Department of Electronic & Telecommunication Engineering (EN)
Department of Biomedical Engineering (BM)
University of Moratuwa

Project Outline

EN4440 - Embedded Systems Engineering

Last Modified: Sep 16, 2025

TABLE OF CONTENTS

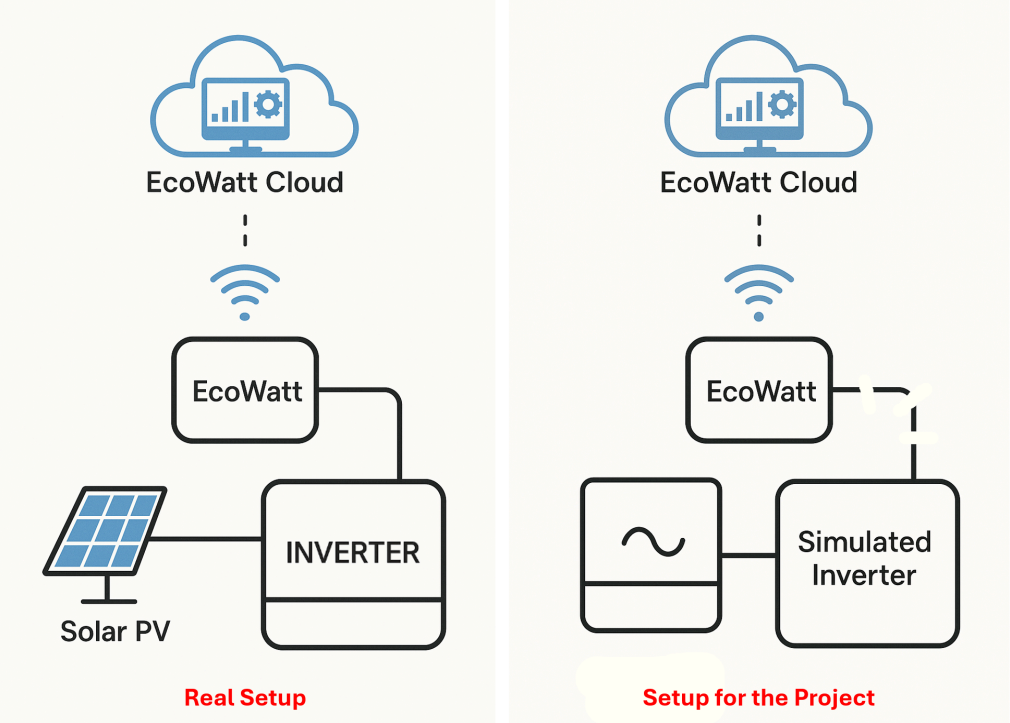
1. Project Overview

1.1. Background (in simple terms)

We are building a small embedded device called **EcoWatt Device** that pretends it is plugged into a real solar inverter. Because we don't have a physical inverter for the project, the device will talk to an online **Inverter SIM** service that behaves just like a real inverter would: it answers register read requests and accepts configuration write commands.

What we want the system to do (plain language):

- **Read a lot, send a little:** EcoWatt Device can read data (voltage, current, and later things like frequency) as often as we like, but it is only allowed to send data to the cloud once every 15 minutes. So it must store, shrink (compress), and package everything before the upload window.
- **Let EcoWatt Cloud stay in charge:** EcoWatt Cloud decides what to read, how often to read it, and can even push a new firmware image to the device. The device ("slave") simply follows those instructions safely.
- **Stay secure but lightweight:** We use small-footprint security (keys, message authentication, simple encryption) so the MCU can still run fast and save power.
- **Save power smartly:** The device should slow down its clock (DVFS) or sleep between jobs to cut its own energy use.
- **Be future-ready:** Later, we can swap the Inverter SIM for a real inverter with minimal code changes because we've cleanly separated the protocol layer.



This background sets the scene. The sections below provide a more detailed description of the project's scope and building blocks.

