POLITECNICO
MILANO 1863

INTNET OF THINGS

Wokwi and Power Consumption

XU XUELI PERSONAL CODE:11075199

SUN YILIN PERSONAL CODE:11072044

CONTENTS

1 Introduction

This document provides a detailed description of the design concept and implementation process for a simple parking occupancy monitoring node using the Wokwi platform. Wokwi is a web-based platform for simulating and testing IoT and embedded system projects.

The primary function of the parking occupancy monitoring node is to detect whether a vehicle is parked in a parking space and send the detection results to a central ESP32 sink node. During the design process, special attention was paid to reducing power consumption and energy usage. The node is equipped with an HC-SR04 ultrasonic distance sensor and uses the ESP-NOW protocol for data transmission.

To optimize energy efficiency, adjustments were made to the transmission power, and tests were conducted by varying the duty cycle. Additionally, a comparative analysis was performed to examine the impact of static and dynamic nodes on the system's overall lifespan.

2 Design and Implementation using Wokwi

2.1 Overview of code

The overall coding approach revolves around creating an energy-efficient system that balances functionality with minimal power usage. The node wakes up every 54 seconds from deep sleep, performs a quick occupancy check using the HC-SR04 sensor, sends the status via ESP-NOW, and immediately returns to deep sleep.

To support energy consumption analysis, detailed timing measurements are integrated using esp_timer_get_time() and micros(), capturing the duration of each operational phase—boot, sensor idle, sensor active, WiFi setup, transmission, and WiFi shutdown.

The modular design enhances maintainability and clarity, breaking the code into distinct functional blocks. The initialization phase sets up libraries and defines constants like sensor pins, the 50 cm threshold, and the deep sleep duration (54 seconds). The timing measurement module, initiated early via a constructor (initBootTime()) and continued throughout setup(), tracks execution times for energy analysis, printing results when DEBUG is enabled. The sensor module configures the HC-SR04 and uses the isOccupied() function to determine occupancy, optimized for efficiency as described later. The communication module initializes ESP-NOW, registers the sink node's MAC address, and handles status transmission with callbacks (onDataSent()). Finally, the power management module ensures the system shuts down WiFi after transmission and enters deep sleep using esp_deep_sleep_start(), minimizing energy waste. This modularity allows each component to be independently tested or modified, such as adjusting the duty cycle.

2.2 Code Explanation

Function Name	Purpose	Logic Description

initBootTime()	Records the earliest boot timestamp for timing analysis	Executes as a constructor before setup(), captures initial time using esp_timer_get_time() to measure boot duration.
setup()	Main initialization and execution flow	Orchestrates the entire cycle: boots, configures sensor, checks occupancy, sets up ESP-NOW, sends status, enters deep sleep. Timing is recorded for each phase.
printTimestamp(String)	Logs timestamps and messages for debugging and timing analysis	If DEBUG is true, prints elapsed time (in µs) since boot plus a message, aiding in performance and energy tracking.
printEnergyTiming Summary()	Summarizes timing data for energy consumption estimation	Calculates and prints durations of each phase (e.g., sensor active, transmission) in milliseconds for power analysis.
setupESPNOW()	Initializes ESP-NOW communication protocol	Sets WiFi to STA mode, configures ESP-NOW, registers send/receive callbacks, and adds the sink node as a peer.
isOccupied()	Detects if the parking spot is occupied (distance ≤ 50 cm)	Sends an ultrasonic pulse, measures echo with a 3000 µs timeout; returns false if no echo (free), true if within 50 cm.
sendParkingStatus (const char*)	Transmits parking status ("FREE" or "OCCUPIED") via ESP-NOW	Sends the status string to the sink node's MAC address, logs success or failure via onDataSent()
onDataSent (const uint8_t*, esp_now_send_status_t)	Callback for transmission status	Logs whether the ESP-NOW message was sent successfully or failed, aiding in debugging communication issues.
OnDataRecv (const uint8_t*,	Callback for received data (if any)	Receives and prints incoming messages (not actively used in this sender-focused node but included for completeness).

const uint8_t*, int)		
loop()	Placeholder for continuous execution (unused due to deep sleep)	Empty, as the node enters deep sleep after setup(), preventing continuous operation to save power.

