REAL-TIME SMART PARKING SLOT DETECTION AND NOTIFACTION SYSTEM USING ESP32

PROJECT REPORT

Submitted by

KN2025-F203865 **SHVETA S**

KN2025-F203878 **ANCY JESLIN G**

INSTITUTION/ORGANISATION NAME: TANSAM

GUIDE/INSTRUCTOR NAME: MADHAN M G

DATE: 19-09-2025

CHAPTER-1

ABSTRACT:

The rapid increase in the number of vehicles has led to growing challenges in urban parking management. Traditional parking systems often result in traffic congestion, time wastage, and inefficient utilization of available parking spaces. To address these issues, this project presents a Smart Parking Slot Monitoring System using an ESP32 microcontroller, IR sensors, and cloud integration. The system employs three IR sensors to monitor the occupancy status of three parking slots. Each slot is represented by an LED indicator, where the LED glows when the slot is occupied and turns off when empty. The ESP32 is connected to Wi-Fi, enabling the system to send real-time SMS alerts to the user whenever a slot status changes. In addition, the occupancy data is logged to InfluxDB Cloud, which allows for remote monitoring, analytics, and visualization of parking trends through dashboards. This project demonstrates a low-cost, scalable, and IoT-based smart parking solution. It provides immediate functionality for small parking areas and offers expandability to larger smart parking systems with multiple slots. The integration of cloud services ensures that the system is not only functional in real-time but also future-ready for advanced applications such as mobile app integration, automated gate control, and smart city deployment.

Keywords — Smart Parking, IoT, ESP32, IR Sensor, Cloud Monitoring, SMS Notification, InfluxDB, Real-time Data Logging, Smart City.

Goal:

To design and implement a Smart Parking Slot Monitoring System using ESP32, IR sensors, and cloud connectivity. The system detects whether each slot is occupied or empty, indicates status using LEDs, sends SMS alerts, and logs real-time data to InfluxDB for monitoring.

Components

Hardware:

- PCB
- Power supply ESP32 Development Board
- 3 × IR Sensors (to detect vehicle presence)
- 3 × LEDs (to indicate slot status: ON = Occupied, OFF = Empty)
- Jumper wires, Breadboard

Software:

- Arduino IDE (for ESP32 programming)
- WiFi (ESP32 connection)
- CircuitDigest SMS API
- InfluxDB Cloud (for data logging and visualization with Grafana/InfluxDB UI)

Outcome

1. **Functional Outcome**
    - The system detects the occupancy status of 3 parking slots using IR sensors.
    - Each slot is indicated by an LED:
        - **ON** → Slot Occupied

        - **OFF** → Slot Empty
    - When a vehicle arrives or leaves, the ESP32 immediately:
        - Sends an SMS alert to the registered mobile number.
        - Updates the data to InfluxDB Cloud for remote monitoring.
    - Users can monitor parking slot availability in real-time without physically being present at the location.
2. **Technical Outcome**
    - Real-time data is stored in the cloud using InfluxDB.
    - Parking trends and occupancy history can be visualized using dashboards (Grafana/InfluxDB UI).
    - The system demonstrates the integration of hardware (sensors & ESP32) with IoT cloud services.
3. **Expandable for Future Applications**
    - Can easily be scaled to more parking slots by adding extra IR sensors and LEDs.
    - Can be integrated with mobile apps or web dashboards for user-friendly access.
    - Supports smart city applications such as:
        - Automatic gate/barrier control.
        - Centralized parking management systems.
        - Payment and reservation features.

