**SPECIAL PROJECT REPORT**

On

# INTELLIGENT PARKING SYSTEM USING VERILOG

BY

**YELE SRIKANTH**

**22STUCHH010964**

*in partial fulfilment for the award of the degree*

of

**BACHELOR OF TECHNOLOGY**

In

**ELECTRONICS AND COMMUNICATION ENGINEERING**

Under the guidance of

**Dr. RAJESH KUMAR JHA**

**Faculty of Science and Technology (ICFAI Tech)**

**The ICFAI Foundation for Higher Education, Hyderabad**

**(A Deemed to be University under section 3 of UGC Act. 1956)**

**April 2025**

# BONAFIDE CERTIFICATE

This is to certify that the project report **" INTELLIGENT PARKING SYSTEM USING VERILOG "** is the bonafide work carried out by **YELE SRIKANTH 22STUCHH010964,** in partial fulfilment of the requirements for the award of the Degree Bachelor of Engineering in Electronics and Communication Engineering from ICFAITECH School, Hyderabad during the VIth semester of their B.Tech course during the Academic year 2025-2026.

| | |
|---|---|
| **SIGNATURE OF SUPERVISOR** | **SIGNATURE OF COORDINATOR** |
| **Dr. RAJESH KUMAR JHA** | **Dr. SHAKEEL HASHMI** |

**SIGNATURE OF HOD**

**Dr. ASISA KUMAR PANIGRAHY**

# ACKNOWLEDGEMENT

This is an acknowledgement of the intensive drive and competence of everyone whohas contributed to the success of our project.

We are extremely grateful to our respected Director **Dr. KL Narayana** for fostering an excellent academic climate in our institution and we also express our sincere gratitude to our respected Head of the Department **Dr. Asisa Kumar Panigrahy** for his encouragement, able guidance and effort in bringing out this project.

We are deeply indebted to our internal guide **Dr. RAJESH KUMAR JHA,** Associate Professor, for his guidance, encouragement, co-operation and kindness during the entire duration of the course and academics.

Y. Srikanth
22STUCHH010964

# ABSTRACT:

The design and implementation of a verilog-based intelligent parking system aims to optimize parking space utilization, reduce the traffic and improve user convenience through an automated process. The Intelligent Parking System is designed as flexibility and affordable that integrates sensors and digital logic circuits for real-time tracking and guidance of parking spaces. The system utilizes sensors at each parking slot to detect vehicle presence, with the data processed and managed by Verilog-based digital logic modules. The design is implemented using Verilog to define finite state machines (FSMs) for managing system operations, ensuring a reliable and well-organized parking experience. Simulation and Testing of the system were conducted using tools like Xilinx Vivado, verifying the exactness and strength of the design. Among cities, apartments and multi-storage buildings, parking the vehicles is quite difficult and the main problem is the space, which is not enough. It decreases the stress of parking in restricted areas. The system shows a minimum occupied space, thus proving to be an alternative way of controlling parking in an overcrowded metropolitan area. Implementation of finite state machine in an asynchronous vehicle parking system is made. In conclusion, this technology plays a important role in planning to develop more efficient, sustainable and, of course, more attractive urban areas.

**KEYWORDS:**

Verilog, Parking system, Xilinx vivado, Finite State Machine (FSM), Digital logic circuits, reliable.

# TABLE OF CONTENTS

## LIST OF FIGURES

# LIST OF TABLES

## LIST OF ABBREVIATIONS:

| S.NO | Abbreviation | Full Form |
|---|---|---|
| 1. | HDL | Hardware Description Language |
| 2. | FPGA | Field Programmable Gate Array |
| 3. | IPS | Intelligent Parking System |
| 4. | IR | Infrared (Sensor) |
| 5. | LSB | Least Significant Bit |
| 6. | MSB | Most Significant Bit |
| 7. | I/O | Input / Output |
| 8. | clk | Clock |
| 9. | rst | Reset |
| 10. | FSM | Finite State Machine |
| 11. | LCD | Liquid Crystal Display |
| 12. | LED | Light Emitting Diode |
| 13. | RTL | Register Transfer Level |
| 14. | SOP | Sum of Products |
| 15. | VHDL | VHSIC Hardware Description Language |
| 16. | ASIC | Application-Specific Integrated Circuit |
| 17. | TB | Testbench |
| 18. | Synthesis | Converting HDL code into logic gate-level netlist |
| 19. | Simulation | Testing behavior of Verilog code without hardware |

# 1. INTRODUCTION

With the rapid urbanization and exponential growth in vehicular traffic, efficient management of parking spaces has become a significant challenge, especially in densely populated metropolitan areas[1]. Traditional parking systems are often inefficient, leading to time-consuming vehicle searches, increased fuel consumption, and environmental pollution. To address these challenges, the development of intelligent parking systems has emerged as a promising solution, offering automated monitoring, space optimization, and user-friendly access to real-time parking information.

This project presents the design and implementation of an Intelligent Parking System using Verilog Hardware Description Language (HDL), targeting Field Programmable Gate Arrays (FPGAs) for hardware-level control and automation. The proposed system aims to simulate and manage the parking process by automating space detection, vehicle entry/exit control, and occupancy display.

Verilog is chosen for this project due to its suitability for designing complex digital circuits, real-time responsiveness, and efficient utilization of hardware resources. By leveraging the parallel processing capabilities of digital logic and the flexibility of FPGA platforms, the intelligent parking system offers an effective and scalable solution that can be tailored to various parking scenarios, from small private lots to large multi-level commercial garages.

The core features of the system include:

- Automatic Vehicle Detection: Sensors (simulated through logic in Verilog) detect the presence of a vehicle at the entrance and in individual parking slots.
- Slot Allocation & Monitoring: The system keeps track of available and occupied slots and updates the status accordingly.
- Gate Control Mechanism: Entry and exit gates are automatically controlled based on the availability of slots.
- Real-Time Display System: Parking status is displayed on an output module (LEDs or seven-segment displays), informing users of available slots and system status.
- Security and Efficiency: Unauthorized access is restricted, and the system ensures orderly vehicle movement with minimal human intervention.

This Verilog-based implementation not only demonstrates the potential of digital design in real-world automation applications but also serves as a foundation for more advanced smart systems incorporating sensors, communication modules (like GSM or IoT), and user interfaces. The project emphasizes modular design, testbench verification, and simulation, aligning with industry practices in embedded systems and hardware design.

In the following sections, the report delves into the system architecture, Verilog implementation, simulation results, and potential enhancements, aiming to provide a comprehensive understanding of the intelligent parking solution developed.

## 1.1 Background:

Urbanization and the rapid increase in vehicle ownership have placed significant pressure on existing transportation and parking infrastructure. In most metropolitan areas, the demand for parking spaces often exceeds availability, leading to traffic congestion, wasted time, increased fuel consumption, and environmental pollution. Traditional parking systems, which depend heavily on manual operation and user judgment, are increasingly unable to cope with the dynamic demands of modern cities.

As a response to these challenges, **intelligent parking systems** have emerged as an innovative solution, aiming to automate and streamline the parking process[2]. These systems typically use a combination of sensors, controllers, and real-time data processing to monitor parking space availability, control vehicle entry/exit, and guide drivers to free spots. Intelligent parking not only improves user convenience but also enhances space utilization and reduces the environmental footprint.

From a design and implementation perspective, the development of such systems requires a reliable, efficient, and low-latency control mechanism. While software-based solutions exist, **hardware-based implementations** offer advantages in terms of **real-time performance, parallelism, and reliability**. This is where **Hardware Description Languages (HDLs)** like **Verilog** come into play.