3 Estimation of power and energy consumption

code=11075199

Deep sleep configuration: 54 = (last two digits of personcode 11075199) 99 % 50 + 5

Battery energy: 5199%5000+15000 = 15199 [J]1

3.1 Power Consumption

To accurately estimate the power and energy consumption of the node, an approach involving the calculation of averages from the data provided in the set of CSV files is employed. These CSV files contain detailed timestamped power measurements for the ESP32 board captured at various operational stages, as described in the subsequent sections. By averaging these measurements, we can derive a more precise understanding of the node's power consumption behavior across different states.

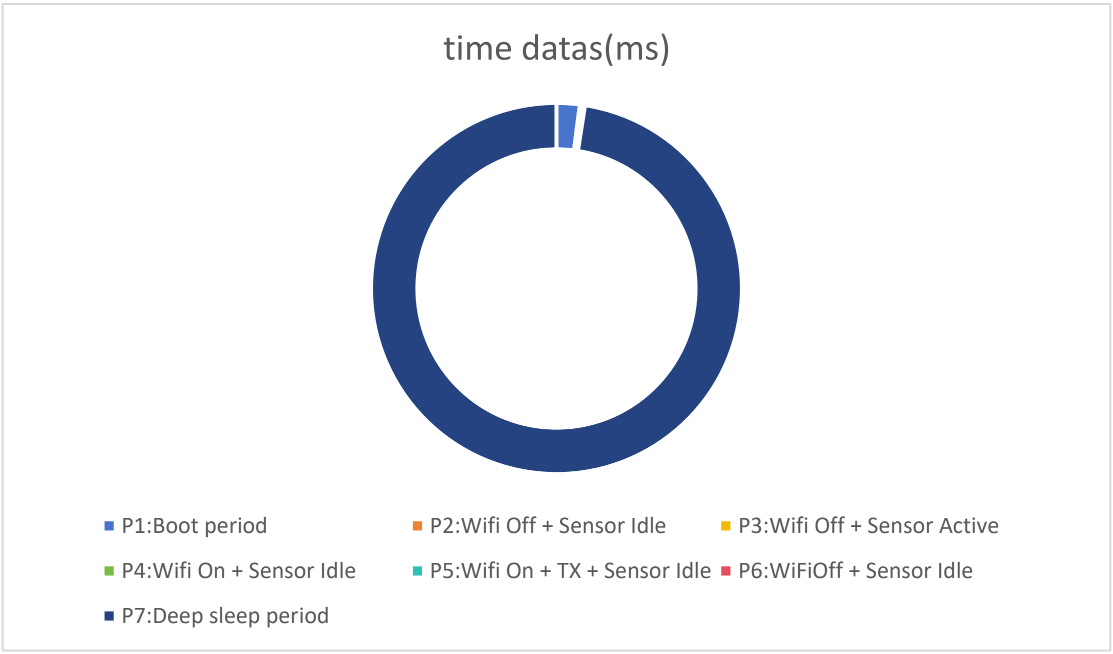
Average power consumption:

deepl_sleep.csv	**send different_TX.csv**	**read sensor.csv**
Deep Sleep: 59.62mW	TX Idle: 704.22mW	Sensor Idle: 331.59mW
Boot: 314.13mW	TX 2dBm:797.29mW	Sensor Active: 466.74mW
WiFi On: 776.62mW	TX19.5dBm: 1221.76mW	
WiFi Off: 308.27mW		

3.2 Time datas provided in Wokwi timestamps

The temporal data employed in this analysis originates from the timestamp feature embedded in the Wokwi simulation environment. To ensure the comprehensiveness and reliability of the outcomes, data collection was meticulously conducted across ten distinct sampling intervals. Following the acquisition of these samples, the data was processed to yield the following results.

