front_sensor <= '0';
back_sensor <= '1'; -- The second car enters
password_1 <= "00";
password_2 <= "00";
pswd_in <= '0'; -- The password is reset
wait for 2*clock_period;
back_sensor <= '0'; -- The second car leaves the parking entrance

wait for 20*clock_period;

-- Timeout
front_sensor <=  '1'; -- Car approaches
wait for 30*clock_period; -- The password is not set in time
password_1 <= "01";
password_2 <= "10";
pswd_in <= '1'; -- The password is set
wait for 10*clock_period;
front_sensor <= '0'; -- The car starts moving
password_1 <= "00";
password_2 <= "00";
pswd_in <= '0'; -- The password is reset
wait for 2*clock_period;
back_sensor <= '1'; -- The car is now inside the parking
wait for 2*clock_period;
back_sensor <= '0'; -- The car leaves the parking entrance

wait for 20*clock_period;

-- Reset
front_sensor <=  '1'; -- Car approaches
wait for 2*clock_period;
password_1 <= "01";
```

```
        password_2 <= "10";
        pswd_in <= '1'; -- The password is set
        wait for 13*clock_period;
        front_sensor <= '0';
        reset <= '0'; -- Reset value is given
        wait for 3*clock_period;

        wait;

    end process main_tb;

end architecture behaviour;
```