1.2. Core Objective

Build a microcontroller device called **EcoWatt Device** ("slave") that behaves like a field gateway. Instead of a physical inverter, it communicates with a **Simulated Inverter** called **Inverter SIM**. EcoWatt Device:

- Polls inverter registers (voltage, current, optional frequency, etc.) at a **configurable rate**.
- Buffers and **compresses** data locally.
- **Uploads once every 15 minutes** through a constrained link.
- Accepts **remote configuration & firmware updates (FOTA)** from the cloud "master."
- Implements **lightweight security** (auth, integrity, confidentiality) suitable for MCUs.
- Uses **power saving / DVFS & sleep** between tasks.

2. Actors & Channels

Three main pieces interact:

- **EcoWatt Device** (NodeMCU/ESP8266 MCU – the "slave")
- **EcoWatt Cloud** (configuration, commands, FOTA, data ingestion – the "master")
- **Inverter SIM** (the inverter emulator EcoWatt Device talks to instead of real hardware) – provided to you.

2.1. Inverter SIM Deployment Options

- Cloud API Mode – Inverter SIM runs as an HTTP/MQTT service online. EcoWatt Device reaches it over Wi-Fi just like it would reach EcoWatt Cloud. The inverter protocol is encapsulated inside these requests.
- PC Modbus Mode – Inverter SIM runs locally on a PC as a Modbus simulator. EcoWatt Device connects to the PC over USB (UART). From the device's point of view, it is still "speaking to an inverter" via a serial link.

For this project, we will use the Cloud API Mode. The other path should still be achievable later with limited changes—mainly in the transport/adapter layer.

2.2. Transport Constraints

- **EcoWatt Device → EcoWatt Cloud:** bulk payload upload strictly once every 15 minutes on a request-response setup.
- **EcoWatt Device ↔ Inverter SIM:** high-frequency polling allowed (configurable). Responses are stored locally and only forwarded (compressed) at the 15-minute.

2.3. EcoWatt Device Functional Blocks

- **Protocol Adapter Layer**
    - Implements inverter serial semantics over network to Inverter SIM.
    - Swappable for real RS-485 later (not in project scope)

- **Acquisition Scheduler**
  - Periodic reads (configurable from EcoWatt Cloud).
  - Aperiodic writes/controls triggered by EcoWatt Cloud.
- **Local Buffer & Compression**
  - Local buffer for samples.
  - Lightweight compression (delta/RLE or time-series scheme).
  - Optional aggregation (min/avg/max) to meet payload cap.
- **Uplink Packetizer**
  - 15-minute tick: finalize compressed block → encrypt/MAC → transmit.
  - Retry/chunk logic for unreliable links.
- **Remote Config & Command Handler**
  - Receives new sampling frequency, additional register list (e.g., add phase angle, active power, etc.), compression level, etc.
  - Applies changes without reflashing firmware (runtime config update).
- **FOTA (Firmware Over-The-Air) Module**
  - Secure, chunked firmware download over multiple intervals.
  - Integrity check (hash/signature), dual-bank/rollback.
- **Security Layer (Lightweight)**
  - Mutual auth (PSK + HMAC).
  - AES-CCM / ChaCha20-Poly1305 for payloads.
  - Anti-replay via nonces/sequence numbers.
  - Secure key storage appropriate for MCU constraints.
- **Power Management / DVFS**
  - Clock scaling when idle; sleep between polls/uploads.
  - Peripheral gating (disable comm blocks outside windows).
  - Optional self-energy-use reporting.
- **Diagnostics & Fault Handling**
  - Handle Inverter SIM timeouts, malformed frames, buffer overflow.
  - Local event log.

2.4. Cloud Components

- **Inverter SIM Service**
  - REST/MQTT endpoints mimicking register reads/writes, stateful behavior (ramps, faults) - provided to you.
- **EcoWatt Cloud Control Service**
  - Device registry, config push, command queue.
  - Receives compressed payloads, decrypts/decompresses, stores data.
  - Schedules and monitors FOTA rollouts.
- **Firmware Repository**
  - Stores signed binaries + manifests (version, size, hash).
- **(Optional) Analytics/Visualization**

- Dashboards for inverter metrics & device health.