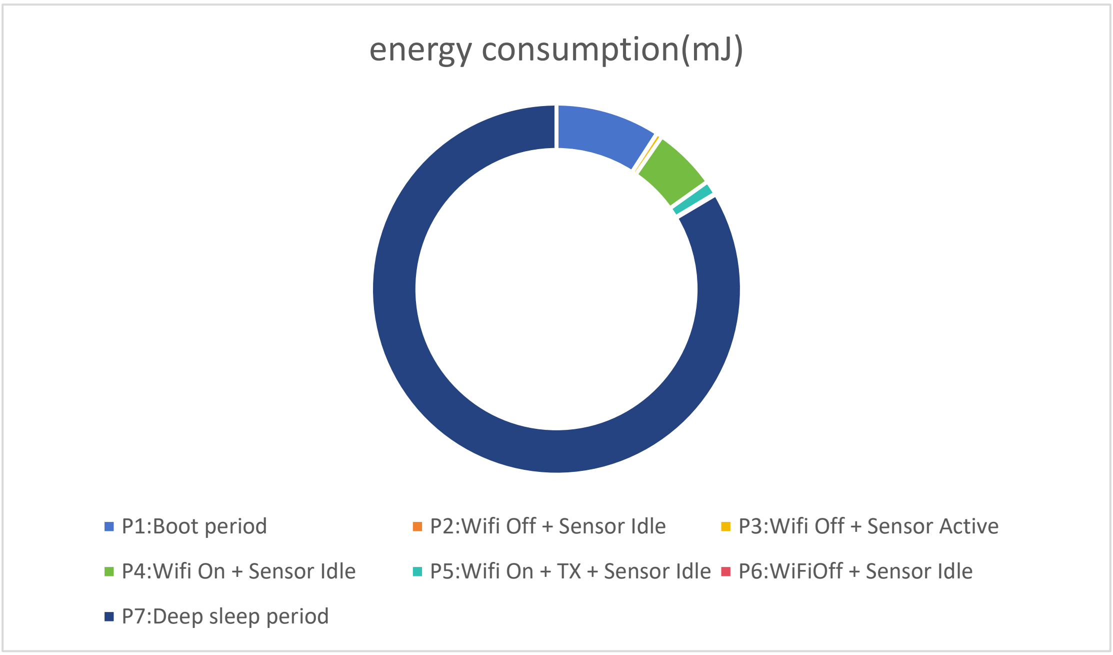
Time Period	data	description
P1:Boot period	1120 ms	Boot
P2:Wifi Off + Sensor Idle	0.65 ms	Sensor Idle
P3:Wifi Off + Sensor Active	24.77 ms	Sensor Active
P4:Wifi On + Sensor Idle	192.18 ms	WiFi On
P5:Wifi On + TX(19.5dBm) + Sensor Idle	20.04 ms	TX 19.5dBm
P6:WiFiOff + Sensor Idle	5.58 ms	WiFi Off
PA:Total active time	1363.22ms	--
P7:Deep sleep period	54000 ms	Deep Sleep
PT:One Cycle Time	55363.22 ms	Singal Cycle Period



3.3 Energy Consumption

To accurately determine the energy consumption of the node at each operational stage, we can combine the power consumption values provided in the CSV files with the corresponding time durations obtained from Wokwi for each stage. By multiplying the average power consumption (in milliwatts) for each stage by the duration (in milliseconds), we can calculate the energy consumption for that specific stage (in millijoules). This method enables us to gain a detailed understanding of how energy is utilized across different operational states of the node. The specific calculations and results are presented in the table below.