CHAPTER – 2

CIRCUIT DESIGN

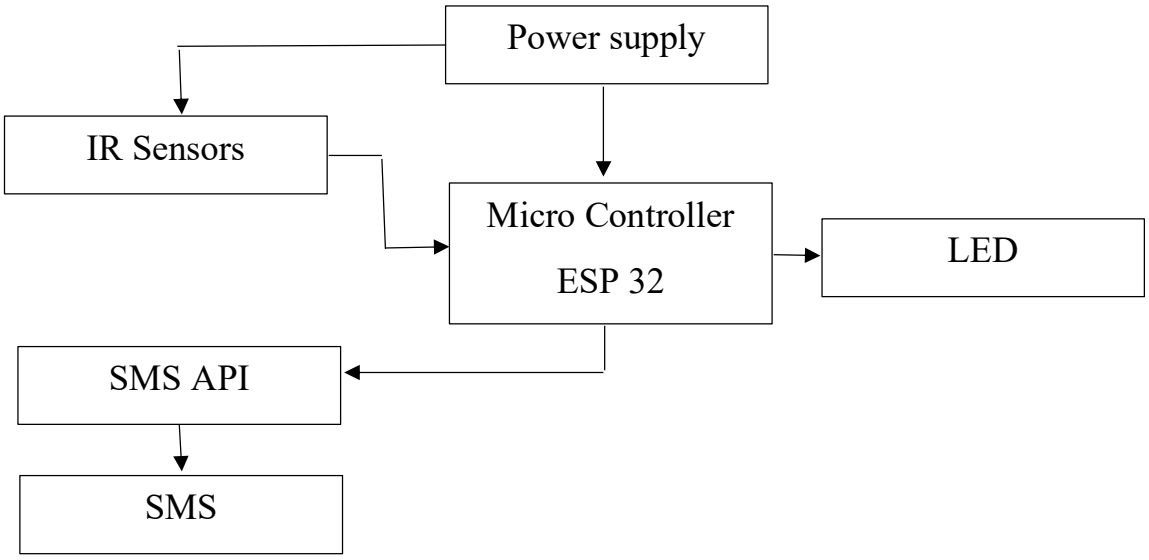

Fig 2.1: Block diagram of proposed system

1. ESP32 (Main Controller)

- The ESP32 is the brain of the circuit.
- It performs three key tasks:
    1. Reads input from IR sensors.
    2. Controls LED indicators to show slot status.
    3. Connects to Wi-Fi and sends SMS + Cloud Data (InfluxDB).
- It is chosen because it supports Wi-Fi + IoT applications natively, unlike simple Arduino boards.

2. IR Sensors (Slot Detection)

- Each parking slot has one IR sensor module.
- Wiring of each IR sensor:
  - VCC → 5V of ESP32
  - GND → GND of ESP32
  - OUT → GPIO pin of ESP32 (IR1 → GPIO 4, IR2 → GPIO 5, IR3 → GPIO 18)
- Working principle:
  - When no vehicle is present → IR light reflects, sensor output is HIGH (1).
  - When vehicle blocks IR → sensor output is LOW (0).
- This binary signal is read by ESP32 to decide if the slot is Empty (HIGH) or Occupied (LOW).

3. LED Indicators (Visual Feedback)

- Each slot has an LED assigned as an indicator light.
- Connections:
  - LED1 → GPIO 2
  - LED2 → GPIO 19
  - LED3 → GPIO 21
  - Each LED is connected with a current-limiting resistor (220Ω–330Ω) in series to prevent burning out.
- Operation:
  - LED ON → Slot Occupied
  - LED OFF → Slot Empty
- This allows people physically near the system to see availability at a glance.

4. Power Supply

- ESP32 and IR sensors are powered using a 5V regulated supply (via USB cable, power adapter, or power bank).
- The ESP32's onboard regulator provides 3.3V logic level internally.
- LEDs use ESP32 GPIO outputs, which safely drive them via resistors.

5. IoT Communication (Wi-Fi + Cloud)

- The ESP32 connects to Wi-Fi using the credentials coded in the program.
- Two cloud functions are performed:
    1. SMS Alerts:
        - Whenever slot status changes, ESP32 sends an HTTP POST request to the CircuitDigest SMS API.
        - The registered user immediately receives a message like *"Slot 2 Occupied"* or *"Slot 1 Empty"*.
    2. Cloud Data Logging (InfluxDB):
        - ESP32 sends slot status in Line Protocol format to InfluxDB Cloud.
        - Example:
        - parking_status,slot=1 occupied=1
        - parking_status,slot=2 occupied=0
        - This data can be visualized using InfluxDB dashboards or Grafana.

6. Overall System Workflow

1. Vehicle enters → IR sensor detects presence.
2. ESP32 updates slot status to "Occupied".
3. LED turns ON → Local visual indication.
4. ESP32 sends:
    - SMS Alert → User's Mobile.
    - Data Update → InfluxDB Cloud.
5. When vehicle leaves, the process repeats with status = Empty.

