Project Based Learning Report

On

To model car parking system using FSM with VHDL and Xilinx simulator

Submitted in the partial fulfillment of the requirements

For the Project based learning in

VLSI Design Technology

In branch

Electronics & Communication Engineering

By

Name of the student	**PRN**
UTKARSH KUMAR MAURYA	2114110490
ARUN DESHMUKH	2114110442
VISHAL KUMAR	2114110476

Under the guidance of Course In-charge

Mrs. M. S. CHAVAN

Department of Electronics & Communication Engineering

Bharati Vidyapeeth
(Deemed to be University)
College of Engineering, Pune – 411043

Academic Year: - 2023-24

Bharati Vidyapeeth

(Deemed to be University)

College of Engineering,

Pune – 411043

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

Certified that the Project Based Learning report entitled, "Design a model car parking system using FSM with VHDL and Xilinx simulator" is a bonafied work done by

Name of the student	**PRN**
UTKARSH KUMAR MAURYA	2114110490
ARUN DESHMUKH	2114110442
VISHAL KUMAR	2114110476

in partial fulfillment of the requirements for the award of credits for Project Based Learning (PBL) in VLSI Design Technology Course of Bachelor of Technology Semester VI, in Electronics & Communication Engineering.

Date:

Mrs. M. S. CHAVAN
Course In-charge

Prof. Dr. A. A. Shinde
Professor & Head

INDEX

1. PROBLEM STATEMENT

Q. Design a car parking system using Finite State Machine (FSM) implemented with VHDL and simulated using Xilinx simulator. The car parking system should be able to manage the entry and exit of cars from a parking lot with multiple parking spaces. The system should track the availability of parking spaces and ensure orderly entry and exit of cars while maintaining proper state transitions.

Solution:

1. **Define the states of the FSM:**
   - Idle: Initial state where the system is waiting for a car to arrive.
   - Car Arriving: Transition state when a car arrives at the parking lot.
   - Parking: State where the car is being parked in an available parking space.
   - Parking Full: State indicating that all parking spaces are occupied.
   - Car Leaving: Transition state when a car is leaving the parking lot.
   - Car Exiting: State where the car is exiting the parking lot.
2. **Define inputs and outputs:**
   - Inputs: Arrival signal, Departure signal, Parking space availability signals.
   - Outputs: Control signals for managing entry and exit of cars, and parking space status.
3. **Implement FSM logic:**
   - Define transition conditions between states based on input signals.
   - Manage parking space availability by updating the availability signals accordingly.
   - Ensure proper sequencing of state transitions to maintain system integrity.
4. **Design VHDL code:**
   - Implement the FSM logic using VHDL constructs such as processes, signals, and state machines.
   - Define input and output ports for interfacing with the Xilinx simulator.
   - Simulate the VHDL code to verify its functionality and correctness.
5. **Simulate using Xilinx simulator:**
   - Use Xilinx ISE or Vivado to create a new project and add the VHDL code.
   - Simulate the design using testbench to validate the functionality under various scenarios.
   - Analyze simulation results to ensure correct behavior of the car parking system.
6. **Test and debug:**
   - Test the car parking system with different arrival and departure sequences.
   - Debug any issues encountered during simulation by examining signals and state transitions.
   - Refine the VHDL code as needed to address any identified issues.
7. **Validate and optimize:**
   - Validate the functionality of the car parking system against the requirements.
   - Optimize the design for performance and resource utilization if necessary.
   - Ensure the design meets timing requirements and operates efficiently.

2. INTRODUCTION

In today's world, the exponential growth in motor vehicle usage has led to various challenges such as pollution, traffic congestion, and parking problems. To address these issues, efficient parking systems are essential. Implementing a Parking System using Finite State Machine (FSM) and VHDL (Very High Speed Integrated Circuit Hardware Description Language) offers a promising solution. This system comprises crucial modules: an identification module and two sensors, namely the front and back sensors.

Efficient time utilization is paramount in today's fast-paced environment. Therefore, optimizing parking processes to minimize time wastage is crucial. Our proposed system aims to streamline parking procedures while ensuring security, thereby offering an effective time-saving solution. Security measures include password authentication before allowing access to parking slots, ensuring systematic and secure parking practices.