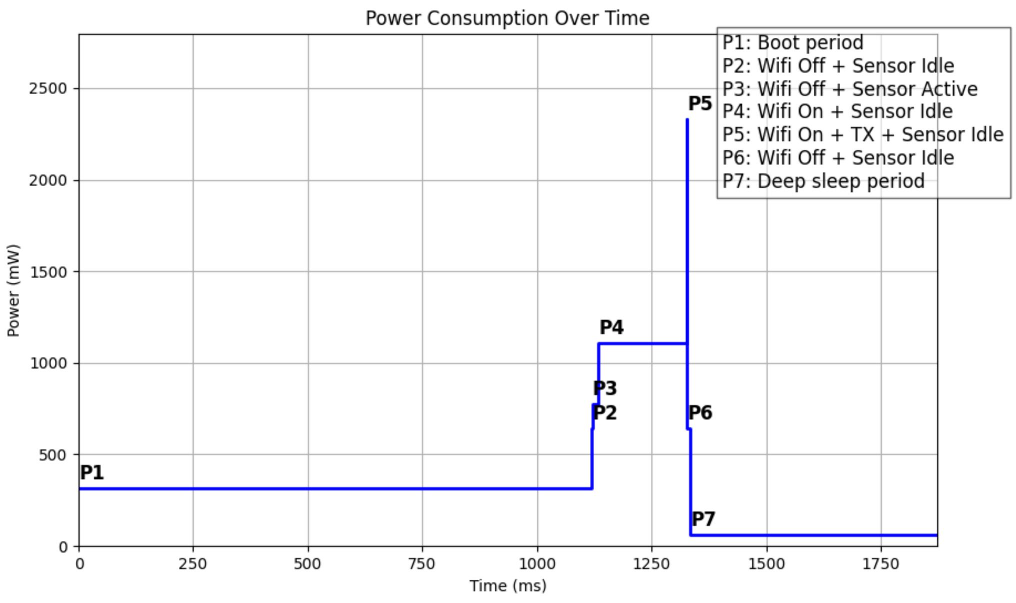
Time Period	time datas(ms)	energy consumption(mJ)
P1:Boot period	1120	351.81
P2:Wifi Off + Sensor Idle	0.65	0.42
P3:Wifi Off + Sensor Active	24.77	19.20
P4:Wifi On + Sensor Idle	192.18	212.98
P5:Wifi On + TX(19.5dBm) + Sensor	20.04	46.69
P6:WiFiOff + Sensor Idle	5.58	3.57
PA:Total active time	1363.22	634.67
P7:Deep sleep period	54000	3219.48
PT:One Cycle Time	55363.22	3854.15



Battery energy: 5199%5000+15000 = 15199 [J] (personcode:11075199)

Given the battery's energy consumption of 3854.15 milliJoules per cycle, it is estimated that the battery will last for approximately 3944 cycles.

One cycle need 55.36322s, then the battery can use 60.65 hours, equal to 2.52 days.



4 Improvements for Energy Consumption

1) Code optimization:

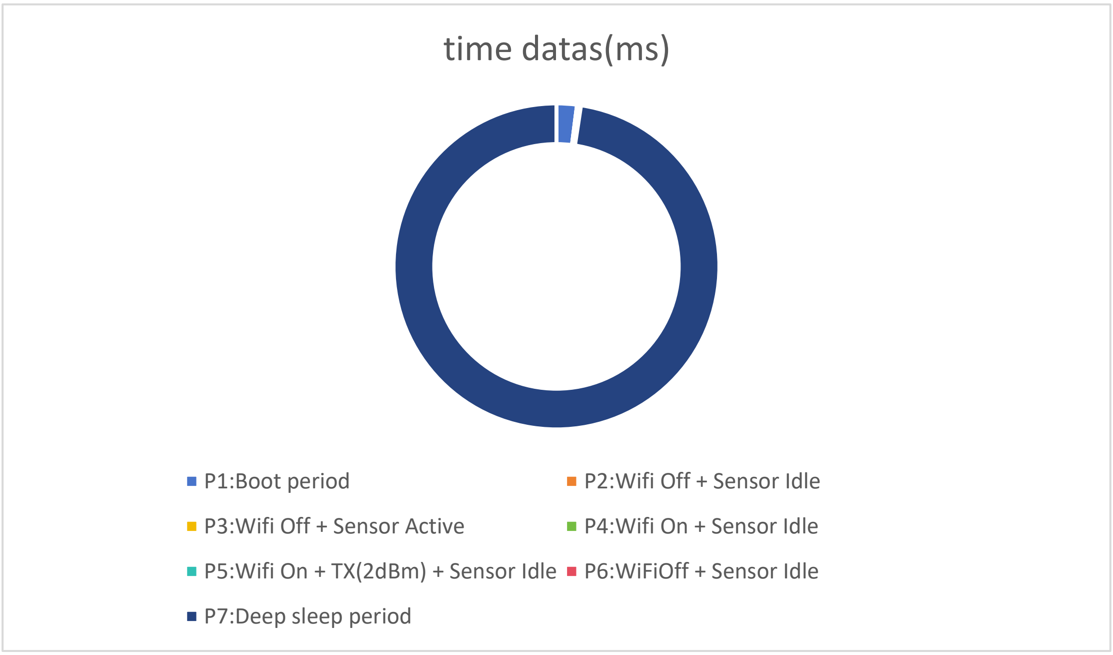
In the first Version of CODE, a key design consideration is the optimization for the 50 cm detection threshold, implemented in the isOccupied() function. The HC-SR04 measures distance by sending an ultrasonic pulse and timing its echo; sound travels at approximately 343 m/s (0.0343 cm/µs), so a 50 cm round trip (100 cm total) takes about 2915 µs. To save time and energy, the code sets an ECHO_TIMEOUT_US of 3000 µs—slightly above the theoretical value for reliability. Using pulseIn(ECHO_PIN, HIGH, ECHO_TIMEOUT_US), the function quickly determines if an object is within 50 cm: if no echo is received within 3000 µs (timeout), it assumes the spot is "FREE" and returns immediately, avoiding unnecessary delays; if an echo is detected, it calculates the distance and confirms occupancy. This optimization ensures the sensor active period is minimized, especially when the spot is unoccupied, directly reducing power consumption. The debug output confirms this efficiency by logging detection results, such as "No object detected within 50 cm" or the approximate distance when occupied.