2.5. Key Workflows

- **Acquisition Loop**
  - EcoWatt Device polls Inverter SIM at configured rate → buffers & compresses → sleeps/DVFS between polls.
- **15-Minute Upload Cycle**
  - Wake → finalize block → encrypt/HMAC → upload to EcoWatt Cloud → receive ACK + pending configs/commands.
- **Remote Configuration Update**
  - EcoWatt Cloud sends new config (e.g., add frequency/phase angle/active power).
  - EcoWatt Device validates & updates acquisition tables instantly (no reboot).
- **Firmware Update (FOTA)**
  - EcoWatt Cloud flags update → EcoWatt Device downloads chunks → verifies → swaps image → reports result.
- **Command Execution (Writes)**
  - EcoWatt Cloud queues write → delivered at next slot → EcoWatt Device forwards to Inverter SIM → returns status next slot.
- **Fault & Recovery**
  - Network/API errors → backoff & retry.
  - Integrity failures/tampered payloads → discard and alert.

3. Overview of Evaluation Criteria and Project Execution

3.1. Success Criteria / Acceptance Tests

- **Compression/Buffering:** All samples fit in the 15-minute payload; target compression ratio achieved.
- **Config Flexibility:** Sampling rate / signal list changed remotely without firmware rebuild.
- **FOTA Reliability:** Update with rollback safety demonstrated.
- **Security:** Replay/tamper tests fail cleanly.
- **Power Saving:** Demonstrated reduction in average MCU current with DVFS/sleep vs. naive always-on mode.
- **Protocol Abstraction:** Minimal code change to swap Inverter SIM for a real inverter later.

3.2. High-Level Project Phases

- **Scaffold & Transport:** Basic EcoWatt Device ↔ EcoWatt Cloud link, 15-min upload.
- **Inverter SIM Integration:** Protocol adapter + successful read/write round trips.
- **Buffering & Compression:** Implement and benchmark.
- **Config & Command Path:** Remote parameter change + write execution.

- **Security Layer:** Auth, encryption, MAC, anti-replay.
- **FOTA Pipeline:** Chunked download, verify, swap.
- **Power Optimization:** DVFS/sleep integration + measurement.
- **Integration & Testing:** End-to-end with fault & attack injection.

3.3. Milestone Breakdown and Marks Allocation

- **Milestone 1:** System Modeling with Petri Nets and Scaffold Implementation (10% of total project grade)
- **Milestone 2:** Inverter SIM Integration and Basic Acquisition (20% of total project grade)
- TBF

4. Milestone 1: System Modeling with Petri Nets and Scaffold Implementation

4.1. Objective

This milestone introduces you to **formal modeling and early-stage system design** using Petri Nets and a simplified scaffold implementation. You must:

- Use a Petri Net to model the behavior of a key workflow in the EcoWatt Device.
- Build a scaffold (basic prototype) of that workflow in software.

4.2. Scope

4.2.1. Part 1: Petri Net Modeling

You must create a Petri Net diagram modeling the following workflow:

- **Periodic polling** of the Inverter SIM to read parameters (e.g., voltage, current).
- **Buffering** of polled data in memory.
- **Uploading** buffered data to the cloud every 15 minutes.

To simplify testing and demonstration:

- You are allowed to simulate the 15-minute upload cycle using a 15-second interval.
- Your Petri Net should clearly show:
    - Places (states or resources)
    - Transitions (events or actions)
    - Tokens (data or events in motion)
    - Optional: Conditions/guards (e.g., timer expired, buffer full)
- You may use any tool (e.g., draw.io, Yasper, PIPE) or scan a clear hand-drawn diagram.

4.2.2. Part 2: Scaffold Implementation

You must implement a basic working prototype of the behavior you modeled. What the scaffold must do:

- **Poll the Inverter SIM (or simulate it):** Read or generate fake voltage/current values every few seconds.
- **Buffer the data:** Store the values in memory (e.g., a list, queue, or array).
- **Upload periodically:** Every 15 seconds (simulated), print or send the buffered data.
- **Modular code structure:** Organize code to reflect separate responsibilities for acquisition, buffering, and uploading.