Additionally, addressing the global challenge of parking space scarcity, we introduce a car parking system utilizing FPGA (Field Programmable Gate Array) technology. This system features an automatic door opening mechanism, activated upon correct password entry. By integrating FPGA technology, our solution not only minimizes time spent searching for parking spaces but also enhances security by employing password protection.

Several research studies have contributed to the development of efficient parking systems. For instance, Ramneet Kaur and Balwinder Singh proposed a parking system utilizing FPGA, incorporating features such as LCD displays for space availability indication and RF modules for transmitting and receiving area availability information. Khor JingYong and Muataz H. Salih designed an embedded auto car parking system using FPGA, incorporating sensors for detecting emergencies such as sudden heart attacks, enhancing driver safety.

Furthermore, Bhavanachendika, Alapati Manideepu, and Fazal Noorbasha introduced a secured car parking system utilizing Verilog HDL. This system employs password or key authentication to access parking slots, ensuring security and preventing unauthorized access.

By integrating the features and advancements from various research studies, our comprehensive parking system offers a holistic solution to address parking challenges efficiently. With its emphasis on time optimization, security, and advanced technologies such as FPGA and VHDL, our proposed system aims to revolutionize parking management in urban environments, contributing to enhanced traffic flow and overall urban efficiency.

Project Objective:

The objective of the project is to design, implement, and simulate a car parking system using VHDL and Xilinx simulator. The main goals include:

System Modeling: Develop a comprehensive FSM model to represent the behavior of the car parking system, including states such as idle, vehicle detected, parking spot occupied, etc.

Sensor Integration: Integrate sensors to accurately detect the presence of vehicles in parking spots and update the system accordingly.

User Interaction: Design a user interface to facilitate interaction with the system, allowing users to select parking spots, retrieve vehicles, and view parking availability.

Barrier Control: Implement automated barrier control mechanisms to regulate vehicle entry and exit based on parking spot availability and user commands.

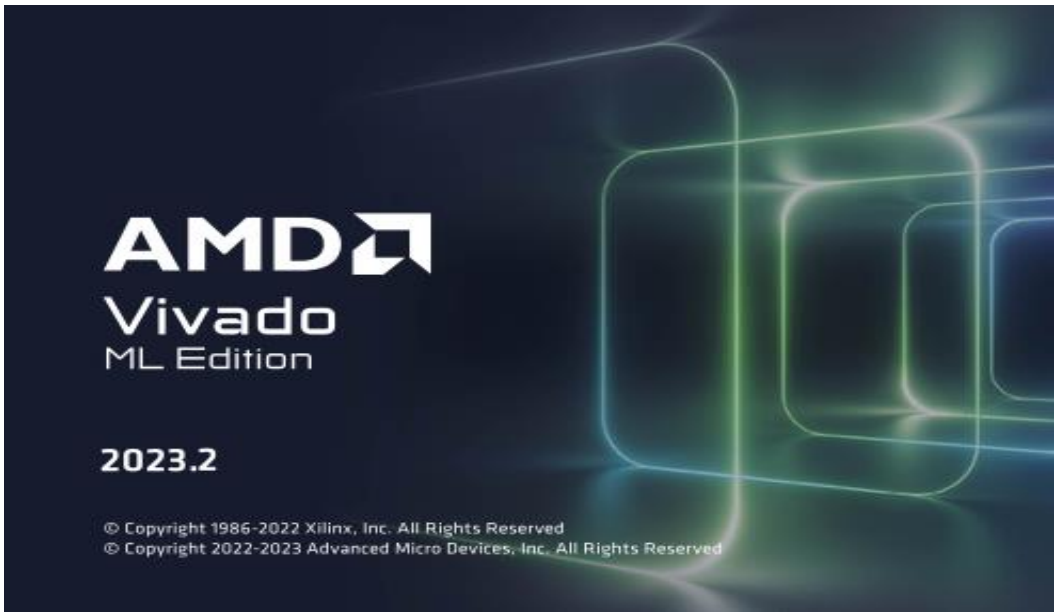
Simulation and Testing: Utilize Xilinx simulator to simulate various scenarios and test the functionality of the car parking system under different conditions, ensuring reliability and efficiency.

Optimization: Optimize the design and implementation of the system to ensure efficient use of resources and fast response times.

Documentation: Provide comprehensive documentation including design specifications, VHDL code, simulation results, and user manuals for future reference and maintenance.

