FPGA VHDL course final project

Car Parking System

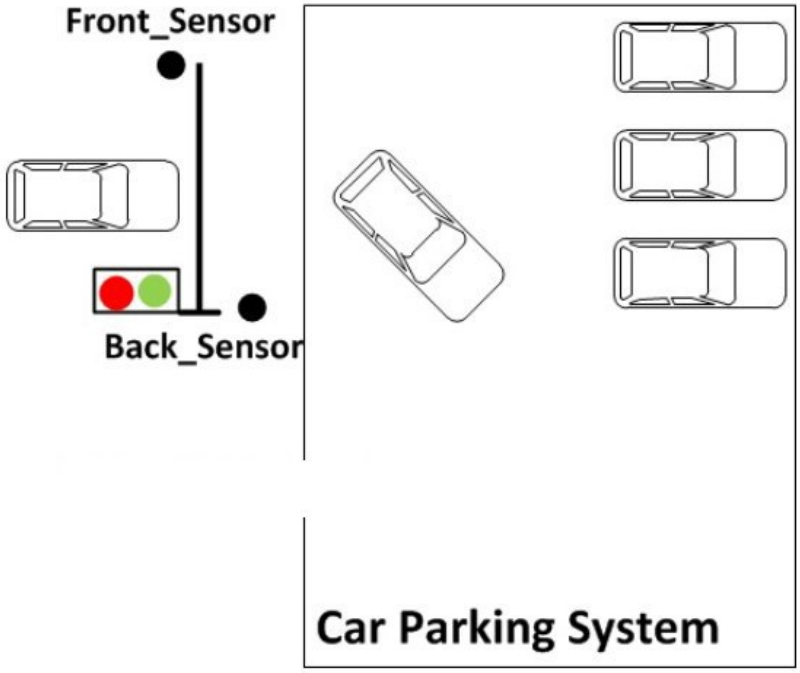
Student: Hossein Soltani
Instructor: Dr. Farhad Pouladi
TA: Arash Rezaee

Introduction

The purpose of this project is **to design and implement a car parking system using FPGA**. The system will control the entrance of the parking lot by prompting each car approaching the gate for a password.

After the correct password is entered, the gate will open and the car will be allowed to enter. If the wrong password is entered, the car will have to try again. Once the car has entered the parking lot, the gate will close and the system will be ready to serve the next car.

Project Description

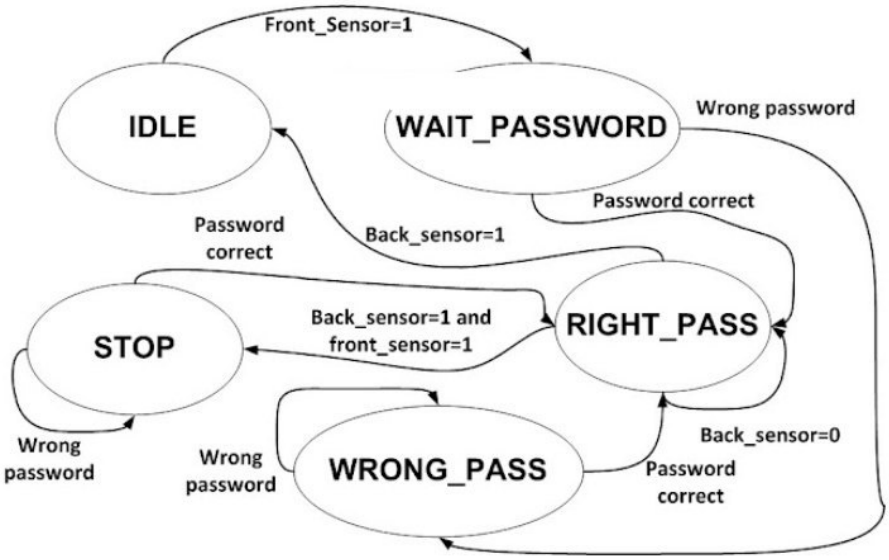


The VHDL code written for the project is designed **to control the gate and password prompt process**.
The system will be activated when **a car approaches the gate** and will prompt the driver to enter a **4-digit password**.
If the password is correct, the gate will open and the car will be allowed to enter.
If the password is incorrect, the driver will have to try again.