Additional optimizations further enhance energy efficiency. The code minimizes active time by streamlining each phase: sensor configuration is brief, WiFi setup uses ESP-NOW's low-overhead protocol, and transmission sends only short strings ("FREE" or "OCCUPIED").

And we change Tx power from 19.5dbm to 2 dbm.

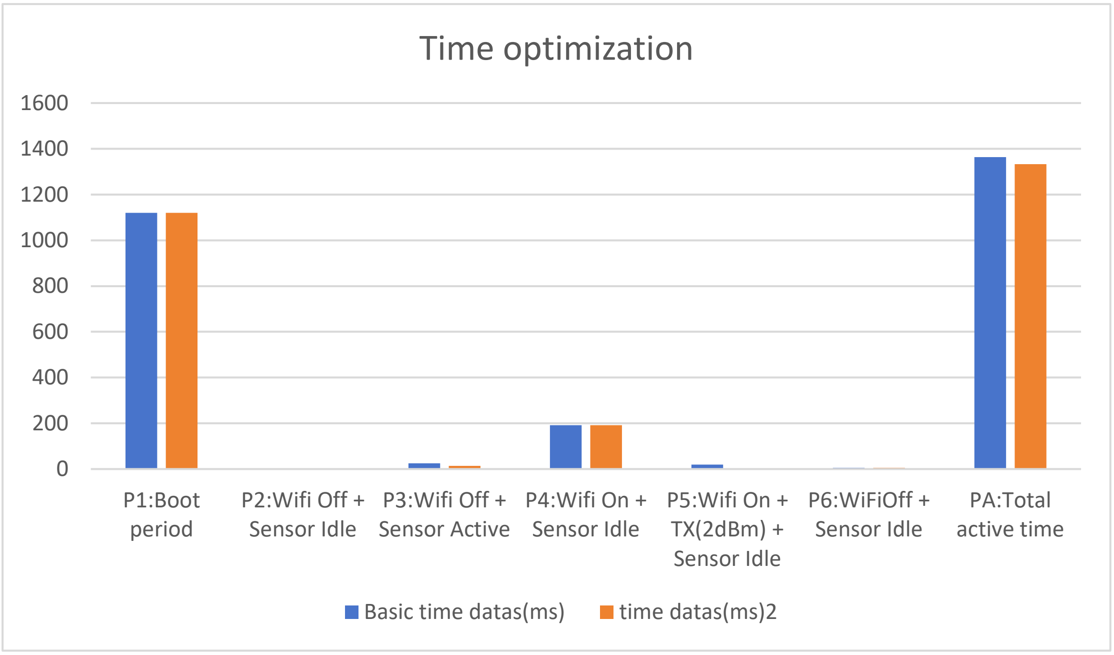
Time Period	**data**	**description**
P1:Boot period	1120 ms	Boot