By successfully completing this project, you will gain valuable experience in FPGA programming, FSM design, sensor integration, and simulation, while also contributing to the development of a practical and efficient car parking system.

3. SOFTWARE USED



Vivado is a widely-used software tool suite developed by Xilinx for designing and implementing digital circuits, particularly for FPGAs (Field-Programmable Gate Arrays) and SoCs (System-on-Chips). Here are some key components and features of Vivado:

Design Entry: Vivado supports various methods for entering your design, including RTL (Register Transfer Level) entry using HDLs (Hardware Description Languages) such as Verilog, VHDL, or SystemVerilog. It also supports IP Integrator for graphical design entry and block-based design methodologies.

Synthesis: After entering the design, Vivado synthesizes the RTL code into a netlist of logical components. This process optimizes the design for area, power, and performance based on the target FPGA or SoC.

Implementation: In this stage, Vivado maps the synthesized design onto the target FPGA or SoC architecture, placing and routing the logical components to physical resources such as LUTs (Look-Up Tables), flip-flops, DSP slices, and I/O pins. It also performs timing analysis to ensure that the design meets timing constraints.

Verification: Vivado provides various verification tools, including simulation, timing analysis, and formal verification, to ensure that the design behaves correctly and meets its specifications.

Debugging: Vivado offers debugging features such as integrated logic analyzer, waveform viewer, and interactive debugging to help designers analyze and debug their designs efficiently.

IP (Intellectual Property) Catalog: Vivado includes a vast library of pre-designed IP cores that can be easily integrated into your design, saving time and effort in development. These IP

cores cover a wide range of functionalities such as memory controllers, interface protocols, signal processing, and more.

High-Level Synthesis (HLS): Vivado HLS allows designers to describe algorithms in C, C++, or SystemC and automatically generate RTL code optimized for FPGAs or SoCs. This enables designers to explore different architectural implementations and accelerate the design process.

Programming and Configuration: Once the design is implemented, Vivado generates configuration files and bitstreams that can be used to program the target FPGA or SoC. It supports various programming methods, including JTAG, PCIe, and Ethernet, depending on the target device and application requirements.

Overall, Vivado provides a comprehensive toolchain for designing, implementing, and verifying FPGA and SoC-based systems, making it a popular choice among hardware designers and engineers.

4. WORKING

Figure.1 represents a diagram for the implemented car parking system on FPGA. In this diagram, when a vehicle enters near to parking Gate, then IR sensor detects the car object and system enters into wait password state. In this state driver enters a password. If the password is correct, the door opens for entrance. But if the password is wrong, the door remains close and waits till the right password is entered. Seven segment displays is used to display information of the different FSM states. According to the information displayed, the driver parks the vehicle. Two IR sensors (front and back) are used to detect the motion of the vehicle. The front sensor detects the vehicle going to the gate of the parking area. Theback sensor detects if the coming vehicle pass the gate and get inside the parking area and also detects for next car coming toward parking area.