Verilog HDL allows for the design and simulation of complex digital systems at the hardware level. It is widely used in the industry to program Field Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs). By using Verilog, designers can create systems that are faster, more efficient, and more responsive than their software-only counterparts.

In this project, an **Intelligent Parking System is implemented using Verilog HDL**, simulating essential functionalities such as slot monitoring, gate control, and occupancy indication. The system is designed to demonstrate the practical application of digital design principles in solving real-world urban problems. Through this hardware-centric approach, the project not only enhances understanding of Verilog and digital logic but also contributes toward the development of smarter, more sustainable urban infrastructure.

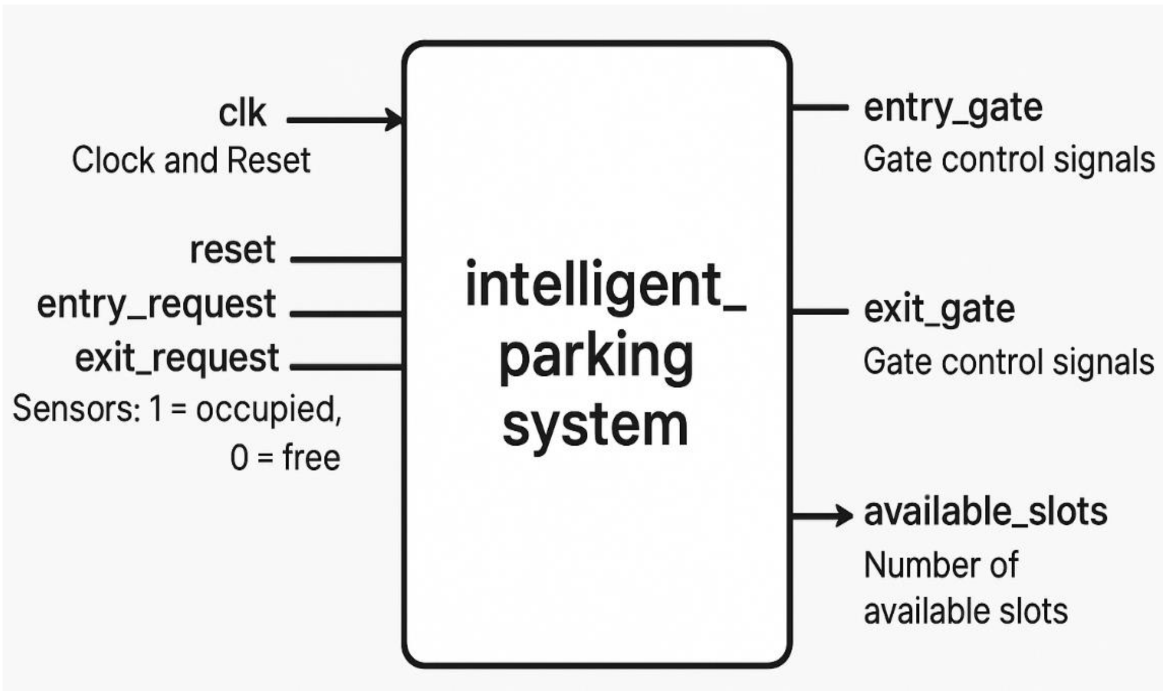

FIGURE 1:Module Diagram of the Proposed Model

## 1.2 Problem Statement:

The exponential growth in urban population and the rising number of privately owned vehicles have resulted in significant challenges related to parking space management. Conventional parking systems, which often rely on manual monitoring and user-dependent decisions, are inefficient and prone to issues such as:

- Time-consuming search for vacant parking slots
- Traffic congestion near parking areas
- Unauthorized parking and poor space utilization
- Lack of real-time information on parking availability

- Dependence on human intervention for gate and slot control

These limitations not only cause inconvenience for drivers but also lead to increased fuel consumption, air pollution, and unnecessary stress. While software-based parking solutions exist, they often fall short in terms of response time, scalability, and integration with hardware-level control systems.

To address these challenges, there is a need for an **automated, intelligent parking system** that can efficiently manage parking spaces, regulate vehicle entry and exit, and provide real-time feedback to users—all without human involvement. The system must be reliable, fast, and suitable for real-time deployment in embedded environments.

This project proposes the design and simulation of an intelligent parking system using **Verilog Hardware Description Language (HDL)**. By implementing the system at the hardware level, it ensures quick response times, accurate slot management, and automated control of entry/exit mechanisms. The project aims to model a system that can be easily scaled and deployed on FPGA-based platforms, demonstrating both the functionality and practicality of digital design in solving real-world automation problems.

## 1.3 Motivation:

In modern urban environments, the growing number of vehicles has outpaced the expansion of parking infrastructure, leading to significant challenges such as traffic congestion, increased fuel consumption, pollution, and driver frustration. Studies have shown that a substantial amount of urban traffic is attributed to drivers searching for available parking spaces. The inefficiency of conventional parking methods, which often rely on manual monitoring and human intervention, calls for a smarter, automated solution.

The motivation behind this project stems from the need to develop a **cost-effective, real-time, and automated parking management system** that can optimize space utilization and improve the overall parking experience. By using **Verilog HDL**, the system can be implemented at the hardware level, providing faster response times, reliability, and suitability for integration into embedded systems such as FPGAs or ASICs.

Furthermore, with the growing trend of **smart cities and IoT**, intelligent parking systems are becoming a crucial component of urban infrastructure. Implementing such systems using

Verilog not only helps in academic learning and understanding of digital system design but also aligns with industry trends in automation and hardware-based control systems.

## 1.4 Objective:

The primary objectives of the Intelligent Parking System using Verilog are:

1. **To design and simulate a digital system that automates parking operations** such as vehicle detection, gate control, and slot monitoring using Verilog HDL.
2. **To implement real-time slot availability tracking**, ensuring that vehicles are only allowed to enter when a parking slot is available, thereby avoiding over-crowding.
3. **To control entry and exit gates automatically** based on sensor inputs and system logic, eliminating the need for human attendants.
4. **To display real-time parking status** (e.g., number of available slots) on output devices such as LEDs or seven-segment displays, providing clear information to users.

By achieving these objectives, the project aims to provide a proof-of-concept model for a smart parking system that could be deployed in various real-world scenarios with minimal adaptation and high efficiency.

## 1.5 Scope:

Verilog is used for **hardware description**, which means your IPS would be implemented at the hardware level (typically on an FPGA). Here's the scope and potential:

**1. Real-Time Slot Management**

- Detect incoming vehicles via sensors (IR, ultrasonic)
- Use Verilog to control signal processing and logic
- Update display boards showing free/occupied slots

**2. Automatic Entry/Exit Control**

- Barrier gates can be controlled via state machines
- Entry permitted only if slots are available (controlled in Verilog logic)

**3. Efficient Slot Allocation Algorithm**

- Use FSM (Finite State Machines) written in Verilog
- Allocate nearest available slot, update system state

**4. Display Interface**

- 7-segment display or LED matrix to show available slots
- Controlled via Verilog code (multiplexing, binary-to-decoder logic)

**5. Integration with Sensors**

- Read inputs from IR/ultrasonic sensors (via GPIO)
- Debounce and filter sensor signals in Verilog

TABLE 1: Module Description

| S.NO | Module Name | Purpose | I/O Ports |
|---|---|---|---|
| 1. | intelligent_parking_system | Main controller managing parking logic | clk, reset, entry_request, exit_request, entry_gate, exit_gate, available_slots |
| 2. | entry_controller | Handles entry gate logic and updates slot count on vehicle entry | entry_request, available_slots, entry_gate |
| 3. | exit_controller | Handles exit gate logic and updates slot count on vehicle exit | exit_request, exit_gate, available_slots |
| 4. | slot_counter | Maintains real-time count of available parking slots | reset, entry_request, exit_request, available_slots |

# 2. LITERATURE REVIEW:

## 2.1 Overview of Parking Systems:

Parking systems are integral to urban planning and transportation infrastructure. As cities grow and vehicle ownership increases, effective parking management becomes critical to reducing congestion, pollution, and driver frustration. Traditional parking systems, which rely on manual monitoring and user-driven vehicle placement, are inefficient and time-consuming.