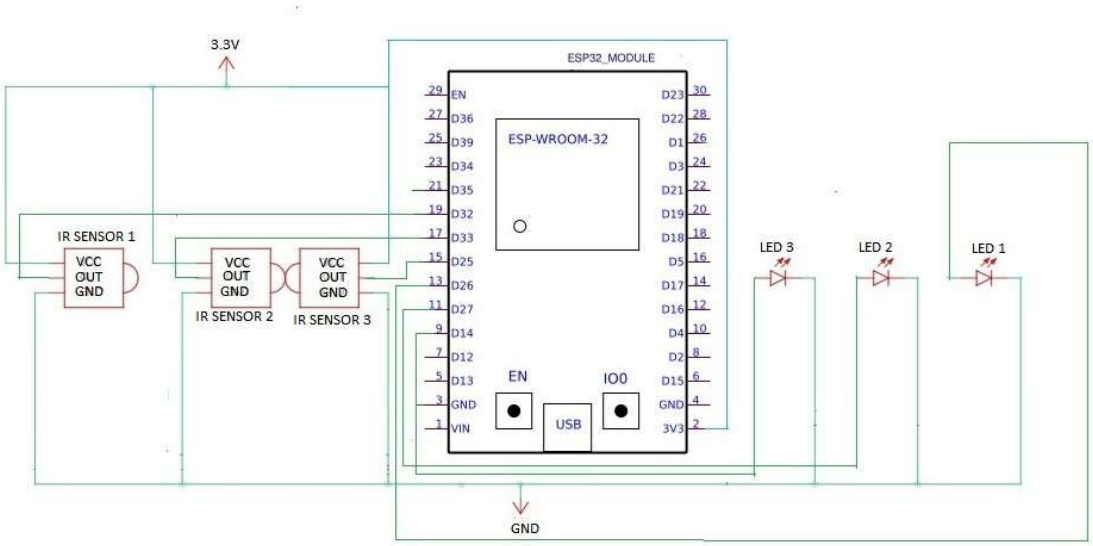

Fig 2.2: Circuit diagram of smart parking system

This circuit diagram shows a smart parking system using ESP32. Three IR sensors detect the presence of vehicles in different slots. Each sensor is connected to the ESP32 GPIO pins for input. Three LEDs act as indicators: ON means slot occupied, OFF means free. The ESP32 processes sensor signals and controls the LEDs accordingly.