Two sensors are being used for detecting cars.
One is **the front sensor** and the other one is **the back sensor**.
The first one is used for **detecting cars approaching the gate** (outside of the parking lot).
The latter is used for **detecting cars that passed the gate and are inside of the parking lot**.

Design and Implementation

For implementing such a system, I used Moore FSM (finite state machine) as the model of the system states.
This is the designed FSM:



We have 5 states for the FSM model of the system:

IDLE

-> The state for when the system is at its starting state.
At this state, the system waits until a car approaches and the front_sensor detects that.
When the front_sensor detects that, the state will be changed to **WAIT_PASSWORD** state.

WAIT_PASSWORD

-> At this state, the system is waiting for the user to enter its password.
If the user enters the password correctly, the system will go to **RIGHT_PASS** state.
Otherwise it'll go into **WRONG_PASS** state.

RIGHT_PASS

-> When the entered password at **WAIT_PASSWORD** was correct, the system will come into this state.
The gate will be opened to allow the car to get into the parking lot.
Also, a green led will blink repeatedly.

At this state, if the **back_sensor** was on (1), meaning that the car already **got into the parking lot**, it'll go into **IDLE** state. So the system will be ready to serve another car.
Otherwise, if the **back_sensor** was off (0), the system would stay in that state. That means the car is still in front of the parking lot gate and hasn't gone in yet.

WRONG_PASS
-> When the entered password at **WAIT_PASSWORD** was correct, the system will come into this state.
A red led will blink repeatedly.
The gate will also be closed.

At this state, the system prompts the user to get its password. It'll continue to be in this state until the user enters the correct password.
When the correct password is entered, the system will go into **RIGHT_PASS** state.

STOP
-> When a car already entered the parking lot (entered the correct password) and **it's near the gate (probably crossing it),**
but also there's another car approaching the gate for entering.
At this state, the system prompts the new car that is in front of the gate for the password.
And waits until it enters the correct password. Just like **WRONG_PASS** state.

But also keeps the gate open because the back_sensor is on (1), and is indicating that that old car is still close to the gate and the gate should be still.

The code

I used the **modular approach** in my project. The code is parted into these files:

-- car_parking_system_tb.vhd

test bench file.

-- car_parking_system.vhd

main design, contains the main entity and architecture codes.

-- states.vhd

a package that contains the code of different states.

-- types.vhd

a package that contains **custom types** and also **constants**.

custom types:
password_t -> password type; An array of integers with the length of 4.
states_t -> FSM states; IDLE, WAIT_PASSWORD, RIGHT_PASS, WRONG_PASS, STOP.

-- utils.vhd

some useful functions that are used across the code.

-- car_parking_system.vhd --

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use work.types.all;
use work.states.all;


entity car_parking_system is
  port (
    front_sensor : in std_logic;
    back_sensor : in std_logic;
```

```
    green_led : inout std_logic;
    red_led : inout std_logic;

    password_in : in integer range 0 to 9; -- can be BCD numbers
    gate_out : out std_logic; -- gate_out = '0' => gate_close , gate_out =
'1' => gate_open

    clk : in std_logic;
    rst : in std_logic

  );
end car_parking_system;

architecture behav of car_parking_system is
  shared variable cur_state : states_t := IDLE;
  signal prev_state : states_t := IDLE;

begin
  process (clk, rst)
  begin
    if (rst'event and rst = '0') then
      prev_state <= IDLE;
    elsif (rising_edge(clk)) then
      prev_state <= cur_state;
    end if;
  end process;

  main_proc: process (prev_state, green_led, red_led, password_in,
front_sensor, back_sensor)
    variable wait_password_state_counter : integer range 0 to 4 := 0;
    variable entered_password : password_t := (others => 0);