Language Requirements:

- The scaffold implementation may be written in any language, such as Python, to simulate the logic easily.
- However, be aware that the final project will be implemented in C/C++ for NodeMCU/ESP8266.
- The structure of the scaffold should be designed with this future porting in mind.

4.2.3. Part 3: Demonstration Video

You must submit a 5–7 minute video that includes:

- **Petri Net walkthrough:**
  - Explain the logic and flow of your model.
  - Clearly describe places, transitions, and what tokens represent.
- **Scaffold code walkthrough:**
  - Explain how the code reflects your Petri Net model.
  - Identify which parts of the code map to which model elements.
- You must keep your video on throughout the demo.

4.2.4. What You Must Submit/Deliverables

- **Petri Net Diagram** (PDF or image)
- **Scaffold Code** (GitHub repo link or downloadable ZIP)
- **Video or the Video Link** (clearly accessible without having to log-in)
- Zip all these into a single file and upload to Moodle.

Notes:

- No report is required.
- You do **not** need to implement compression, encryption, cloud auth, or FOTA at this stage.
- Focus on clear structure and accurate modeling of system behavior.

4.3. Evaluation Rubric

Criterion	Weight
Accuracy and clarity of the Petri Net model	30%
Mapping between model and scaffold code	20%
Correct functioning of polling, buffering, and upload	20%
Clarity and completeness of the video explanation	15%
Code modularity and readability	15%

5. Milestone 2: Inverter SIM Integration and Basic Acquisition

5.1. Objective

This milestone integrates the EcoWatt Device with the Inverter SIM and establishes correct round-trip communication for reading and writing inverter registers. You must:

- Use the selected Inverter SIM deployment option to connect the EcoWatt Device to the Inverter SIM.
- Implement the protocol adapter that formats requests and parses responses.
- Build a basic acquisition scheduler that polls inverter registers and stores readings in EcoWatt Device memory.

5.2. Scope

5.2.1. Part 1: Protocol Adapter Implementation

You must implement the protocol adapter to allow the EcoWatt Device to speak to the Inverter SIM. The implementation must:

- Handle request formatting and response parsing.
- Detect and recover from timeouts and malformed frames.
- Log protocol errors and provide simple retry logic for transient failures.

You are allowed to simulate the Inverter SIM responses for parts of testing if the live SIM is unavailable.

5.2.2. Part 2: Basic Data Acquisition

You must implement an acquisition scheduler that:

- Polls a minimum set of registers, including voltage and current, at a configurable rate. You may poll once every 5 seconds for this milestone.
- The registers polled by the EcoWatt Device must also be based on the acquisition tables and configurable during runtime (see “Remote Configuration Update” under Section 2.5). This will be implemented in a later milestone.
- Stores raw samples in memory using a clear, modular structure separate from protocol code.
- Performs at least one write operation to the Inverter SIM.

5.2.3. Part 3: Demonstration Video

You must submit a 2 to 4-minute video that includes:

- Protocol adapter walkthrough that explains message formats and error handling.
- An acquisition code walkthrough that maps components to the design.
- Live demo of a successful read and write round-trip between EcoWatt Device and Inverter SIM.
- You must keep your video on throughout the demo.
- The video must be done by another member of the group who did not do the first milestone video.

5.2.4. Resources Provided

The following resources will be provided to assist with implementation:

- Inverter SIM cloud API endpoint and documentation (includes request/response formats and register map): Link

5.2.5. What You Must Submit: Deliverables

- Source code for protocol adapter and acquisition scheduler (you can submit a GitHub link).
- Short protocol documentation and test scenarios.
- Video or video link accessible without logging in.
- Zip all these into a single file and upload to Moodle.

5.3. Evaluation Rubric

Criterion	Weight
Correct functioning of read and write operations with the Inverter SIM	30%
Clarity and correctness of protocol adapter implementation	25%
Code modularity and readiness for porting to MCU	20%
Clarity and completeness of the video explanation and live demo	15%
Quality of documentation and test scenarios	10%

6. Milestone 3: Local Buffering, Compression, and Upload Cycle

6.1. Objective

This milestone implements local buffering and a lightweight compression scheme on the EcoWatt Device so that it can meet the upload payload constraint. It also implements the periodic upload cycle from the EcoWatt Device to the EcoWatt Cloud. You must:

- Design and implement a buffer that holds acquired samples for the upload interval.
- Implement a lightweight compression method and benchmark it (delta/RLE or time-series scheme) against the uncompressed code.
- Implement the upload packetizer and perform periodic uploads (every 15 minutes).