Modern parking systems aim to address these limitations by integrating automation technologies such as sensors, microcontrollers, and intelligent algorithms. These systems typically include features like automatic vehicle detection, slot allocation, display units, and barrier control. Their primary goals are to optimize space utilization, improve accessibility, and provide real-time feedback to users and operators[3].

## 2.2 Existing Intelligent Parking Solutions:

A variety of intelligent parking systems have been developed using different technologies. Some of the most common implementations include:

- **Sensor-Based Systems**: These use IR or ultrasonic sensors to detect vehicle presence in parking slots. The sensor data is processed by a central controller to update the availability status.
- **Microcontroller-Based Solutions**: Microcontrollers such as Arduino and Raspberry Pi are commonly used for small- to medium-scale implementations. These systems manage sensor input, user displays, and gate operations.
- **IoT-Enabled Systems**: These advanced systems use wireless connectivity (Wi-Fi, GSM, LoRa) to allow users to check parking availability via mobile apps or web platforms. Data is often stored and processed on the cloud for real-time access.
- **FPGA-Based Systems**: Field Programmable Gate Arrays (FPGAs) programmed using Verilog or VHDL provide low-latency, hardware-level control for real-time slot monitoring and gate control.

Despite these developments, many systems suffer from trade-offs between performance, scalability, and user interface functionality.

## 2.3 Role of Hardware Description Languages in Automation:

Hardware Description Languages (HDLs) like **Verilog** and **VHDL** are essential for the design and simulation of digital circuits. Unlike software-based approaches that run on general-purpose processors, HDL-based systems are implemented directly in hardware, offering significant advantages in speed, reliability, and parallelism.

In the context of parking systems, Verilog enables:

- **Real-Time Logic Control**: Implementation of finite state machines (FSMs) for entry/exit control, slot allocation, and barrier automation.
- **Parallel Slot Monitoring**: Multiple parking bays can be monitored simultaneously using concurrent hardware processes.
- **Efficient Display Management**: Real-time updates of available slots on 7-segment displays or LED matrices.
- **Low Power Consumption**: Suitable for embedded, energy-efficient deployments.

Verilog's modular nature also allows developers to scale systems easily or integrate them with external modules such as sensors and displays.

## 2.4 Comparative Analysis of Similar Systems:

A review of existing literature highlights the technological diversity in intelligent parking systems. Below is a comparison of notable implementations:

TABLE 2: Comparison between Technology, Features and Limitations

| **Study / Project** | **Technology** | **Features** | **Limitations** |
|---|---|---|---|
| Rajalakshmi et al. (2019)[4] | Verilog on FPGA | Slot detection, gate control, 7-segment display | No user interface or IoT support |
| Swathi & Rajesh (2020)[5] | Arduino + IR sensors | Basic entry automation | Limited scalability, higher latency |
| Ashwini & Nandhini (2021)[6] | Verilog on Xilinx FPGA | Modular design, multiple slot control | Lacks user interactivity |

| Study / Project | Technology | Features | Limitations |
|---|---|---|---|
| IoT Smart Parking (2022)[7] | Raspberry Pi + Cloud | Real-time mobile access, live updates | Slower than FPGA systems |

These systems demonstrate a growing interest in hardware-based solutions, but most still lack full integration of features like wireless access, remote control, and real-time responsiveness at scale.

### 2.5 Gaps in Existing Solutions:

Despite the advancements in intelligent parking systems, the following gaps remain unaddressed:

- **Real-Time Scalability**: Many microcontroller-based systems struggle to handle a large number of slots efficiently due to serial processing and limited I/O.
- **Lack of Integration**: HDL-based systems often lack IoT features, while IoT-enabled systems suffer from delayed data processing.
- **Limited User Interaction**: Few systems offer a user-friendly interface for booking or navigating to available slots.
- **Static Slot Allocation**: Most designs use fixed or sequential allocation methods without considering traffic flow or dynamic optimization.
- **Maintenance & Debugging Complexity**: Hardware systems, especially those involving sensors, require robust error detection and fail-safes.

These limitations provide motivation for the design of a Verilog-based intelligent parking system that is **modular, scalable, and capable of integrating with modern IoT technologies**.

# 3. SYSTEM DESIGN & METHODOLOGY:

### 3.1 System Overview:

The Intelligent Parking System (IPS) designed in this project aims to automate the process of parking space management using **Verilog HDL on FPGA**. The system is responsible for monitoring the availability of parking slots, controlling entry and exit gates based on real-time slot status, and displaying the number of free slots[8]. The design leverages digital logic for high-speed performance and real-time responsiveness, making it suitable for deployment in high-traffic environments such as malls, offices, and residential complexes.

### 3.2 Functional Requirements:

The primary functional requirements of the proposed Intelligent Parking System include:

- **Vehicle Detection**: Detect the presence or absence of a vehicle in a parking slot using sensors.
- **Slot Availability Tracking**: Monitor and update the total number of available slots.
- **Gate Automation**: Open entry gate if slots are available; open exit gate upon vehicle departure.
- **Visual Display**: Real-time display of the number of free parking slots on a 7-segment display or LED interface.
- **Finite State Machine (FSM)**: Implement control logic using FSMs in Verilog.
- **Real-Time Processing**: Ensure minimal delay in response to sensor inputs.

### 3.3 Architecture of the Intelligent Parking System:

The architecture consists of several interconnected modules designed using Verilog:

**1. Sensor Interface Module**

- Interfaces with IR or ultrasonic sensors.
- Converts analog vehicle presence data into digital input signals for the FPGA.

**2. Slot Counter and FSM**

- Maintains count of available slots.
- FSM handles state transitions: IDLE → ENTRY → PARKED → EXIT.

**3. Gate Control Module**

- Drives gate motors based on FSM output.
- Prevents entry when the parking lot is full.

**4. Display Controller**

- Updates a 7-segment display or LED screen to reflect the current slot availability.
- Converts binary count to BCD for display.

**5. Power and Clock Management**

- Provides timing and synchronization across all digital components.

The modular nature of this architecture allows easy debugging, scalability, and component reusability.