  begin
      case prev_state is
        when IDLE =>
          idle_state(wait_password_state_counter, entered_password,
green_led, red_led, gate_out, front_sensor, cur_state);

        when WAIT_PASSWORD =>
```

```
          wait_password_state(password_in, wait_password_state_counter,
entered_password, cur_state);

        when RIGHT_PASS =>
          right_password_state(front_sensor, back_sensor, green_led,
red_led, gate_out, cur_state);

        when WRONG_PASS =>
          wrong_password_state(entered_password, green_led, red_led,
password_in, gate_out, cur_state);

        when STOP =>
          stop_state(entered_password, cur_state, green_led, red_led,
password_in);

       end case;
  end process;
end behav;
```

-- states.vhd --

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use work.types.all;
use work.utils.all;

package states is
  procedure idle_state(
    variable wait_password_state_counter : out integer range 0 to 4;
    variable entered_password            : out password_t;
    signal green_led                     : inout std_logic;
    signal red_led                       : inout std_logic;
    signal gate_out                      : out std_logic;
    signal front_sensor                  : in std_logic;
    variable cur_state                   : out states_t
  );

  procedure wait_password_state(
    signal password_in                   : in integer range 0 to 9;

    variable wait_password_state_counter : inout integer range 0 to 4;
    variable entered_password            : inout password_t;

    variable cur_state                   : out states_t
  );

  procedure right_password_state(
    signal front_sensor                  : in std_logic;
    signal back_sensor                   : in std_logic;

    signal green_led                     : inout std_logic;
    signal red_led                       : inout std_logic;
    signal gate_out                      : out std_logic;
    variable cur_state                   : out states_t
  );

  procedure wrong_password_state(
    variable entered_password            : inout password_t;
    signal green_led                     : inout std_logic;
```

```
    signal red_led                    : inout std_logic;
    signal password_in                    : in integer range 0 to 9;
    signal gate_out                       : out std_logic;

    variable cur_state                    : inout states_t
  );

  procedure stop_state(
    variable entered_password           : inout password_t;
    variable cur_state                  : inout states_t;
    signal green_led                    : inout std_logic;
    signal red_led                      : inout std_logic;
    signal password_in                  : in integer range 0 to 9
  );
end package;

package body states is
  procedure idle_state(
    variable wait_password_state_counter : out integer range 0 to 4;
    variable entered_password           : out password_t;
    signal green_led                    : inout std_logic;
    signal red_led                      : inout std_logic;
    signal gate_out                       : out std_logic;
    signal front_sensor                 : in std_logic;
    variable cur_state                  : out states_t
  ) is
  begin
    if (front_sensor'event and front_sensor = '1') then
      cur_state := WAIT_PASSWORD;
    end if;

    wait_password_state_counter := 0;
    entered_password := (others => 0);

    green_led <= '0';
    red_led <= '0';
    gate_out <= '0';
  end idle_state;

  procedure wait_password_state(
```

```
    signal password_in                 : in integer range 0 to 9;

    variable  wait_password_state_counter : inout integer range 0 to 4;
    variable entered_password           : inout password_t;

    variable cur_state                  : out states_t
  ) is
  begin
    if (wait_password_state_counter < WAIT_PASSWORD_STATE_MAX_COUNTER)
then
      entered_password(wait_password_state_counter) := password_in;
      wait_password_state_counter := wait_password_state_counter + 1;

    else
      -- verify password
      if (verify_password(CORRECT_PASSWORD, entered_password)) then -- ?
is this kind of condtiion gonna work?

        cur_state := RIGHT_PASS;
      else
        cur_state := WRONG_PASS;
      end if;
    end if;
  end wait_password_state;

  procedure right_password_state(
    signal front_sensor                : in std_logic;
    signal back_sensor                 : in std_logic;

    signal green_led                   : inout std_logic;
    signal red_led                    : inout std_logic;
    signal gate_out                    : out std_logic;
    variable cur_state                   : out states_t
  ) is
  begin
    gate_out <= '1';

    green_led <= not green_led;
    red_led <= '0';
```

```
    if (back_sensor = '1') then
      if (front_sensor = '1') then
        cur_state := STOP;
      else
        cur_state := IDLE;
      end if;
    else
      cur_state := RIGHT_PASS; -- don't change the state
    end if;
  end right_password_state;

  procedure wrong_password_state(
    variable entered_password             : inout password_t;
    signal green_led                    : inout std_logic;
    signal red_led                     : inout std_logic;
    signal password_in                    : in integer range 0 to 9;
    signal gate_out                       : out std_logic;

    variable cur_state                    : inout states_t
  ) is
  begin
    gate_out <= '0';

    green_led <= '0';
    red_led <= not red_led; -- red led blinking

    entered_password := password_left_shift_insert(entered_password,
password_in);

    -- verify password
    if (verify_password(CORRECT_PASSWORD, entered_password)) then
      cur_state := RIGHT_PASS;
    else
      cur_state := WRONG_PASS;
    end if;
  end wrong_password_state;

  procedure stop_state(
    variable entered_password            : inout password_t;
    variable cur_state                   : inout states_t;
```

```
    signal green_led                    : inout std_logic;
    signal red_led                     : inout std_logic;
    signal password_in                  : in integer range 0 to 9
  ) is
  begin
    green_led <= '0';
    red_led <= not red_led; -- red led blinking

    entered_password := password_left_shift_insert(entered_password,
password_in);

    -- verify password
    if (verify_password(CORRECT_PASSWORD, entered_password)) then
      cur_state := RIGHT_PASS;
    else
      cur_state := STOP;
    end if;
  end stop_state;

end states;
```