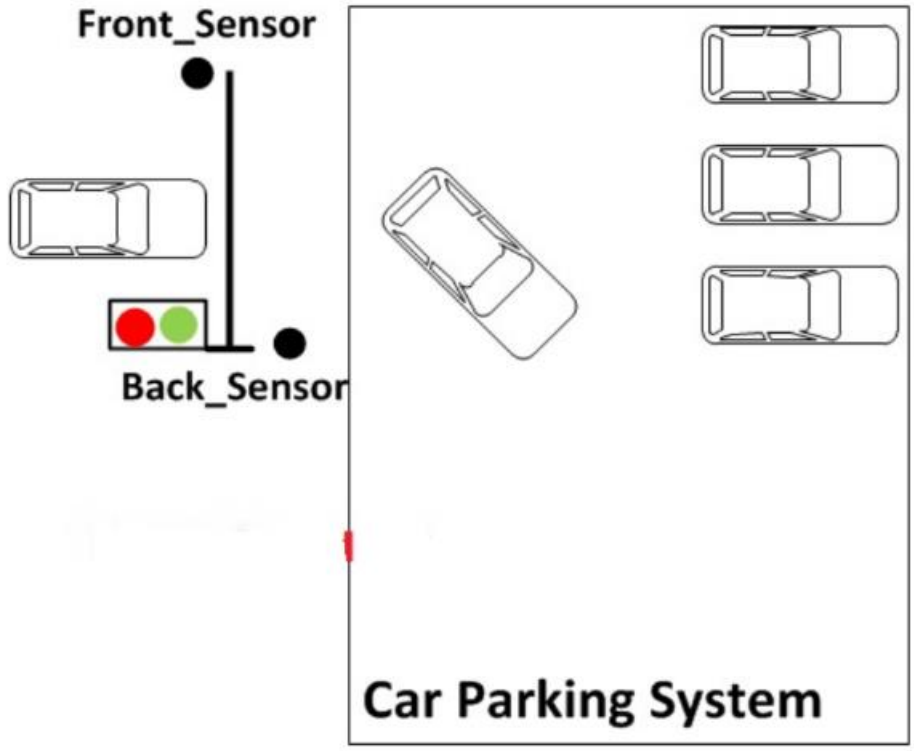


Figure 1: Diagram for the implemented car parking system

State machine diagram for the implemented car parking system is shown in Figure.2.In this figure, at first, the FSM (Finite State Machine) is in IDLE state. If a vehicle approaching near the gate is detected by the front sensor, the FSM is changed to WAIT_PASSWORD state for 4 cycles. In this state, the car will enter the password; and if the password entered is correct, the gate is opened to let the vehicle get inside the parking area and the FSM changes to RIGHT_PASSWORD state; a Green LED will be blinking. And if the entered password is incorrect, the FSM changes to WRONG_PASSWORD state; a Red LED will be blinking and the vehicle needs to re-enter the password until the password is correct. When the current vehicle coming inside the parking area is detected by the back sensor and if there is a next vehicle coming at the gate, the FSM is switched to STOP state and the Red LED will be blinking so that the next car will become aware to stop and enter the password. After the vehicle passes the gate and gets inside the parking area, the FSM returns to IDLE state.