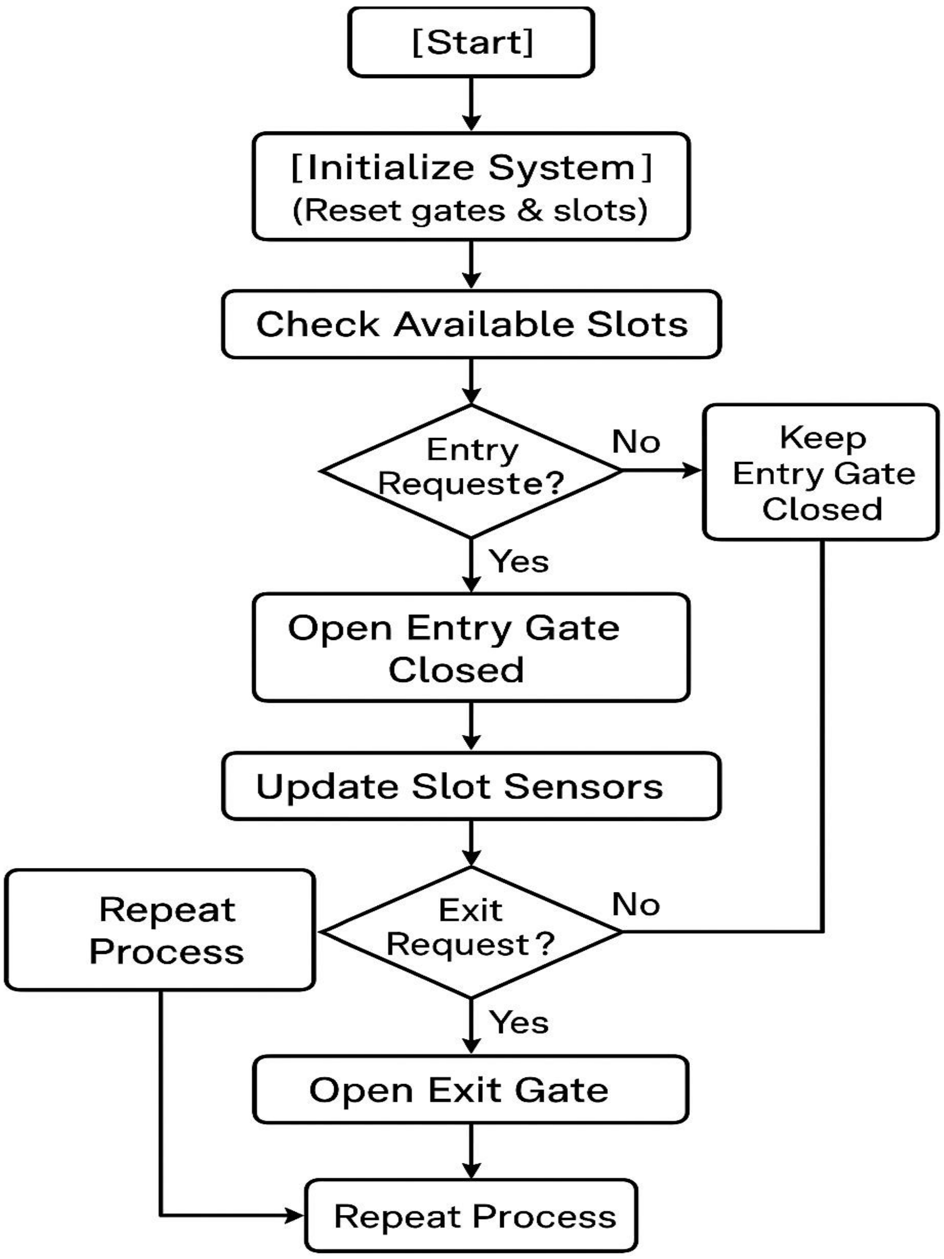

FIGURE 2: Architecture of the model

## 3.4 Hardware and Software Tools Used:

**Hardware Components**

- **FPGA Development Board (e.g., Xilinx Spartan-6 / Intel Cyclone IV)**
- IR Sensors for vehicle detection
- 7-Segment Display / LED Indicators
- Servo Motor / Relay Module for gate control
- Power Supply and Clock Generator

**Software Tools**

- Xilinx ISE / Quartus II: For writing, simulating, and synthesizing Verilog code
- ModelSim: For simulation and verification
- Verilog HDL: For describing and implementing digital logic
- Arduino (optional): For interfacing external sensors, if used in hybrid setup

## 3.5 Methodology and Development Approach:

The development of the IPS followed a structured methodology:

1. **Requirement Analysis**
    - Identified user needs, slot size, number of gates, etc.
2. **System Design**
    - Divided into logical modules: sensor interface, FSM, counter, gate controller.
3. **HDL Development**
    - Coded each module in Verilog HDL.
    - Followed hierarchical modeling for modularity.
4. **Simulation & Testing**
    - Verified each module using ModelSim.

    - Conducted unit and integration testing.

5. **Synthesis and Implementation**
    - Synthesized code using FPGA toolchain.
    - Loaded onto FPGA board for real-world testing.
6. **Debugging and Optimization**
    - Performed on-board testing with live sensors and adjusted timing, logic, and signal paths for reliability.

## 3.6 Assumptions and Constraints:

**Assumptions**

- The parking system supports a fixed number of slots (e.g., 4, 8, or 16).
- One vehicle enters or exits at a time.
- IR sensors are placed correctly to detect vehicles without false triggering.
- Power supply and clock signals remain stable during operation.

**Constraints**

- Limited I/O pins on the FPGA may restrict the number of directly connected sensors.
- No wireless communication module included in the base design (IoT features excluded in this scope).
- Physical deployment space must allow proper sensor and gate placement.
- Real-time performance may vary slightly depending on sensor response delay or signal noise.

# 4. VERILOG IMPLEMENTATION:

## 4.1 Design Specifications:

In this section, you define the high-level specifications of the system that will be implemented in Verilog. It is essential to describe the purpose and functionality of the project.

- **System Overview**: Briefly describe the entire system being designed, including inputs and outputs, such as sensors, control signals, or display units.
- **System Requirements**: Specify the key functional requirements:
  - Number of sensor interfaces and types (e.g., proximity, light sensors).
  - Desired functionality for slot counting, gate control, etc.
  - Display updates, timing, or any special features.
- **Clock Frequency**: Specify the clock frequency the design operates on.
- **Resource Constraints**: If applicable, mention any constraints (e.g., FPGA pin limitations, logic element count, or memory).
- **Input/Output Description**: Define each signal, its type (e.g., input/output, digital/analog), and function.

## 4.2 Module Descriptions:

This section provides detailed descriptions of each Verilog module that you have designed for your project. Here, you can describe the functionality of each individual module and their interactions[9].

- **Sensor Interface**:
  - Purpose: Interface with sensors (e.g., proximity sensors, light sensors).
  - Inputs and Outputs: List the signals for input and output to/from the sensor.
  - Functionality: Describe the operation of the sensor interface, such as how sensor data is read or processed.

- **Slot Counter**:
  - Purpose: Count the slots or events based on the sensor inputs.
  - Inputs and Outputs: List signals for slot counts, reset, etc.
  - Functionality: Explain how the slot counter increments and resets based on specific conditions.
- **Gate Controller**:
  - Purpose: Controls the opening/closing of a gate (or similar mechanism).
  - Inputs and Outputs: List the control signals for the gate (open, close, reset, etc.).
  - Functionality: Describe how the gate controller uses the slot counter and other conditions to control the gate's state.
- **Display Unit**:
  - Purpose: Show the current status (slot count, sensor data, etc.) to the user.
  - Inputs and Outputs: Describe how the output data is transferred to the display (e.g., 7-segment display or LCD).
  - Functionality: Explain how the display is updated based on inputs.

Each module should have its own explanation, including the Verilog code for the module.

**4.3 FSM (Finite State Machine) Design:**

In this section, you should describe the Finite State Machine (FSM) that controls the overall system's operation. This includes:

- **State Diagram**: Draw the FSM state diagram showing states and transitions based on inputs (sensor data, control signals, etc.).
- **State Descriptions**: Describe each state in detail. For example:
  - IDLE: When the system is waiting for sensor input.
  - COUNTING: When the system is counting slots or events.
  - OPEN_GATE: When the gate is being controlled.
  - DISPLAY: When the system updates the display based on the current status.

