The following resources are included to support your implementation of power optimization, fault recovery, and final integration testing on the ESP8266-based EcoWatt Device.

1. ESP8266 Power Optimization Guidelines & Script Support

You are expected to implement as many relevant power-saving techniques as possible to achieve measurable energy reductions. The table below outlines possible approaches and implementation suggestions. Some methods are optional or may be deprecated; select those that are compatible with your build and document your choices.

Technique	Description & Tip
Light CPU Idle	Use `delay()` in polling loops to reduce active power.
Dynamic Clock Scaling	Use `system_update_cpu_freq()` to switch between 160 MHz and 80 MHz.
Light Sleep Mode	Use `wifi_set_sleep_type(LIGHT_SLEEP_T)` and `WiFi.forceSleepBegin()`
Peripheral Gating	Power down peripherals (WiFi, UART, ADC) when not needed.
Deep Sleep Mode *(Optional)*	Use `ESP.deepSleep()` for shutdown between cycles (requires GPIO16–RST jumper).

Note: Not all techniques are compatible simultaneously. Explore their trade-offs and compatibility via the ESP8266 Technical Reference Manual.

Included Script: `power_logger.py`

A sample script using a USB-based INA219 sensor to log power draw over time. This is only an example for you to get a clear picture of the expectations. You can decide what is possible based on the resources available to you.

Even estimation based on code profiling, simulation, or approximate methods is allowed if hardware is not available, but the methodology must be described and justified.

```python
# power_logger.py
from ina219 import INA219
from ina219 import DeviceRangeError
import time

SHUNT_OHMS = 0.1
SAMPLE_INTERVAL = 2  # seconds

ina = INA219(SHUNT_OHMS)
ina.configure()

print("Timestamp, Voltage (V), Current (mA), Power (mW)")
while True:
    try:
        voltage = ina.voltage()
        current = ina.current()
        power = ina.power()
        print(f"{time.time()}, {voltage:.2f}, {current:.2f}, {power:.2f}")
        time.sleep(SAMPLE_INTERVAL)
    except DeviceRangeError:
        print("Current out of range")
```

This script can be used with a USB-connected INA219 breakout board or similar. Connect in series with NodeMCU USB power supply.

2. Fault Injection Utilities

These resources allow you to simulate failures and validate your system's fault-handling and recovery features.

Inverter SIM Fault Samples

- **Malformed CRC frames**
- **Truncated payloads**
- **Buffer overflow triggers**
- **Random byte garbage**

The “Inverter SIM cloud API endpoint and documentation” now includes these faults, allowing you to trigger and test your EcoWatt device with them.

3. FOTA Rollback Simulation Instructions

You are expected to simulate successful and failed firmware-over-the-air updates, including rollback scenarios.

Demo Instructions

- Begin with a known working firmware (`v1.0.0`)
- Simulate a FOTA update to `v1.1.0` with a bad hash → verify rollback
- Retry with correct hash → verify success
- Document each stage in your video

Use this as a template to create your own manifest files for different firmware builds. The hash must be calculated using SHA-256 of the raw binary.

```
{
  "version": "1.2.0",
  "size": 40960,
  "hash": "9e107d9d372bb6826bd81d3542a419d6",
  "chunk_size": 1024
}
```

4. Final System Integration Checklist

To help you organize your final test video and ensure all required features are demonstrated, use the checklist below.

Feature	Demonstrated (Y/N)	Notes
Data acquisition and buffering		
Secure transmission		

Remote configuration		
Command execution		
FOTA update (success)		
FOTA update (failure + rollback)		
Power optimization comparison		
Fault injection (network error)		
Fault injection (Inverter SIM)		