CHAPTER 3

PROJECT CODE

```
#include <WiFi.h>

#include <HTTPClient.h>

#include <WiFiClientSecure.h>

// === WiFi Credentials ===

const char* ssid     = "XXXXX";

const char* password = "XXXXXXXX";

// === InfluxDB credentials ===

const char* influxURL = "https://us-east-1-
1.aws.cloud2.influxdata.com/api/v2/write?org=Data_base&bucket=esp32&precision=s";

const char* token     = "7geVbMvR9y7Xhu6EChuOs68-
yzQF4KGUoZFA2z2KlTzlZlygBIVy3mhQpS4zuPRCdXKCjGrFn4gZW3Ai4Ws-8A==";


// === CircuitDigest SMS credentials ===

const char* apiKey      = "ohPg6N85uYcH";       // replace

const char* templateID  = "101";   // replace

const char* mobileNumber= "91XXXXXXXX";       // replace


// === Hardware Pins ===

const int irPins[3]  = {32, 33, 25}; // IR sensor OUT pins

const int ledPins[3] = {26, 27, 14}; // LED pins
```

```
// Previous states to detect change
int prevStatus[3] = {-1, -1, -1}; // -1 means uninitialized


// --- URL-encode helper for SMS ---
String urlencode(const String &str) {
  String encodedString = "";
  char c;
  char code0, code1;
  for (int i = 0; i < str.length(); i++) {
    c = str.charAt(i);
    if (isalnum(c)) {
      encodedString += c;
    } else {
      code1 = (c & 0xf) + '0';
      if ((c & 0xf) > 9) code1 = (c & 0xf) - 10 + 'A';
      c = (c >> 4) & 0xf;
      code0 = c + '0';
      if (c > 9) code0 = c - 10 + 'A';
      encodedString += '%';
      encodedString += code0;
      encodedString += code1;
    }
  }
```

```
  return encodedString;
}
// --- SMS function ---
void sendSMS(const String &message) {
  WiFiClientSecure client;
  client.setInsecure(); // skip SSL certificate check
  HTTPClient https;
  String url = "https://circuitdigestapi.com/api/v1/?api_key=" + String(apiKey) +
               "&template_id=" + String(templateID) +
               "&mobile_number=" + String(mobileNumber) +
               "&var1=" + urlencode(message) +
               "&var2=" + urlencode(" ");  // blank if not used
  Serial.println("SMS URL: " + url);
  https.begin(client, url);
  int httpResponseCode = https.GET();
  Serial.print("SMS HTTP Response: ");
  Serial.println(httpResponseCode);
  https.end();
}
// --- InfluxDB function ---
void sendToInflux(int slot, bool free) {
  HTTPClient http;
  http.begin(influxURL);
```

```
  http.addHeader("Authorization", String("Token ") + token);
  http.addHeader("Content-Type", "text/plain");
  String data = "parking,slot=" + String(slot) + " status=" + String(free ? 1 : 0);
  int httpResponseCode = http.POST(data);
  Serial.print("InfluxDB HTTP Response (Slot ");
  Serial.print(slot);
  Serial.print("): ");
  Serial.println(httpResponseCode);
  http.end();
}
void setup() {
  Serial.begin(115200);
  Serial.println("Parking Slot Monitor Started");
  // Setup pins
  for (int i = 0; i < 3; i++) {
    pinMode(irPins[i], INPUT);
    pinMode(ledPins[i], OUTPUT);
    digitalWrite(ledPins[i], LOW);
  }
  // Connect WiFi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Connected! IP: ");
  Serial.println(WiFi.localIP());
}
void loop() {
  String smsMessage = ""; // build SMS message
  bool stateChanged = false;
  for (int i = 0; i < 3; i++) {
    int irValue = digitalRead(irPins[i]);
    bool slotFree = (irValue == HIGH); // HIGH = free, LOW = occupied
    // LED ON when free, OFF when occupied
    digitalWrite(ledPins[i], slotFree ? HIGH : LOW);
    // Serial output
    Serial.print("Slot ");
    Serial.print(i + 1);
    if (slotFree) {
      Serial.print(" Available  | LED=HIGH  ");
      smsMessage += "Slot " + String(i + 1) + " Available  | LED=HIGH  ";
    } else {
      Serial.print(" Occupied   | LED=LOW   ");
```

```
      smsMessage += "Slot " + String(i + 1) + " Occupied   | LED=LOW    ";
    }
    // Check if state changed
    int currentState = slotFree ? 1 : 0;
    if (prevStatus[i] != -1 && currentState != prevStatus[i]) {
      stateChanged = true;
    }
    prevStatus[i] = currentState;
    // Send to Influx
    sendToInflux(i + 1, slotFree);
  }
  Serial.println();
  // Send SMS once per loop if any slot changed
  if (stateChanged) {
    sendSMS(smsMessage); // send full line as var1
  }
  delay(2000); // update every 2 sec
}
```

CHAPTER 4

Result and Discussion

The proposed Smart Parking Slot Monitoring System (3 Slots) was successfully implemented and tested using ESP32, IR sensors, and LEDs. Each IR sensor accurately detected the presence of a vehicle in its respective slot, and the corresponding LED provided immediate visual feedback ON when occupied and OFF when empty. The ESP32 maintained a stable Wi-Fi connection and reliably performed two IoT functions: sending alerts to the registered mobile number whenever slot status changed, and logging real-time data to InfluxDB Cloud for remote monitoring and visualization. The results validate that the system meets its design objectives by providing local monitoring (LEDs) and remote monitoring (SMS + Cloud dashboards) simultaneously. During testing, the IR sensors responded quickly with negligible delay, SMS alerts were delivered within a few seconds, and cloud data updates were consistently accurate. This demonstrates the system's effectiveness as a low-cost and scalable smart parking solution**.** However, some limitations were observed. IR sensors may occasionally give false readings under strong ambient light or reflective surfaces, and the current prototype is limited to three slots. Despite this, the system can be easily expanded to multiple slots by adding more sensors and LEDs, and sensor accuracy can be improved through calibration or alternative technologies such as ultrasonic sensors.