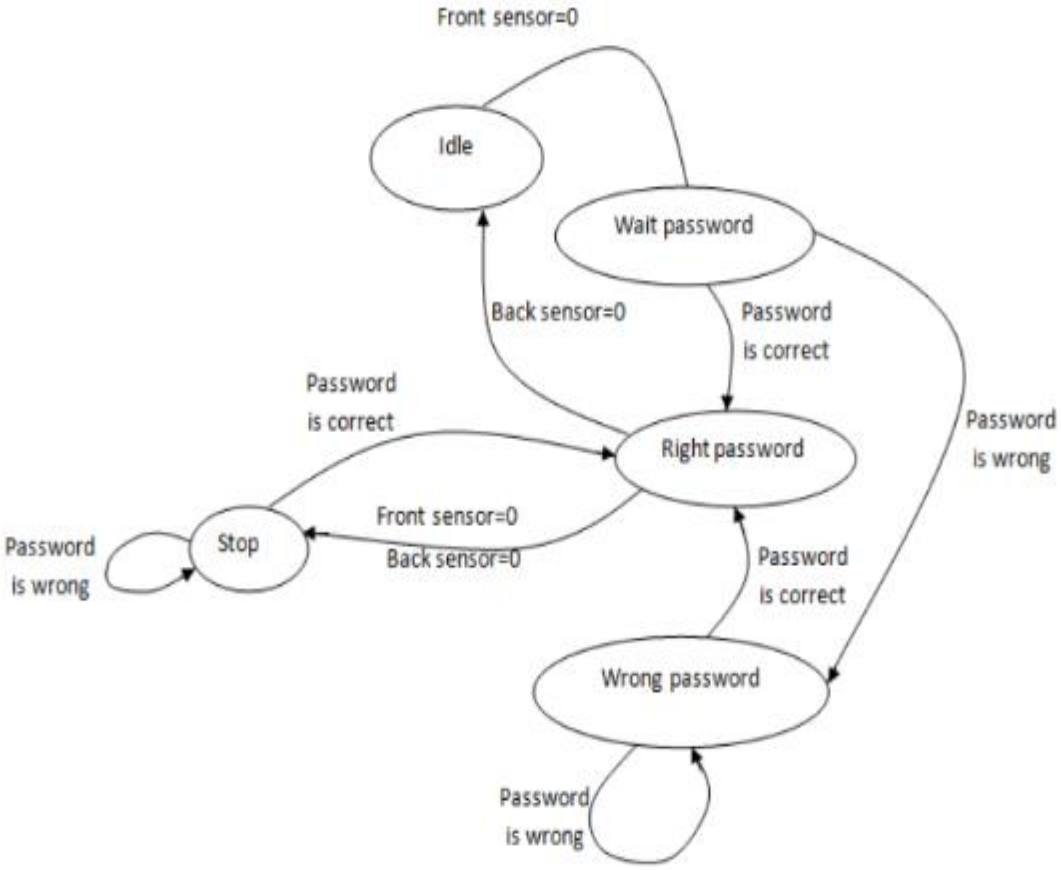


Figure 2: State Machine for the implanted car parking system

5. SOURCE CODE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
entity Car_Parking_System_VHDL is
port
(
  clk,reset_n: in std_logic; -- clock and reset of the car parking system
  front_sensor, back_sensor: in std_logic; -- two sensor in front and behind the gate of the car
parking system
  password_1, password_2: in std_logic_vector(1 downto 0); -- input password
  GREEN_LED,RED_LED: out std_logic; -- signaling LEDs
  HEX_1, HEX_2: out std_logic_vector(6 downto 0) -- 7-segment Display
);
end Car_Parking_System_VHDL;

architecture Behavioral of Car_Parking_System_VHDL is
-- FSM States
type FSM_States is (IDLE,WAIT_PASSWORD,WRONG_PASS,RIGHT_PASS,STOP);
signal current_state,next_state: FSM_States;
signal counter_wait: std_logic_vector(31 downto 0);
signal red_tmp, green_tmp: std_logic;

begin
-- Sequential circuits
process(clk,reset_n)
begin
 if(reset_n='0') then
  current_state <= IDLE;
 elsif(rising_edge(clk)) then
  current_state <= next_state;
 end if;
end process;
-- combinational logic
-- fpga4student.com FPGA projects, Verilog projects, VHDL projects
process(current_state,front_sensor,password_1,password_2,back_sensor,counter_wait)
 begin
 case current_state is
 when IDLE =>
 if(front_sensor = '1') then -- if the front sensor is on,
 -- there is a car going to the gate
  next_state <= WAIT_PASSWORD;-- wait for password
 else
  next_state <= IDLE;
 end if;
 when WAIT_PASSWORD =>
 if(counter_wait <= x"00000003") then
  next_state <= WAIT_PASSWORD;
 else -- check password after 4 clock cycles
```