6.2. Scope

6.2.1. Part 1: Buffer Implementation

You must implement a local buffer that:

- Stores the full set of samples expected within the upload interval without data loss.
- It is modular and separable from the acquisition and transport code.

6.2.2. Part 2: Compression Algorithm and Benchmarking

You must implement at least one lightweight compression technique, such as delta/RLE or time-series scheme, and:

- Measure compression ratio and CPU overhead on data acquired from Inverter SIM
- Document test methodology and results in a short benchmark report. Fields to include;
    a. Compression Method Used:
    b. Number of Samples:
    c. Original Payload Size:
    d. Compressed Payload Size:
    e. Compression Ratio:
    f. CPU Time (if measurable):
    g. Lossless Recovery Verification:
- Verify lossless decompression and data integrity.
- Aggregation (min/avg/max) to meet payload cap.

6.2.3. Part 3: Uplink Packetizer and Upload Cycle

You must implement the packetizer that:

- Finalizes a compressed block at the upload window and prepares the payload for transmission.
- Implements retry/chunk logic for unreliable links.
- Uses a placeholder or stub for encryption at this stage since security implementation will be done in a future milestone.
- You may simulate the 15-minute upload interval using a 15-second interval for demonstrations.
- 15-minute tick: finalize compressed block → encrypt/MAC → transmit.
    - Overall workflow: Wake → finalize block → encrypt/HMAC → upload to EcoWatt Cloud → receive ACK + pending configs/commands.
- You must implement the cloud API endpoint that accepts the uploaded data and stores it. Students are expected to define both the client and server sides and implement the cloud API using Flask, NodeJS, or similar tools.

6.2.4. Part 4: Demonstration Video

You must submit a max 10-minute video that includes:

- Walkthrough of buffer design and its interaction with acquisition.
- Compression explanation and benchmark summary.

- Live demo of a complete acquisition to upload cycle, including payload size measurement.
- You must keep your video on throughout the demo. The demo video must be presented by a group member who has not presented in either of the previous milestone videos.

6.2.5. Resources Provided

The following resources will be provided to assist with implementation:

- Information required for the implementation:
    - Max payload size that can be uploaded to EcoWatt Cloud: No specific value is defined to reduce the complexity of the scope. Report the compression ratio achieved after implementing 6.2.2. Furthermore, you need to implement aggregation (min/avg/max) to meet the payload cap, but you are not required to use it when sending data.
    - You can assume that the local memory available on the EcoWatt Device is the same as the hardware that you are using (e.g., NodeMCU/ESP8266 MCU)

6.2.6. What You Must Submit: Deliverables

- Source code for buffer, compression, and packetizer modules.
- Cloud API endpoint and documentation (includes request/response formats and register map).
- Compression benchmark report with methodology and measured results.
    - You may use bullet points, a table, or a brief markdown-style report for the benchmark.
- Video or video link accessible without logging in.
- Zip all items into a single file for upload to Moodle.

6.3. Evaluation Rubric

Criterion	Weight
Correct buffering implementation and data integrity	20%
Compression efficiency and benchmark quality	20%
Robustness and correctness of the packetizer and upload logic	20%
Correct cloud API implementation	20%
Clarity of video demonstration, including payload measurements	10%
Code readability and modularity for porting to the MCU	10%

7. Milestone 4: Remote Configuration, Security Layer & FOTA

7.1. Objective

This milestone enables runtime remote configuration from EcoWatt Cloud to EcoWatt Device, introduces command execution in the Inverter SIM, implements a lightweight security layer, and establishes a secure firmware-over-the-air (FOTA) update mechanism. You must:

- Implement runtime configuration updates that take effect after the next upload without reboot.
- Implement command execution, where commands are executed in the Inverter SIM and results are reported back to the EcoWatt cloud.
- Implement a security layer for authentication, integrity, confidentiality, and anti-replay.
- Implement a FOTA module with secure download, verification, controlled reboot, and rollback.