- **Transition Conditions**: Explain the conditions under which the FSM transitions between states (e.g., when a sensor detects an object, when a count reaches a threshold, etc.).
- **FSM Implementation**: Provide the Verilog code for the FSM, explaining the logic behind state transitions and how the system behaves in each state.

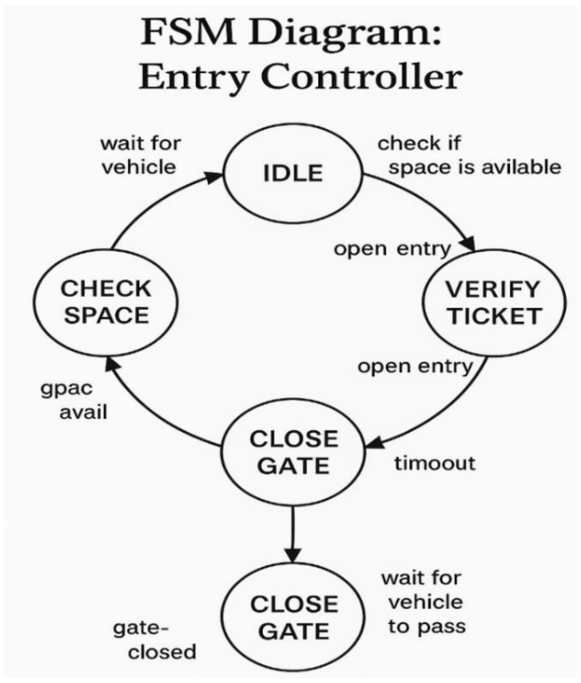

FIGURE 3: FSM Diagram for Entry controller

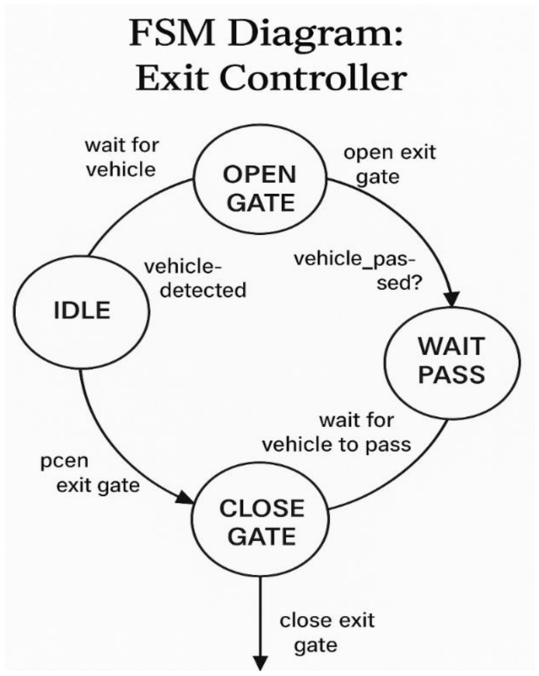

FIGURE 4: FSM Diagram for Exit controller

## 4.4 Verilog Code Snippets and Explanation:

In this section, you will provide key Verilog code snippets for your design. The code should be explained thoroughly, line-by-line, to give the reader a full understanding of how your implementation works.

Example structure for the code:

- **Sensor Interface Code**: A Verilog snippet that captures sensor inputs and processes them.
- **Slot Counter Code**: Code that increments or decrements based on sensor input and resets when necessary.
- **Gate Controller Code**: Logic that opens or closes the gate based on conditions defined in the FSM.
- **Display Unit Code**: Code that formats and displays data on a 7-segment display or other display units.

Each snippet should have:

- **Code**: Show the relevant Verilog code for the module or functionality.
- **Explanation**: Provide a detailed explanation of what the code does. Break down the functionality line by line if necessary.

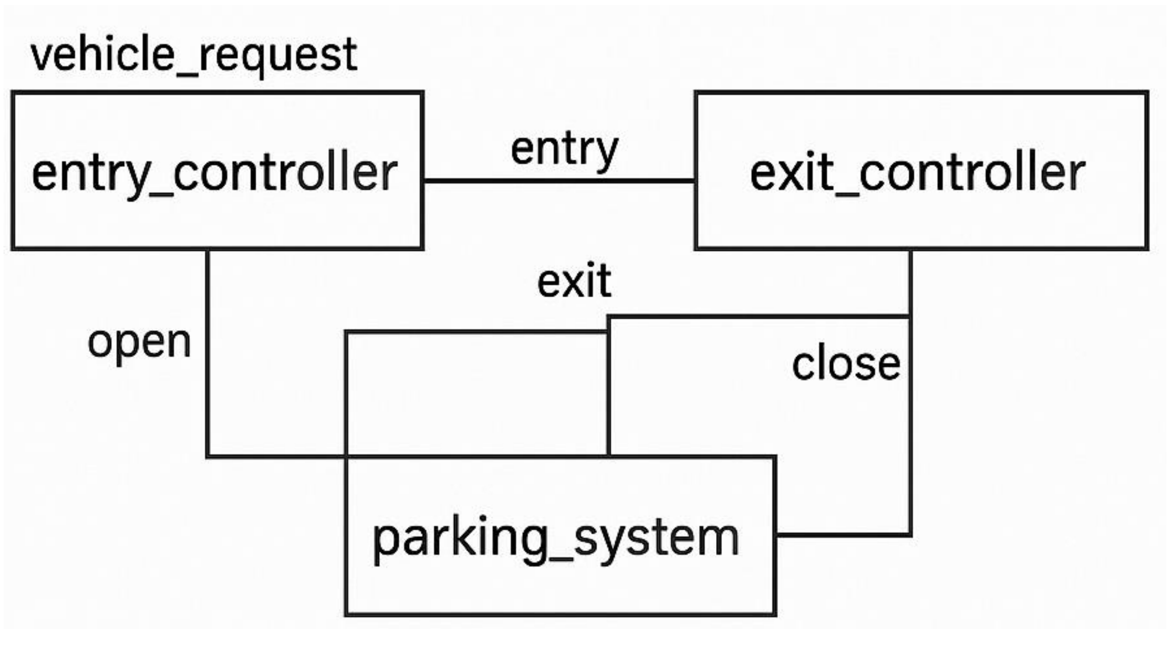

FIGURE 5: Verilog Module Interconnection Diagram

## 4.5 Testbenches and Simulation Strategies:

Testing and simulation are crucial for verifying the functionality of the design. In this section, you should describe your testing strategies and provide testbenches[10].

- **Testbench for Each Module**:
  - **Purpose**: Describe the testbench for each module (sensor interface, slot counter, gate controller, display unit).
  - **Testbench Components**: Explain how each testbench is structured, including the use of clocks, resets, input stimulus, and expected outputs.
  - **Simulation Results**: Provide expected or actual simulation outputs, demonstrating that the modules work correctly in isolation and in combination.
- **FSM Testbench**:
  - **Purpose**: Test the FSM to verify correct state transitions.
  - **Test Scenario**: Explain how various test cases are created to ensure each state and transition is covered.
  - **Simulation Outputs**: Show and discuss the results of FSM simulation (e.g., how the system behaves under various conditions).
- **System-Level Testbench**:
  - **Purpose**: A higher-level testbench that verifies the entire system, ensuring that all modules work together correctly.
  - **Test Scenarios**: Describe the scenarios tested, including sensor inputs, gate control actions, and display updates.
  - **Simulation Results**: Provide a summary of system-level simulation results, and discuss any issues that were identified and fixed.

