Power Consumption Reduction and Estimation Report

1. Introduction

This report summarizes the finalized methods used to (1) reduce the power consumption of the ESP8266-based inverter monitoring system, and (2) estimate its power consumption using a software-only state-based model. The goal of both efforts is to minimize energy usage during the system's periodic 5-second polling cycle while maintaining continuous Wi-Fi connectivity, stable cloud communication, and reliable inverter SIM polling.

2. Finalized Power Reduction Approach

2.1 Overview

The ESP8266 performs inverter SIM polling within roughly 0.5–1.0 seconds of each 5-second cycle. During the remaining idle time, unnecessary system activity results in excess power consumption unless explicitly reduced. The final optimization approach focuses on minimizing Wi-Fi modem power during idle periods while keeping the device fully connected and operational.

2.2 Wi-Fi Modem Sleep During Idle Periods

Method Used

The system now activates **Wi-Fi Light Modem Sleep** during the idle portion of every polling interval:

```
WiFi.setSleepMode(WIFI_LIGHT_SLEEP);
delay(idleMs);
WiFi.setSleepMode(WIFI_NONE_SLEEP);
```

This approach reduces the energy consumed by the Wi-Fi modem while ensuring the module stays associated with the access point, supporting continuous cloud communication without reconnection delays.

Reason for Using Modem Sleep

- Modem sleep is the **only Wi-Fi-compatible power-saving mode** that preserves the STA connection.
- It reduces Wi-Fi radio energy usage while maintaining immediate responsiveness.
- It is stable, watchdog-safe, and fully compatible with the system's HTTP and SIM polling requirements.
- It integrates cleanly into the existing 5-second polling architecture without restructuring communication logic.

Why Other Power Optimization Methods Were Not Used

Optimization Method	Why Not Used
Deep Sleep	Turns off CPU, WiFi, RAM → device restarts every wake. Breaks continuous polling, cloud uploads, and buffer state.
Full Modem Sleep	Disables WiFi radio → GET/POST and NTP fail, causing unreliable cloud communication.
WiFi ON/OFF Cycling	Reconnecting takes 2–6 seconds and uses high current spikes → increases total energy and breaks timing.
Peripheral Gating (UART/SPI off)	Saves very little; required peripherals cannot be turned off because they are used continuously.

3. Finalized Power Estimation Method

3.1 Purpose

The system requires a software-only method to estimate power usage, since the ESP8266 lacks built-in current sensors, The goal is **comparative accuracy** (before vs after optimization), not absolute hardware precision.

3.2 Estimation Principle

The estimator is based on a **state-based energy model**, where the ESP8266's power consumption depends on the operational state of the CPU and Wi-Fi subsystem. The system measures how long it remains in each state and multiplies these durations by typical current values for that state.

This method is widely used in embedded systems when external measurement hardware is not available.

3.3 Operational States and Current Parameters

Four final states are tracked:

State	Description	Current Used
Idle	CPU running, Wi-Fi associated	~70 mA
CPU Active	Compression, encryption, parsing	~90 mA
Wi-Fi Active	HTTP requests, RF bursts	~150 mA
Sleep	Wi-Fi modem sleep	~15 mA

Reason for Choosing These Parameters

- Values match **Espressif datasheet specifications** and stable community measurements.
- Each state represents a **distinct and well-understood power profile** of the ESP8266.
- The separation of idle, CPU, Wi-Fi, and sleep enables clear attribution of where power is used.
- These numbers offer **high consistency**, which is essential for comparing firmware changes.

3.4 Timing and State Classification

The estimator uses microsecond- and millisecond-level timers to measure state durations:

- `millis()` for idle and sleep durations.

- `micros()` for high-resolution CPU task timing.
- `millis()` around HTTP requests for Wi-Fi activity.

Sleep time is logged independently and explicitly removed from idle time to avoid double-counting.

3.5 Power Calculation Method

At regular reporting intervals, the estimator computes:

Average Current

$$I_{\text{avg}} = \frac{\sum(I_{\text{state}} \times t_{\text{state}})}{T_{\text{total}}}$$

Energy Consumption

$$E_{\text{mWh}} = I_{\text{avg}} \times 3.3 \times T_{\text{hours}}$$

These values provide a clear understanding of the device's long-term average consumption and how different states contribute to total energy use.

4. Power Optimization Result Justification

The average current dropped from **70 mA** → **33.34 mA**, and energy consumption decreased from **0.3208 mWh** → **0.1528 mWh**, showing a reduction of more than **52%**.

This improvement is justified because the optimized firmware introduced **light sleep**, allowing the ESP8266 to keep WiFi connected while significantly lowering current during idle periods. Previously, the device stayed fully awake between polling cycles, maintaining high idle current. After optimization, most of that time is spent in low-power sleep, which draws only ~15 mA, directly reducing the average power across each cycle. Thus, the measured drop is an expected and reliable outcome of shifting from continuous active mode to a sleep-dominant duty cycle.