-- types.vhd --

```
package types is
 type password_t is array (0 to 3) of integer range 0 to 9; -- this will
store the entered password/PIN
 type states_t is (
   IDLE, WAIT_PASSWORD, RIGHT_PASS, WRONG_PASS, STOP
 );
  constant WAIT_PASSWORD_STATE_MAX_COUNTER : integer range 0 to 4 := 4; --
wait 4 cylces -> user must enter a number at each cycle ==> we have 4
digits password/PIN
 constant CORRECT_PASSWORD: password_t := (1, 2, 3, 4);
end types;
```

-- utils.vhd --

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

use work.types.all;

package utils is
 function verify_password(correct_password: password_t; password:
password_t) return boolean;
 function password_left_shift_insert(password: password_t; password_in:
integer range 0 to 9) return password_t;

end utils;

package body utils is
 function verify_password(correct_password: password_t; password:
password_t)
 return boolean is
 begin
   for i in correct_password'range loop
     if (correct_password(i) /= password(i)) then
       return false;
     end if;
   end loop;

   return true;
 end verify_password;

 function password_left_shift_insert(password: password_t; password_in:
integer range 0 to 9)
 return password_t is
 variable returning_password: password_t := (others => 0);
 begin
   returning_password(0) := password(1);
   returning_password(1) := password(2);
   returning_password(2) := password(3);
   returning_password(3) := password_in;