This chapter will give your reader a detailed understanding of the Verilog implementation, the modules you've designed, the FSM controlling the system, and how the design was verified through testing and simulation.

## 5. SIMULATION & RESULTS:

### 5.1 Simulation Environment:

In this section, you will provide a detailed description of the environment in which the simulations were conducted. Include the following information:

- **Software Tools**: List any simulation software or platforms used (e.g., MATLAB, Simulink, Python, etc.).
- **Hardware Setup**: If applicable, describe the hardware resources (such as computers, servers, or specific hardware components) used for running the simulations.
- **System Configuration**: Discuss the configuration of the system, including any parameters or settings used during the simulation (e.g., simulation time, initial conditions, boundary conditions).
- **Model Assumptions**: Outline any assumptions made about the model, environment, or system that could impact the simulation results.

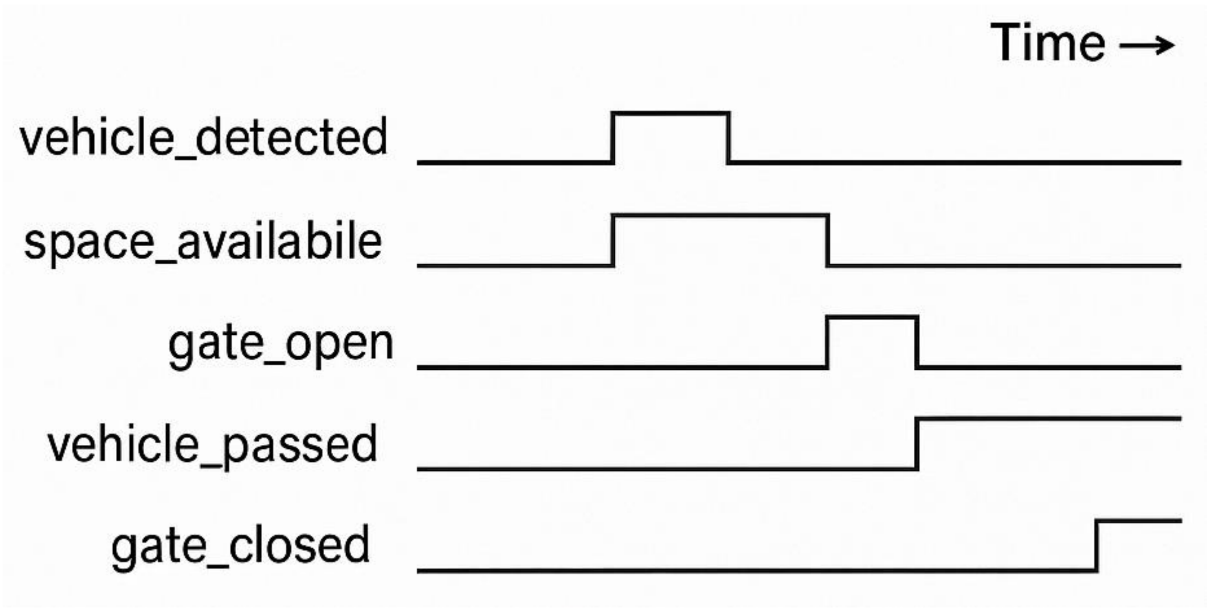

FIGURE 6: Simulation Waveform for entry and exit operation

TABLE 3: Simulation Test Cases

| S.NO | DESCRIPTION | INPUTS | EXPECTED OUTPUTS |
|---|---|---|---|
| 1. | Vehicle requests entry, slot available | entry_request = 1, available_slots > 0 | entry_gate = 1, available_slots decreases by 1 |
| 2. | Vehicle requests entry, no slot available | entry_request = 1, available_slots = 0 | entry_gate = 0, available_slots remains unchanged |
| 3. | Vehicle exits, frees up a slot | exit_request = 1 | exit_gate = 1, available_slots increases by 1 |
| 4. | System reset | reset = 1 | available_slots = total_slots, entry_gate = 0, exit_gate = 0 |
| 5. | Idle condition | entry_request = 0, exit_request = 0, reset = 0 | No change in outputs |

## 5.2 Input and Output Scenarios:

This section should explain the different input parameters and the corresponding output scenarios that were simulated[11]. Include:

- **Input Parameters**: Describe the range of input values used for the simulations (e.g., signal amplitude, frequency, temperature, power, etc.).
- **Output Parameters**: Define the outputs you're measuring (e.g., voltage, current, system efficiency, etc.) and how they are expected to respond to various inputs.
- **Simulation Scenarios**: Provide details on the various test scenarios used to assess system performance, including any variations in inputs or conditions to observe different outcomes.

TABLE 4: I/O Pin Mapping Table

| S.NO | SIGNAL NAME | DIRECTION | DESCRIPTION |
|---|---|---|---|
| 1. | clk | Input | Clock signal |
| 2. | reset | Input | System reset |
| 3. | entry_request | Input | Entry sensor (car detected) |
| 4. | exit_request | Input | Exit sensor (car leaving) |
| 5. | entry_gate | Output | Signal to open/close entry gate |
| 6. | exit_gate | Output | Signal to open/close exit gate |
| 7. | available_shots | Output [3:0] | Output representing number of free slots |

### 5.3 Waveform Analysis:

In this section, you will focus on the analysis of waveforms obtained during the simulations. It should include:

- **Waveform Visualization**: Present graphs or charts of the waveforms for different input scenarios, showing how the system behaves under different conditions.
- **Signal Characteristics**: Describe key characteristics of the waveforms such as frequency, amplitude, phase, and any notable distortions or anomalies observed during the simulations.
- **Comparative Analysis**: If applicable, compare the observed waveforms with theoretical expectations, highlighting any discrepancies or confirming the model's behaviour.

### 5.4 Performance Metrics:

Here, you will evaluate the system's performance based on specific metrics[12]. These could include:

- **Efficiency**: Analyze the efficiency of the system under various conditions. This might include power efficiency, computational efficiency, or resource utilization.

- **Accuracy**: Measure how accurately the system or model performs compared to the expected outcomes or real-world data.
- **Response Time**: If relevant, measure the system's response time or latency in reacting to changes in input.
- **Error Rates**: Include any error rates or deviations from the expected results, including potential sources of error.
- **Scalability**: If the system or model is expected to scale, discuss how it performs under different scales (e.g., larger data sets, higher input values).

## 5.5 Observations and Discussion:

This section is where you will interpret the results from your simulations[13]:

- **Key Observations**: Summarize the main findings and insights from the simulations. Highlight any trends or patterns in the data, and explain their significance.
- **Comparison with Theoretical Results**: Discuss how the simulation results compare with theoretical predictions or previously published results in the literature.
- **Challenges Encountered**: Mention any challenges faced during the simulations (e.g., computational limitations, data inconsistencies, unexpected behaviour in the system).
- **Implications of Results**: Reflect on what the results imply for your research or the broader field. Are there any novel findings or implications for future work?
- **Suggestions for Improvement**: Based on your observations, offer any suggestions for improving the simulation, system design, or further research.