```
 if((password_1="01")and(password_2="10")) then
 next_state <= RIGHT_PASS; -- if password is correct, let them in
 else
 next_state <= WRONG_PASS; -- if not, tell them wrong pass by blinking Green LED
 -- let them input the password again
 end if;
 end if;
 when WRONG_PASS =>
  if((password_1="01")and(password_2="10")) then
 next_state <= RIGHT_PASS;-- if password is correct, let them in
  else
 next_state <= WRONG_PASS;-- if not, they cannot get in until the password is right
  end if;
 when RIGHT_PASS =>
  if(front_sensor='1' and back_sensor = '1') then
 next_state <= STOP;
 -- if the gate is opening for the current car, and the next car come,
 -- STOP the next car and require password
 -- the current car going into the car park
  elsif(back_sensor= '1') then
    -- if the current car passed the gate an going into the car park
    -- and there is no next car, go to IDLE
 next_state <= IDLE;
  else
 next_state <= RIGHT_PASS;
  end if;
when STOP =>
  if((password_1="01")and(password_2="10"))then
  -- check password of the next car
  -- if the pass is correct, let them in
 next_state <= RIGHT_PASS;
  else
 next_state <= STOP;
  end if;
 when others => next_state <= IDLE;
 end case;
 end process;
 -- wait for password
process(clk,reset_n)
 begin
 if(reset_n='0') then
 counter_wait <= (others => '0');
 elsif(rising_edge(clk))then
  if(current_state=WAIT_PASSWORD)then
  counter_wait <= counter_wait + x"00000001";
  else
  counter_wait <= (others => '0');
  end if;
 end if;
 end process;
```

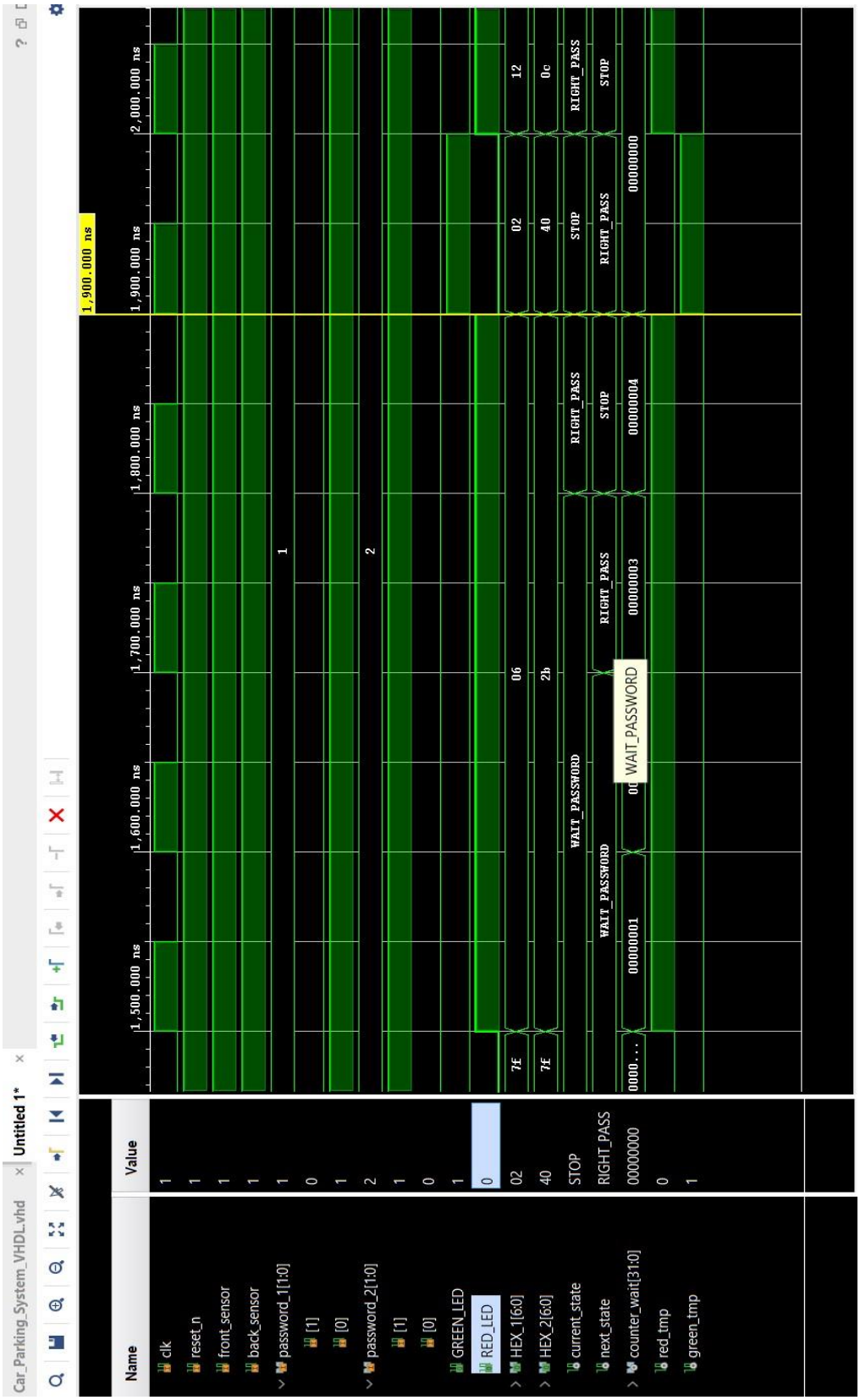
```
 -- output
 process(clk) -- change this clock to change the LED blinking period
 begin
 if(rising_edge(clk)) then
 case(current_state) is
 when IDLE =>
 green_tmp <= '0';
 red_tmp <= '0';
 HEX_1 <= "1111111"; -- off
 HEX_2 <= "1111111"; -- off
 when WAIT_PASSWORD =>
 green_tmp <= '0';
 red_tmp <= '1';
 -- RED LED turn on and Display 7-segment LED as EN to let the car know they need to
input password
 HEX_1 <= "0000110"; -- E
 HEX_2 <= "0101011"; -- n
 when WRONG_PASS =>
 green_tmp <= '0'; -- if password is wrong, RED LED blinking
 red_tmp <= not red_tmp;
 HEX_1 <= "0000110"; -- E
 HEX_2 <= "0000110"; -- E
 when RIGHT_PASS =>
 green_tmp <= not green_tmp;
 red_tmp <= '0'; -- if password is correct, GREEN LED blinking
 HEX_1 <= "0000010"; -- 6
 HEX_2 <= "1000000"; -- 0
 when STOP =>
 green_tmp <= '0';
 red_tmp <= not red_tmp; -- Stop the next car and RED LED blinking
 HEX_1 <= "0010010"; -- 5
 HEX_2 <= "0001100"; -- P
 when others =>
 green_tmp <= '0';
 red_tmp <= '0';
 HEX_1 <= "1111111"; -- off
 HEX_2 <= "1111111"; -- off
  end case;
 end if;
 end process;
  RED_LED <= red_tmp  ;
  GREEN_LED <= green_tmp;