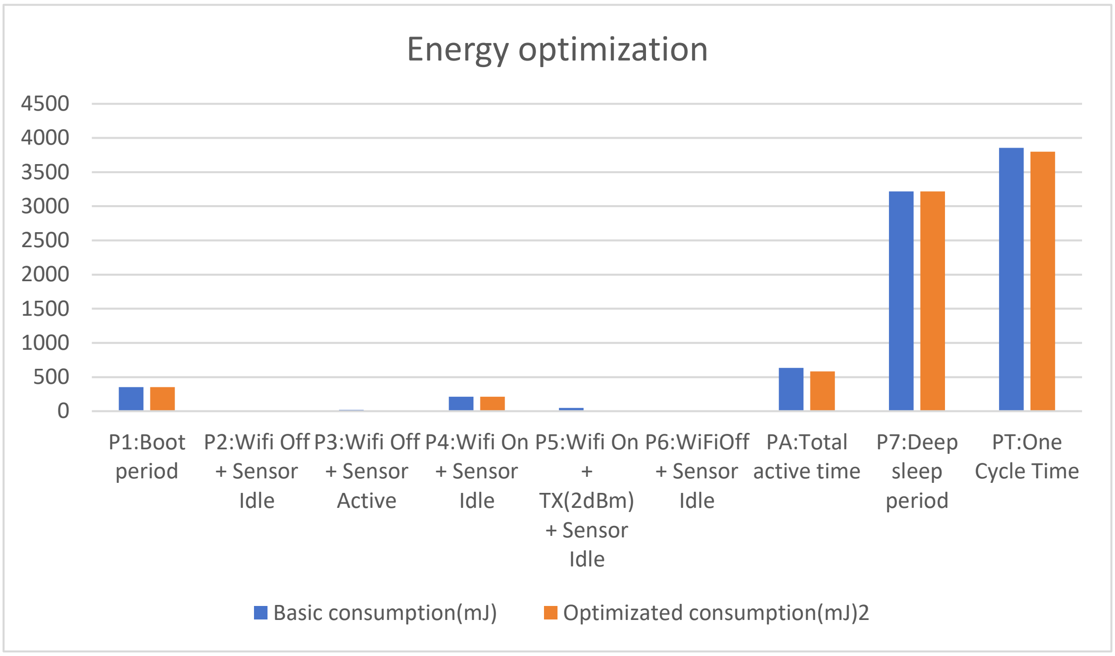
P2:Wifi Off + Sensor Idle	0.65 ms	Sensor Idle
P3:Wifi Off + Sensor Active	14.15 ms	Sensor Active
P4:Wifi On + Sensor Idle	192.18 ms	WiFi On
P5:Wifi On + TX(2dBm) + Sensor Idle	0.89 ms	TX 2dBm
P6:WiFiOff + Sensor Idle	5.58 ms	WiFi Off
PA:Total active time	1333.45 ms	--
P7:Deep sleep period	54000 ms	Deep Sleep
PT:One Cycle Time	55333.45 ms	Singal Cycle Period



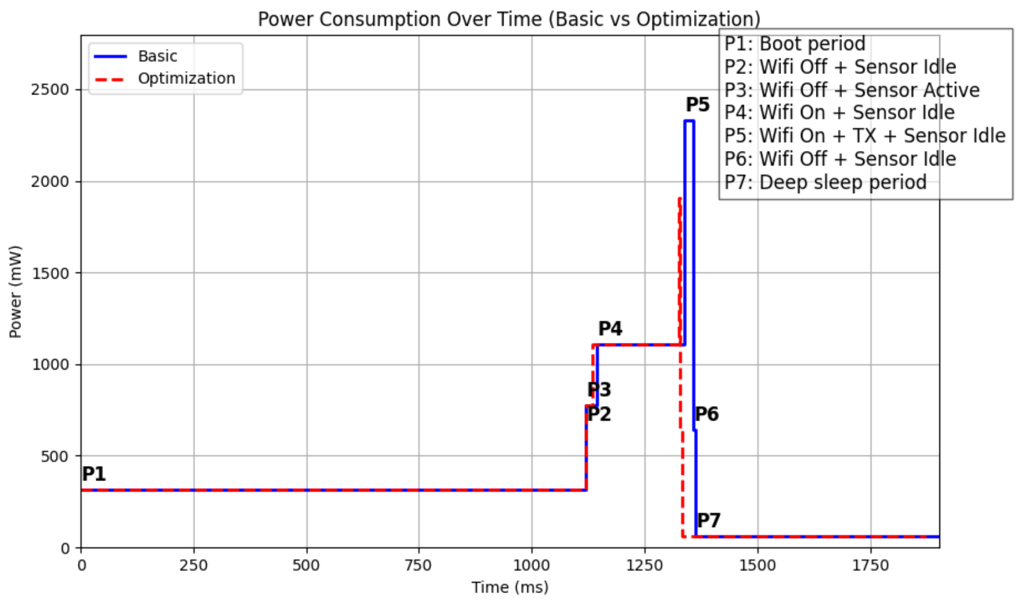
Time Period	**time datas(ms)**	**energy consumption(mJ)**
P1:Boot period	1120	351.81
P2:Wifi Off + Sensor Idle	0.65	0.42
P3:Wifi Off + Sensor Active	14.15	**10.97**