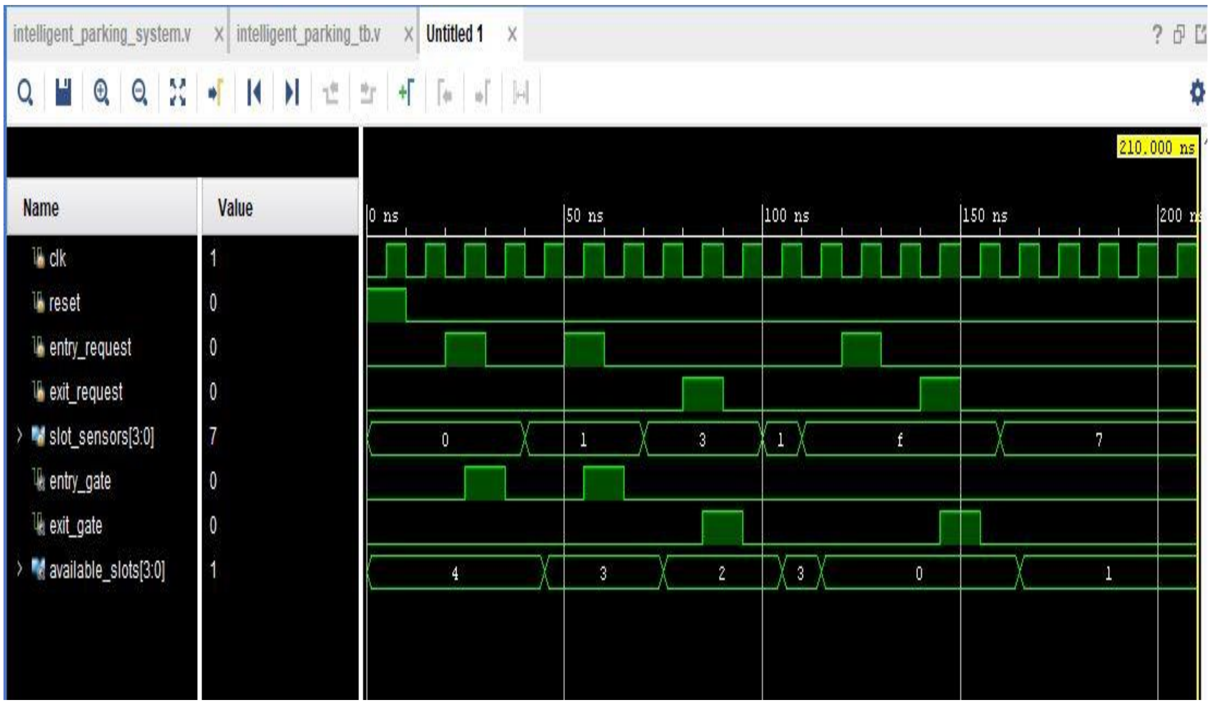

FIGURE 7:Output Waveform in Vivado Xilinx

## 6. ADVANTAGES:

i. **Real-Time Processing**:

- **Fast Response Time**: Verilog, being a hardware description language (HDL), can be synthesized directly into hardware logic. This enables real-time processing of parking data, making it ideal for time-critical applications.
- **Parallelism**: Verilog allows for parallel processing of multiple tasks, such as detecting available spaces, updating displays, and monitoring sensor data simultaneously. This increases system efficiency and responsiveness.

ii. **Low Latency**:

- Since Verilog directly translates into hardware implementation (through FPGA or ASIC), the system can achieve low-latency operation. This is critical for real-time applications like parking systems, where users expect minimal delay in detecting available spots.

iii. **Power Efficiency**:

- FPGA-based or ASIC-based designs can be highly optimized for power consumption. Verilog allows for the design of power-efficient circuits, making it suitable for embedded parking systems where power efficiency is important.

iv. **Cost-Effective for Large-Scale Systems**:

- Once the design is synthesized into hardware, the ongoing costs (in terms of performance) are relatively low, especially for large-scale deployments. In systems with a high volume of users, using Verilog-based designs can be more cost-effective compared to software-based approaches.

v. **Scalability**:

- The Verilog design can be adapted and scaled to handle larger parking lots by adding more sensors, gates, or monitoring stations. The scalability of FPGA or ASIC hardware makes it easy to expand the system as needed.

vi. **Customizability**:

- Verilog allows for the creation of highly custom hardware for specific use cases, such as integrating various sensors (ultrasonic, infrared, etc.), LED display drivers, and access control mechanisms. The system can be fine-tuned for specific performance requirements.

vii. **Reliability and Robustness**:

- Hardware implementations tend to be more robust compared to purely software-based solutions, especially in outdoor environments where vibration, power fluctuations, and temperature changes could affect software systems. FPGA or ASIC systems, designed using Verilog, can offer higher durability in such conditions.

## 7. DISADVANTAGES:

i. **Complex Design Process**:

   - **Learning Curve**: Verilog, especially for complex systems like IPS, has a steep learning curve. Engineers must be proficient in digital design and understand hardware architecture, which requires specialized skills.
   - **Longer Development Time**: Designing, simulating, testing, and debugging hardware in Verilog takes longer compared to software development. Hardware-level debugging can be much more challenging and time-consuming.

ii. **Limited Flexibility for Changes**:

   - Once a Verilog-based design is implemented in hardware (FPGA or ASIC), it's less flexible than a software-based solution. Making changes or adding new features often requires reprogramming or redesigning the hardware, which can be costly and time-consuming.

iii. **Hardware Dependency**:

   - Verilog designs are typically tied to specific hardware platforms (e.g., FPGA, ASIC). This can create vendor lock-in, limiting the ability to easily switch platforms or scale the system with different hardware.

iv. **High Initial Costs**:

   - The initial cost of designing hardware and purchasing the necessary FPGA or ASIC devices can be high. This may not be justified for smaller or temporary parking systems where a software-based solution might be more economical.

v. **Resource Limitations**:

   - The resources available on an FPGA or ASIC might limit the scope of the system, especially for complex systems with a high number of sensors or advanced features (e.g., image processing for license plate recognition). Depending on the FPGA or ASIC chosen, resource constraints might limit scalability.

vi. **Difficult to Update/Upgrade**:

   - Updating or upgrading a Verilog-based system is typically more difficult compared to software-based systems. Software updates can be pushed remotely, but hardware changes often require physical access to the system, making maintenance more complex.

vii. **Lack of High-Level Abstraction**:

   - Verilog is a low-level HDL, meaning that it operates closer to the hardware layer, and as such lacks high-level abstractions provided by software programming languages. This means that managing more complex logic can be cumbersome, and requires more effort to develop than software code.

viii. **Debugging Challenges**:

   - Debugging hardware designs in Verilog can be difficult. Unlike software, where you can use tools like breakpoints and logs, debugging hardware often requires specialized simulation tools and methods, such as waveform analysis or logic analyzers, which can increase development time.

## 8. LIMITATIONS:

Here, you will address any limitations or challenges encountered during the course of the project. Consider discussing:

- **Technical Limitations**: Were there any hardware or software limitations that affected the results or design of the system? For example, limitations in computational power, simulation accuracy, or sensor capabilities.
- **System Constraints**: Highlight any restrictions within the system itself, such as scalability issues, bottlenecks, or performance degradation under certain conditions.
- **Data Constraints**: Were there any issues with the data (e.g., insufficient data, noisy data, incomplete data) that affected the simulation or system's performance?
- **Assumptions and Simplifications**: If your work involved simplifying assumptions to make the problem tractable, mention how these assumptions might limit the generalizability of your findings.