end Behavioral;
```

6. ALGORITHM

1. **Entity Declaration:**
    a) Declare the entity Car_Parking_System_VHDL with input and output ports including clock (clk), reset (reset_n), front sensor (front_sensor), back sensor (back_sensor), two password inputs (password_1, password_2), signaling LEDs (GREEN_LED, RED_LED), and two 7-segment display outputs (HEX_1, HEX_2).
2. **Architecture:**
    a) Define the architecture Behavioral of the entity.
    b) Declare signals for FSM states (current_state, next_state), a counter for waiting time (counter_wait), and temporary signals for LED control (red_tmp, green_tmp).
3. **Sequential Process:**
    a) Implement a sequential process triggered by the clock (clk) and reset (reset_n).
    b) Update the current state based on the next state at the rising edge of the clock.
4. **Combinational Process:**
    a) Implement a combinational process sensitive to changes in inputs (current_state, front_sensor, password_1, password_2, back_sensor, counter_wait).
    b) Define the behavior for each FSM state using a case statement.
    c) Update the next state based on the current state and input conditions.
5. **Wait for Password Process:**
    a) Implement a process to count the waiting time for entering the password.
    b) Increment the counter if in the "WAIT_PASSWORD" state and reset it otherwise.
6. **Output Process:**
    a) Implement a process to control the LEDs and 7-segment displays based on the current state and clock signal.
    b) Define LED and display patterns for each state, indicating system status to the user.
7. **Assign Outputs:**
    a) Assign the temporary signals red_tmp and green_tmp to the RED_LED and GREEN_LED outputs, respectively.
8. **End Architecture.**

This algorithm outlines the behavior of the VHDL code for the car parking system, describing how the FSM transitions between states based on inputs and how outputs are controlled accordingly to provide feedback to the user.

7. OUTPUT


Car_Parking_System_VHDL.vhd
Untitled 1*
Name
Value
clk
reset_n
front_sensor
back_sensor
password_1[1:0]
[1]
[0]
password_2[1:0]
[1]
[0]
GREEN_LED
RED_LED
HEX_1[6:0]
HEX_2[6:0]
current_state
next_state
counter_wait[31:0]
red_tmp
green_tmp
1,500.000 ns
1,600.000 ns
1,700.000 ns
1,800.000 ns
1,900.000 ns
2,000.000 ns
1,900.000 ns
WAIT_PASSWORD
RIGHT_PASS
STOP
WAIT_PASSWORD

8. CONCLUSION

The VHDL code effectively models a car parking system using finite state machines and sensor integration. Through LED signaling, users receive clear feedback, facilitating interaction. While the implementation is solid, further refinement and rigorous testing are necessary for real-world applicability.

9. COURSE OUTCOME

- **CO2-** Apply concepts of Finite State Machine on sequential circuits.
- Hence, CO2 is attained.

10. REFERENCES

- https://www.fpga4student.com/2017/08/car-parking-system-in-vhdl-using-FSM.html
- https://www.researchgate.net/publication/360165372_Design_and_Implementation_of_Secured_Car_Parking_System_using_FPGA
- "Digital Design and Computer Architecture" by David Harris and Sarah Harris.
- "FPGA Prototyping by VHDL Examples" by Pong P. Chu