7.2. Scope

7.2.1. Part 1: Remote Configuration

You must implement a runtime configuration mechanism that:

- Accepts update requests of acquisition parameters such as sampling frequency, additional register reads (e.g., phase angle, active power), and similar parameters.
- Applies each validated configuration update to all relevant subsystems (e.g., acquisition scheduler, register list). The parameters must take effect after the next upload at runtime without reboot or reflashing.
- It is expected that configuration updates are applied in a thread-safe or non-blocking manner that does not interrupt ongoing acquisition/other running processes.
- Handles duplicate or previously applied configurations gracefully (idempotent behavior).
- Stores accepted parameters in non-volatile memory to survive power cycles.
- Validates all new parameters and rejects unsafe/infeasible/invalid values with error reporting.
- Confirms to EcoWatt Cloud which parameters were successfully applied and which were rejected. The cloud must maintain reports/logs of when updates occurred and their outcomes (including which parameters were updated and which were not, etc.).
- Implement the EcoWatt Cloud behavior that facilitates this sequence. Using a Node-RED dashboard for this is acceptable.

7.2.2. Part 2 – Command Execution

You must implement a command execution flow that:

- EcoWatt Cloud queues a write command.
- At the next transmission slot, the EcoWatt Device receives the queued command.
- The EcoWatt Device forwards the command to the Inverter SIM.
- The Inverter SIM executes the command and generates a status response.
- At the following slot, the EcoWatt Device reports the execution result back to EcoWatt Cloud.
- EcoWatt Cloud must maintain logs of all commands and results for both EcoWatt Device and EcoWatt Cloud.

7.2.3. Part 3: Security Implementation

You must implement a practical, lightweight security layer suitable for MCU constraints that must not rely on heavyweight cryptographic libraries or hardware secure elements. It must include:

- Pre-shared key-based mutual authentication (PSK + HMAC).
- Payload confidentiality and integrity using a suitable encryption scheme for payloads.
- Anti-replay via nonces/sequence numbers.
- Secure key storage appropriate for MCU constraints.

7.2.4. Part 4: FOTA Module

You must implement a secure firmware-over-the-air mechanism that:

- Supports chunked firmware download across multiple communication intervals and resumes after interruptions.
- Verifies firmware integrity and authenticity.
- Uses a rollback approach so that if verification fails or the new firmware fails to boot, the device reverts to the previous firmware.
- Performs a controlled reboot only after successful verification.
- Reports progress, verification results, reboot status, and final boot confirmation to EcoWatt Cloud.
- Maintains detailed logs for all FOTA operations in the EcoWatt Cloud.
- Implement the EcoWatt Cloud behavior that facilitates this sequence.

7.2.5. Part 5: Demonstration Video

You must submit a max 10-minute video that includes:

- Configuration updates showing that the device verifies, correctly applies updates internally, and reports success/failure.
- Command execution round-trip (EcoWatt Cloud → EcoWatt Device → Inverter SIM → EcoWatt Cloud).
- Secure transmission demo showing authentication and integrity checks.
- Full FOTA update including chunked download, verification, controlled reboot, and confirmation of new firmware.
    - At least one FOTA demo must include a simulated failure and demonstrate a rollback to the previous version.
- You must keep your video on throughout the demo.
- The demo video must be presented by a group member who has not presented in either of the previous milestone videos.

7.2.6. Resources Provided

The following resources will be provided to assist with implementation:

- Milestone 4 - Resources - In21-EN4440-Project Outline

7.2.7. What You Must Submit: Deliverables

- Source code implementing the functionalities.
- Documentation detailing supported configuration parameters, how the device applies them internally, command execution flow, security design, and FOTA workflow, including rollback handling.
- Video or video link accessible without requiring a login.
- Zip all items into a single file for upload to Moodle.