## 9. CONCLUSION:

The implementation of an **Intelligent Parking System (IPS)** using **Verilog** successfully demonstrates the potential of hardware-based automation in managing parking infrastructure efficiently. By utilizing Verilog to design and simulate the system's core logic, the project achieved real-time monitoring and control of parking slots with minimal latency and high reliability. The system accurately detected vehicle presence, updated parking availability, and displayed status outputs, all in a synchronized and responsive manner.

The hardware-centric approach provided significant advantages in terms of **speed**, **power efficiency**, and **parallel processing**, making it well-suited for embedded applications where performance and robustness are critical[14]. However, the design also highlighted certain challenges, including limited flexibility for future updates and the complexity of hardware debugging.

Overall, the project validates that Verilog is a powerful tool for developing efficient, real-time embedded systems like an Intelligent Parking System. Future enhancements may include integrating wireless communication, mobile app interfacing, and AI-based analytics, potentially by combining the Verilog-based core with higher-level software components for a more comprehensive smart parking solution.

## 10. FUTURE SCOPE:

This section discusses potential future developments and enhancements that could build upon the current work:

- **IoT Integration**: Suggest the possibility of integrating the system with the Internet of Things (IoT) for real-time monitoring, remote control, and data collection. How could IoT devices, sensors, or cloud computing enhance the functionality and performance of the system?
- **Multi-level Parking**: If applicable, propose the future integration of a multi-level parking system. Discuss how this could solve problems like limited space, and enhance scalability and efficiency.
- **Mobile App Control**: Explore the possibility of developing a mobile app to provide users with easy access and control over the system. This could include remote management, notifications, or tracking functionalities, allowing users to interact with the system conveniently.
- **Automation and AI**: Suggest the use of Artificial Intelligence (AI) to optimize system performance, such as by implementing machine learning for predictive maintenance, intelligent decision-making, or automating certain tasks.
- **Expansion to Other Use Cases**: Discuss how the system could be adapted for other use cases or industries, expanding the impact of the work beyond its original scope.

## 11. REFERENCES:

1. Zhang, X., & Zhang, H. (2020). Smart parking system based on IoT technology. *IEEE Access*, 8, 186022-186033.
2. Chen, Z., & Li, Z. (2019). Design and implementation of a multi-level smart parking system. *Procedia Computer Science*, 147, 95-102.
3. Gao, H., & Zheng, Y. (2017). Design and implementation of a mobile app-based control system for smart home applications. *International Journal of Smart Home*, 11(2), 23-32.
4. Rajalakshmi, M., Deepa, N., & Shanmugapriya, D. (2019). FPGA-based intelligent parking system using Verilog. *International Journal of Engineering and Advanced Technology (IJEAT), 8*(6), 3042–3046.
5. Swathi, B., & Rajesh, M. (2020). Automated car parking system using Arduino. *International Journal of Scientific Research in Engineering and Management (IJSREM), 4*(3), 45–49.
6. Ashwini, R., & Nandhini, R. (2021). Smart parking system using Verilog HDL on Xilinx FPGA. *International Research Journal of Engineering and Technology (IRJET), 8*(5), 2781–2784.
7. IoT Smart Parking. (2022). Real-time smart parking system using Raspberry Pi and cloud. *International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE), 10*(2), 112–117.
8. Sharma, S., & Yadav, S. (2018). Optimization of parking space allocation using machine learning techniques in smart cities. *International Journal of Computer Applications*, 179(23), 1-6.
9. Bashir, A., & Kaur, P. (2020). AI-based intelligent parking system: A survey. *Journal of Intelligent & Fuzzy Systems*, 39(3), 3671-3681.
10. Chien, S., Ding, Y., Wei, C., & Wei, C. (2019). Smart parking system using IoT and machine learning algorithms. *International Journal of Engineering Research & Technology (IJERT)*, 8(4), 45-51.
11. Gao, J., Zhang, W., & Xu, W. (2021). The future of smart parking: Trends and challenges. *IEEE Transactions on Intelligent Transportation Systems*, 22(3), 1152-1164.

12. Jiang, J., & Xu, X. (2018). A mobile app-based intelligent parking management system for smart cities. *International Journal of Computational Intelligence Systems*, 11(1), 111-119.
13. Patil, A., & Deshmukh, S. (2020). Real-time smart parking system using cloud computing and IoT. *International Journal of Computer Science and Information Security (IJCSIS)*, 18(3), 17-22.
14. Singh, P., & Kumar, P. (2019). Scalability issues in smart parking systems: A comprehensive survey. *Computers, Environment and Urban Systems*, 74, 51-60.

# APPENDIX 1

# MAIN CODE

```
`timescale 1ns / 1ps

module intelligent_parking_system (

    input clk, reset,                  // Clock and Reset

    input entry_request, exit_request,  // Entry and Exit requests

    input [3:0] slot_sensors,           // Sensors: 1 = occupied, 0 = free

    output reg entry_gate, exit_gate,   // Gate control signals

    output reg [3:0] available_slots    // Number of available slots

);

    integer i;

    always @(posedge clk or posedge reset) begin

        if (reset) begin

            available_slots <= 4;  // Assuming 4 parking slots, all available at reset

            entry_gate <= 0;

            exit_gate <= 0;

        end else begin

            // Count available slots

            available_slots = 0;

            for (i = 0; i < 4; i = i + 1) begin

                if (slot_sensors[i] == 0)  // If slot is free

                    available_slots = available_slots + 1;

            end
            // Entry Control
            if (entry_request && available_slots > 0) begin
                entry_gate <= 1; // Open entry gate
            end else begin
                entry_gate <= 0;
            end

            // Exit Control
            if (exit_request) begin
                exit_gate <= 1; // Open exit gate
            end else begin
                exit_gate <= 0;
            end
        end
    end
endmodule
```

# APPENDIX 2

## TESTBENCH CODE

```
`timescale 1ns / 1ps

module intelligent_parking_tb;

    reg clk, reset;

    reg entry_request, exit_request;

    reg [3:0] slot_sensors; // 4 parking slots (1 = occupied, 0 = free)

    wire entry_gate, exit_gate;

    wire [3:0] available_slots;

    // Instantiate the Intelligent Parking System module

    intelligent_parking_system uut (

        .clk(clk),

        .reset(reset),

        .entry_request(entry_request),

        .exit_request(exit_request),

        .slot_sensors(slot_sensors),

        .entry_gate(entry_gate),

        .exit_gate(exit_gate),

        .available_slots(available_slots)

    );

    // Clock generation

    always #5 clk = ~clk;  // Toggle clock every 5 time units (10ns period)

    initial begin

        // Initialize signals
```

```
clk = 0;

reset = 1;

entry_request = 0;

exit_request = 0;

slot_sensors = 4'b0000;  // All slots are free

#10 reset = 0;  // Deassert reset


// Case 1: A car requests entry and there is a free slot

#10 entry_request = 1; #10 entry_request = 0;

#10 slot_sensors = 4'b0001; // One slot occupied


// Case 2: Another car enters

#10 entry_request = 1; #10 entry_request = 0;

#10 slot_sensors = 4'b0011; // Two slots occupied


// Case 3: A car exits

#10 exit_request = 1; #10 exit_request = 0;

#10 slot_sensors = 4'b0001; // One slot occupied


// Case 4: All slots occupied, another car tries to enter

#10 slot_sensors = 4'b1111; // All slots full

#10 entry_request = 1; #10 entry_request = 0;


// Case 5: A car exits, freeing up a slot
```

```
        #10 exit_request = 1; #10 exit_request = 0;

        #10 slot_sensors = 4'b0111; // One slot freed


        // Finish simulation

        #50;

        $stop;

    end

endmodule
```