P4:Wifi On + Sensor Idle	192.18	212.98
P5:Wifi On + TX(2dBm) + Sensor Idle	0.8	**1.74**
P6:WiFiOff + Sensor Idle	5.58	3.57
PA:Total active time	1333.45	**581.48**
P7:Deep sleep period	54000	3219.48
PT:One Cycle Time	55333.45s	**3800.96**





Here are the comparison:



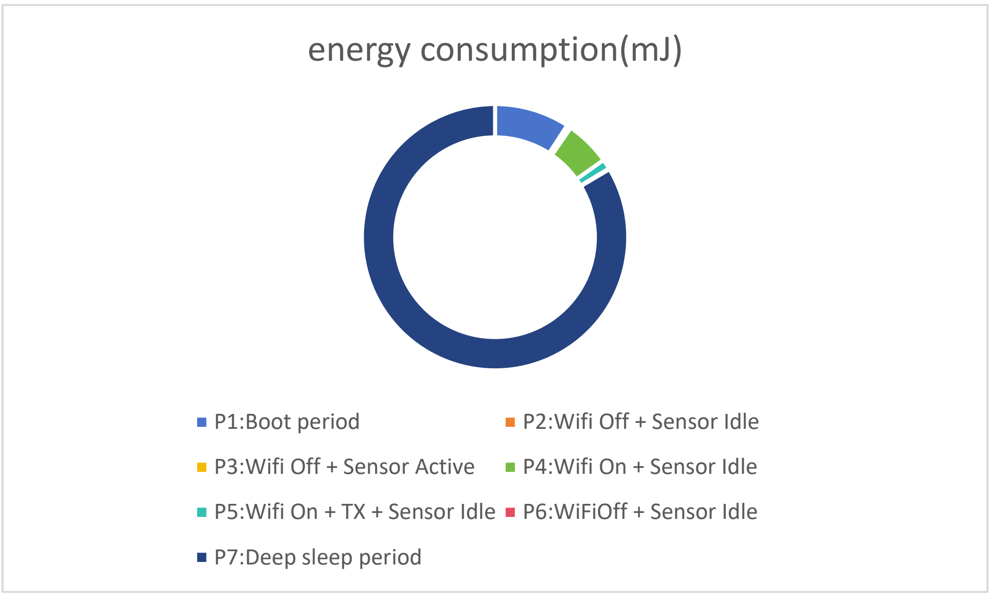
Energy Comparison:

Time Period	**Old Version Energy (mJ)**	**New Version Energy (mJ)**	**Optimization Reduction (mJ)**	**Optimization Ratio (%)**
P1:Boot period	351.81	351.81	0	0
P2:Wifi Off + Sensor Idle	0.42	0.42	0	0

P3:Wifi Off + Sensor Active	**19.20**	**10.97**	**8.23**	**-42.86**
P4:Wifi On + Sensor Idle	212.98	212.98	0	0
P5:Wifi On + TX + Sensor Idle	**46.69**	**1.74**	**44.96**	**-96.29**
P6:WiFiOff + Sensor Idle	3.57	3.57	0	0
PA:Total active time	**634.67**	**581.48**	**53.19**	**-8.38**
P7:Deep sleep period	3219.48	3219.48	0	0
PT:One Cycle Time	**3854.15**	**3800.96**	**53.19**	**-1.33**

Power comparison:

Time Period	**Old Version Power (mW/ms)**	**New Version Power (mW/ms)**	**Optimization Reduction (mW/ms)**	**Optimization Ratio (%)**
PA:Total active period	**0.4656**	**0.4361**	**0.0295**	**-6.33**
PT:One Cycle Time	**0.0696**	**0.0687**	**0.009**	**-1.33**



Results Analysis:

A. P3 (WiFi Off + Sensor Active):
Time: 24.77 ms → 14.15 ms (42.9% reduction)
Energy: 19.20 mJ → 10.97 mJ (8.23 mJ reduction)

B. P5 (WiFi Setup + TX):
Power: 2329.97 mW(19.5dBm) → 1950.5 mW(2dBm) (16.3% reduction)
Time: 20.04 ms → 0.89 ms (95.6% reduction)
Energy: 46.69 mJ → 1.74 mJ (44.96 mJ reduction, 96.3% decrease)

Conclusion:

This optimization primarily targeted ultrasonic sensor detection duration and WiFi transmission power reduction. By setting ECHO_TIMEOUT_US = 3000 μs, the system can quickly exit the sensor detection routine when no object is detected, significantly reducing the sensor's active time. As a result, the energy consumption of P3 (WiFi Off + Sensor Active) was reduced by 42.9%.

Additionally, lowering the WiFi transmission power from 19.5 dBm to 2 dBm led to a drastic reduction in P5 (WiFi Transmission) energy consumption, down by 96.3%. This phase contributed the most to overall energy savings.

From a broader perspective, the new version achieved an 8.38% reduction in total P1-P6 energy consumption, indicating that power optimization was successfully applied to the active operation phases. Although the DeepSleep power consumption remained unchanged, the reduction in active-phase energy caused its overall contribution to increase by 1.17%, further improving the device's power efficiency. Moreover, the average power consumption (excluding DeepSleep) was reduced by 6.33%, emphasizing that the optimizations effectively decreased energy usage during active states.

Overall, this optimization successfully reduced energy consumption in sensor detection and WiFi transmission, improving device efficiency and increasing DeepSleep's share in total power consumption. Future optimizations could focus on boot process efficiency, WiFi connection strategy improvements, and intelligent data transmission methods to further extend device battery life.

2) Duty Cycle Adjustment:

Adjust the duty cycle according to real-time traffic data and predictive models to meet the needs of different time periods. Implement a feedback mechanism to automatically adjust the duty cycle based on energy consumption and performance data. Additionally, selecting an appropriate duty cycle, such as increasing sleep time when it's not busy, can extend battery life. The following table shows that when sleep time is appropriately increased, battery usage time grows. Of course, beyond a certain range, battery life will also decrease.

Deep sleep time(s)	Energy spent in deep sleep state(mJ)	Total energy(mJ)	Total time(s)	Number of cycles	Battery use days
30	1788. 6	2370. 08	30. 58	6412. 86	2. 27
54	3219. 48	3800. 96	54. 58	3998. 73	2. 53
100	5962. 00	6543. 48	100. 58	2322. 77	2. 70
200	8943. 00	9524. 48	150. 58	1595. 78	2. 78
300	11924. 00	12505. 48	200. 58	1215. 39	2. 82

3)Optimizing Duty Cycles:

Historical Data Analysis for Usage Patterns:Analyze historical parking data to identify periods of high and low parking utilization.Use this data to strategically adjust the duty cycles during different times of the day or week.For instance, during weekdays, the parking spots might be in higher demand compared to weekends, thus requiring more frequent checks.

Adaptive Algorithms for Real-Time Adjustments:Implement adaptive algorithms that can process real-time data to dynamically adjust the frequency of sensor node wake-ups.These algorithms can learn from ongoing parking patterns and adjust the duty cycle to optimize energy consumption without compromising the node's functionality.For example, if a parking spot is consistently occupied during morning hours and free in the evenings, the system can adjust its wake-up schedule accordingly.

4)Implementing Event-Based Wake-Up:

Utilizing motion sensors or other triggering mechanisms to wake up the node, rather than relying on timers, is an effective approach to reduce energy consumption. Conducting an energy efficiency analysis on the event-based wake-up mechanism is essential to determine its impact on overall energy consumption. This analysis includes assessing the energy consumption for waking up, processing, and returning to the sleep state, as well as the energy savings compared to timer-based wake-ups. Designing intelligent triggering algorithms can help reduce unnecessary energy consumption caused by false triggers.

5)Using low-power communication protocols:

Choose or develop a communication protocol designed for low power consumption, such as Bluetooth Low Power (BLE) or Zigbee. Implement protocol optimization, such as dynamically adjusting transmission power and rate to adapt to different communication conditions. Expand an article.