7.3. Evaluation Rubric

Criterion	Weight
Successful runtime application of remote configuration parameters	15%
Correct and reliable command execution with reporting	15%
Correctness and completeness of security implementation	25%
Correctness and reliability of FOTA implementation, including rollback	30%
Clarity and completeness of the video explanation and live demo	10%
Quality of documentation and test scenarios	5%

8. Milestone 5: Fault Recovery, Power Optimization & Final Integration

8.1. Objective

This milestone integrates power-saving features and performs end-to-end integration testing, including fault handling and recovery. You must:

- Apply power management techniques to reduce average MCU current and document savings.
- Perform full integration tests that demonstrate all major features working together.

8.2. Scope

8.2.1. Part 1: Power Management and Measurement

You must integrate power-saving mechanisms that include:

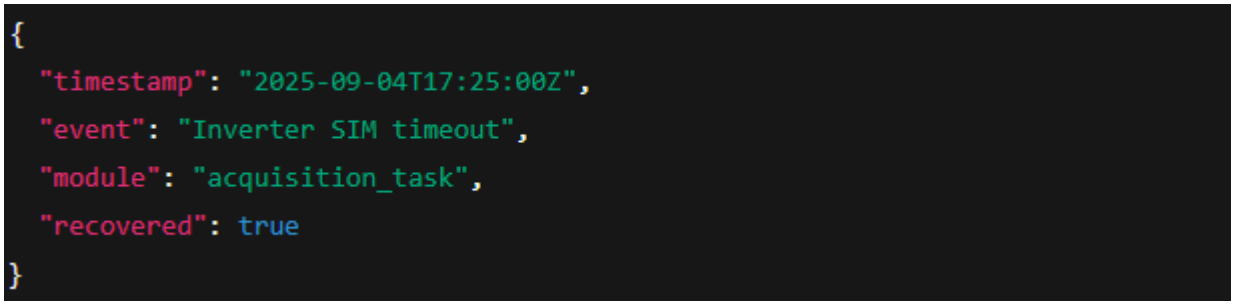
- Clock scaling when idle; sleep between polls/uploads.
- Peripheral gating (disable comm blocks outside windows).
- Optional self-energy-use reporting.
- Prepare a short report comparing the before and after results.

You must apply as many of these mechanisms as are technically feasible, with justification for choices and measurement of resulting savings

8.2.2. Part 2: Fault Recovery

You must implement fault recovery mechanisms that:

- Handle Inverter SIM timeouts, malformed frames, and buffer overflow.
- Maintains a local event log. Sample log shown below:



Faults can be simulated using the provided API tools or pre-crafted payloads (e.g., malformed SIM responses, corrupted JSON, or network timeouts).

8.2.3. Part 3: Final Integration and Fault Testing

You must perform full system integration tests that cover:

- Acquisition, buffering, compression, secure upload, remote configuration, command execution, FOTA, and power management.
- Fault injection tests for network errors, corrupted payloads, and interrupted FOTA, showing recovery behavior.
- You must submit a unified firmware build that integrates all major features implemented in Milestones 1–4 (e.g., config, command, security, FOTA, etc.) and demonstrate their interoperability under typical operating conditions and during controlled fault scenarios (e.g., SIM timeout, network dropout, FOTA hash mismatch)

8.2.4. Part 4: Live Demonstration

- Demonstrate the end-to-end system behavior based on a system integration checklist (provided), verifying each feature was tested under normal and fault conditions.
- This checklist will be marked during the live demonstration

8.2.5. Resources Provided

The following resources will be provided to assist with implementation:

- Milestone 5 - Resources - In21-EN4440-Project Outline
- Updated In21-EN4440-API Service Documentation document

8.2.6. What You Must Submit: Deliverables

- Full source code for the integrated project, including FOTA and power management features.
- Power measurement report summarizing methodology and before and after results.
- Zip all items into a single file for upload to Moodle.

8.3. Evaluation Rubric

Criterion	Weight
Measured power savings and quality of the measurement report	20%
End-to-end system performance and fault recovery correctness	40%
Clarity of video demonstration, including payload measurements	10%
Quality of documentation and test scenarios	5%
Code readability and modularity for porting to the MCU	25%