   return returning_password;
 end password_left_shift_insert;
```

```
end utils;
```

-- car_parking_system_tb.vhd --

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity car_parking_system_tb is
end car_parking_system_tb;

architecture test_bench of car_parking_system_tb is

   component car_parking_system is
   port (
       front_sensor : in std_logic;
       back_sensor : in std_logic;

       green_led : inout std_logic;
       red_led : inout  std_logic;

       password_in : in integer range 0 to 9; -- can be BCD numbers
       gate_out : out std_logic; -- gate_out = '0' => gate_close ,
gate_out = '1' => gate_open
       clk : in std_logic;
       rst : in std_logic
   );
   end component;

   signal front_sensor : std_logic := '0';
   signal back_sensor : std_logic := '0';

   signal green_led : std_logic := '0';
   signal red_led : std_logic := '0';

   signal password_in : integer range 0 to 9 := 0; -- can be BCD numbers
   signal gate_out : std_logic := '0'; -- gate_out = '0' => gate_close ,
gate_out = '1' => gate_open

   signal clk : std_logic := '0';
```

```
    signal rst : std_logic := '1';

    constant CLK_PERIOD : time := 100 ns;
    signal finished : std_logic := '0';

begin
    dut: car_parking_system
    port map (
        front_sensor => front_sensor,
        back_sensor => back_sensor,

        green_led => green_led,
        red_led => red_led,

        password_in => password_in,
        gate_out => gate_out,

        clk => clk,
        rst => rst
    );


    clk_process: process
    begin
        while (finished = '0') loop
            clk <= '1';
            wait for CLK_PERIOD / 2;

            clk <= '0';
            wait for CLK_PERIOD / 2;
        end loop;
    end process;

    stimulus: process
    begin
        wait for CLK_PERIOD;

        -- case_1: IDLE --(front_sensor = 1)--> WAIT_PASSWORD
--password_wrong--> WRONG_PASS --password_wrong--> WRONG_PASS
        front_sensor <= '1';
```

```
        wait for CLK_PERIOD;
        password_in <= 0, 0 after CLK_PERIOD, 3 after 2*CLK_PERIOD, 4 after
3*CLK_PERIOD;
        wait for 6*CLK_PERIOD;
        password_in <= 0, 5 after CLK_PERIOD, 3 after 2*CLK_PERIOD, 7 after
3*CLK_PERIOD;
        wait for 6*CLK_PERIOD;

        -- case_2: IDLE --(front_sensor = 1)--> WAIT_PASSWORD
--password_wrong--> WRONG_PASS --password_correct--> RIGHT_PASS
        password_in <= 1, 2 after CLK_PERIOD, 3 after 2*CLK_PERIOD, 4 after
3*CLK_PERIOD;
        wait for 6*CLK_PERIOD;

        -- case_3: IDLE --(front_sensor = 1)--> WAIT_PASSWORD
--password_correct--> RIGHT_PASS --(back_sensor = 0)--> RIGHT_PASS
        password_in <= 1, 2 after CLK_PERIOD, 3 after 2*CLK_PERIOD, 4 after
3*CLK_PERIOD;
        wait for 6*CLK_PERIOD;

        -- case_4: RIGHT_PASS --(back_sensor = 1 and front_sensor = 1)-->
STOP
        front_sensor <= '1';
        back_sensor <= '1';
        wait for 6*CLK_PERIOD;

        -- case_5: STOP --wrong_password--> STOP
        front_sensor <= '0';
        back_sensor <= '0';
        password_in <= 0, 5 after CLK_PERIOD, 3 after 2*CLK_PERIOD, 7 after
3*CLK_PERIOD;
        wait for 6*CLK_PERIOD;

        -- case_6: STOP --correct_password--> RIGHT_PASS
        password_in <= 1, 2 after CLK_PERIOD, 3 after 2*CLK_PERIOD, 4 after
3*CLK_PERIOD;
        wait for 6*CLK_PERIOD;

        -- case_7: RIGHT_PASS --(back_sensor = 1 and front_sensor = 0)-->
IDLE
```

```
        front_sensor <= '0';
        back_sensor <= '1';
        wait for 6*CLK_PERIOD;

        back_sensor <= '0';
        wait for CLK_PERIOD;

        -- case_8: IDLE --(front_sensor = 1)--> WAIT_PASSWORD
--password_correct--> RIGHT_PASS --(back_sensor = 1 and front_sensor =
0)--> IDLE
        front_sensor <= '1';
        wait for CLK_PERIOD;

        password_in <= 1, 2 after CLK_PERIOD, 3 after 2*CLK_PERIOD, 4 after
3*CLK_PERIOD;
        wait for 10*CLK_PERIOD;

        front_sensor <= '0';
        back_sensor <= '1'; -- going to IDLE
        wait for CLK_PERIOD;


        -- case_9: IDLE --(front_sensor = 1)--> WAIT_PASSWORD
--password_correct--> RIGHT_PASS --(rst = 1)--> RIGHT_PASS
        front_sensor <= '1';
        back_sensor <= '0';

        password_in <= 1, 2 after CLK_PERIOD, 3 after 2*CLK_PERIOD, 4 after
3*CLK_PERIOD;
        wait for 10*CLK_PERIOD;

        rst <= '1';
        wait for 3*CLK_PERIOD;

        -- case_10: RIGHT_PASS --(rst = 0)--> RIGHT_PASS
        rst <= '0';
        wait for 3*CLK_PERIOD;

        wait for 50 * CLK_PERIOD;
        finished <= '1';
```