Overall, the project demonstrates a functional, IoT-enabled smart parking system with practical applications for small parking lots and significant potential for future integration into smart city infrastructure. Possible extensions include mobile app integration, automated gate control, and support for reservation and payment systems.

OUTPUT

Fig 4.1: Smart Parking System Prototype using ESP32 and IR Sensors

This Fig 4.1 setup shows a prototype model of smart parking. An ESP32 microcontroller is used to collect parking slot data. IR sensors with LEDs detect whether a car is present or absent. Green LEDs indicate free slots, while Red LEDs show occupied slots. The detected data is sent to a database (InfluxDB) and visualized in Grafana.

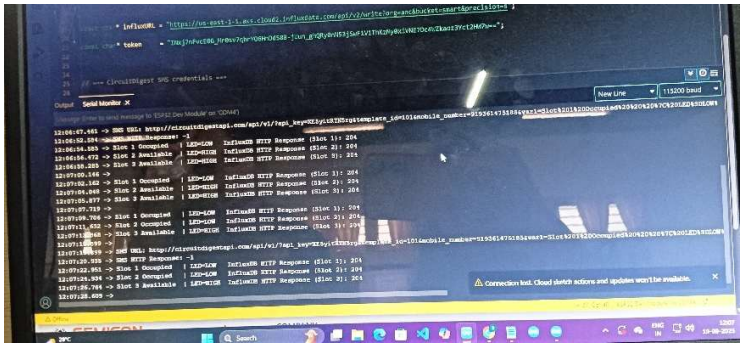

Fig 4.1: Serial Monitor Output of ESP32 – based Smart Parking System

Fig 4.1 output shows the Serial Monitor of ESP32 running a smart parking system code. It prints the status of each parking slot (Occupied or Available). LED status is also displayed (LED-LOW = Occupied, LED-HIGH = Available). Data is sent to InfluxDB with HTTP response code `204` (success). An SMS API link is also generated to notify users about slot availability.

Fig 4.3: Database Table Output of Parking Slot Status

This output shows the stored parking slot data from InfluxDB. Each entry contains the slot number, its status (0 = Free, 1 = Occupied), and a timestamp**.** The system has logged 836 rows of data over time. Users can view the data as a table or graph for analysis. This helps track real-time and historical parking usage efficiently.

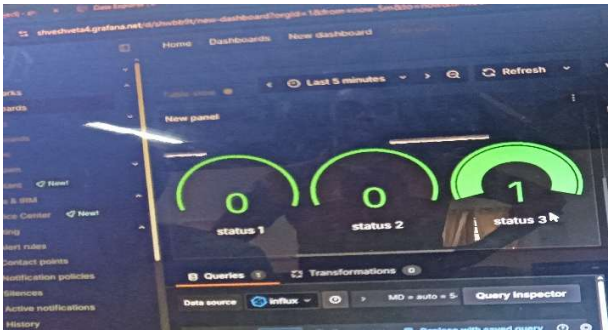

Fig 4.2: Smart Parking System Dashboard (Grafana + InfluxDB)

This Fig 4.2 shows that the Grafana dashboard panel setup screen. Three gauges are displayed, showing status 1 = 0, status 2 = 0, and status 3 = 1. The data source is InfluxDB, connected to real-time parking slot data. On the right, panel options like title, description, and visualization settings are visible.

Conclusion and Future Scope

This project successfully demonstrates a smart parking system prototype using ESP32, IR sensors, and Grafana visualization. It shows how real-time monitoring of parking slots can be achieved effectively. The system reduces time wasted in searching for free slots and enhances parking management. Data logging in InfluxDB ensures historical analysis of parking usage. Integration with IoT dashboards makes the system user-friendly and efficient. In the future, this model can be expanded to large-scale parking areas. Mobile applications can be developed to guide drivers to free slots. Integration with payment systems can enable automated billing. AI and image processing can replace sensors for more accuracy. The system can be connected to smart city infrastructure for better traffic control.