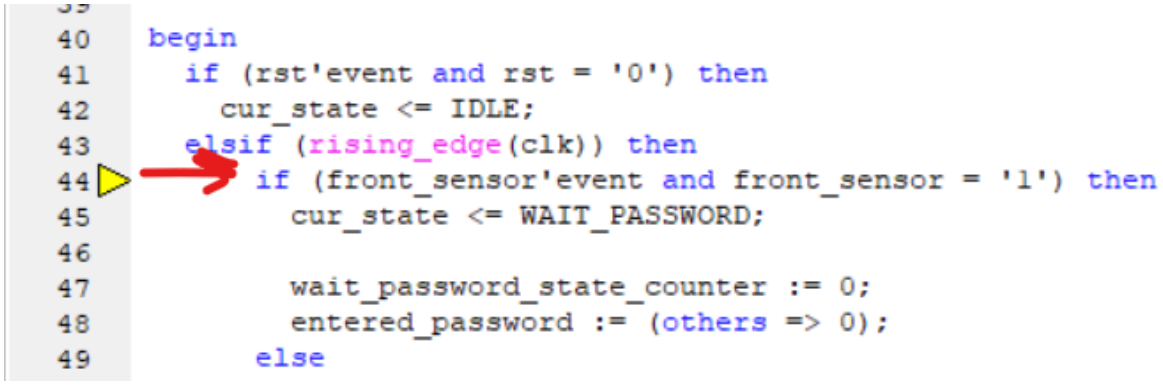
```
        wait;
    end process;
end test_bench;
```

Synthesizing process

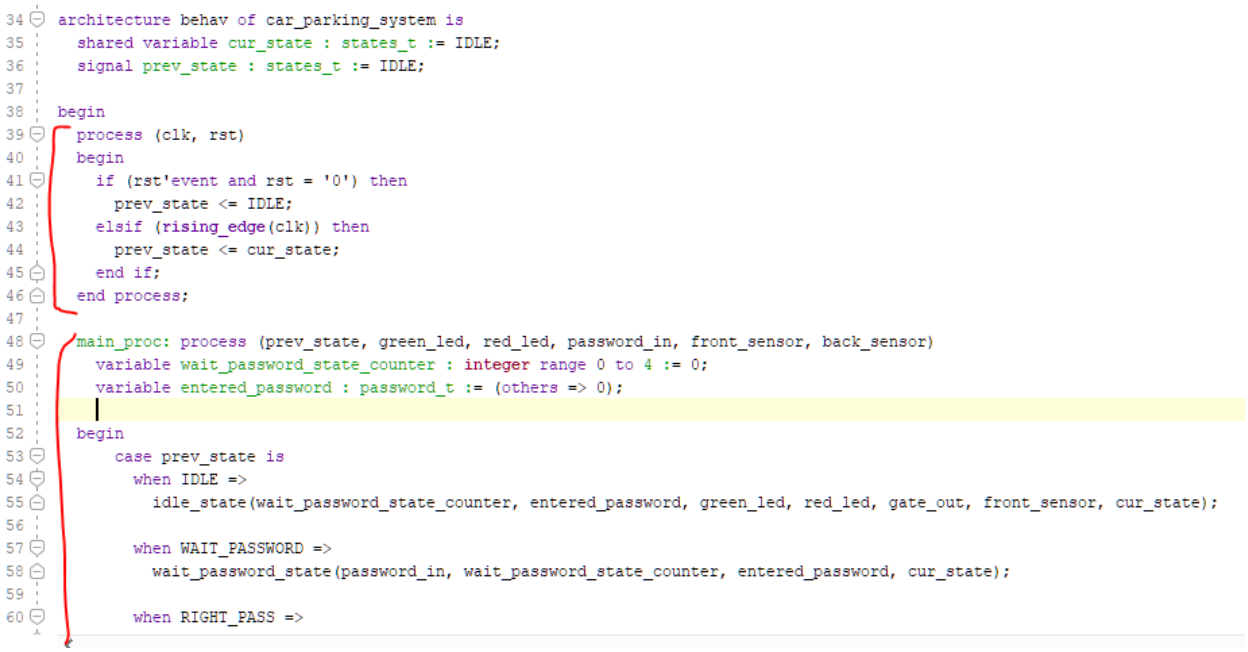
I used **ISE** as the program for synthesizing my VHDL code into circuit.
At first, I got an error like this:

Line 44: statement is not synthesizable since it does not hold its value under NOT(clock-edge) condition.

Line 44 of the code was this:



After lot of searching and asking friends, finally I realized that I should've written my main process like this:



```
architecture behav of car_parking_system is
  shared variable cur_state : states_t := IDLE;
  signal prev_state : states_t := IDLE;

begin
  process (clk, rst)
  begin
    if (rst'event and rst = '0') then
      prev_state <= IDLE;
    elsif (rising_edge(clk)) then
      prev_state <= cur_state;
    end if;
  end process;

  main_proc: process (prev_state, green_led, red_led, password_in, front_sensor, back_sensor)
    variable wait_password_state_counter : integer range 0 to 4 := 0;
    variable entered_password : password_t := (others => 0);

  begin
      case prev_state is
        when IDLE =>
          idle_state(wait_password_state_counter, entered_password, green_led, red_led, gate_out, front_sensor, cur_state);

        when WAIT_PASSWORD =>
          wait_password_state(password_in, wait_password_state_counter, entered_password, cur_state);

        when RIGHT_PASS =>
```

This code will be synthesizable.

In this code I used **2 processes**.
One that will be run repeatedly **because of the clock being sensed each time**.
Another process will be run **when current state or other inputs change**.

cur_state is actually the next state that is being set in the second process code.

I also had to change the **Idle state procedure** code:

```
procedure idle_state(
  variable wait_password_state_counter : out integer range 0 to 4;
  variable entered_password            : out password_t;
  signal green_led                   : inout std_logic;
  signal red_led                     : inout std_logic;
  signal gate_out                      : out std_logic;
  signal front_sensor                   : in std_logic;
  variable cur_state                   : out states_t
) is
begin
  if (front_sensor'event and front_sensor = '1') then
    cur_state := WAIT_PASSWORD;
  end if;

  wait_password_state_counter := 0;
  entered_password := (others => 0);

  green_led <= '0';
  red_led <= '0';
  gate_out <= '0';
end idle_state;
```

This code part was the real part that was causing the problem.
I put this into the **Idle procedure**, instead of it being a stand-alone statement inside of the main process.

After that I tried to Synthesize my code using ISE and it was successful!

I sent the video of the synthesized circuit along with my files. Because the circuit was very very big and couldn't fit here :)

Conclusion

The car parking system FPGA project was successfully designed and simulated.
The system is able to control the entrance of the parking lot by prompting for a password and only allowing access with the correct password.
The system also functions correctly, opening and closing the gate as needed.
This project demonstrates **the capabilities of an FPGA in controlling a real-world system** and can serve as a starting point for further developments in the field.