MANTECH Publications

Journal of Embedded Systems & Its Applications

Volume 5 Issue 1, January-June 2020

Ultra Low Power Firmware Design for Battery-Free IoT Nodes

Gitanjali Sharma[1], Manoj Bairagi[2], Sneha Kulkarni[3], Anurag Jain[4]

Assistant Professor, Associate Professor

Department of Electronics

Vidyodaya Engineering College

gitanjalisharma300@gmail.com[1], manoj4bairagi@yahoo.com[2], Kulkarni_07sneha@rediffmail.com[3]

Abstract

Battery-free Internet of Things (IoT) nodes are emerging as an important solution for sustainable sensing, monitoring, and automation systems. These nodes operate only on harvested energy from environment such as light, vibration, RF signals, and thermal gradients. In such systems, firmware design plays a very critical role because hardware alone cannot guarantee ultra-low power operation. Proper firmware techniques like aggressive sleep scheduling, event-driven execution, energy aware task scheduling, and memory optimization decides whether the node can operate continuously or not. This paper reviews the principles and methods for ultra low power firmware development specifically for battery-free IoT nodes. It discusses energy harvesting models, intermittent computing, state retention, communication optimization, and real time constraints. Tables and figures are provided to explain firmware strategies. This review will help embedded designers to understand how firmware architecture directly impacts energy consumption and reliability of battery-less devices.

Keywords: *Battery-free IoT, Ultra low power firmware, Energy harvesting, Intermittent computing, Sleep scheduling, Embedded systems, Energy aware scheduling.*

1. INTRODUCTION

Internet of Things devices are increasing very fast in number across industries like agriculture, healthcare, smart cities, and environment monitoring. Most of these devices depend on batteries, which creates problems of maintenance, replacement cost, and environmental pollution. Battery-free IoT nodes are developed to overcome this limitation by using harvested energy.

In battery-free systems, the available power is very small and not continuous. The node may run for few milliseconds and then shut down when energy is depleted. Therefore, firmware must be designed in a way that it can operate correctly even under such unstable power supply. This leads to concept of ultra low power firmware and intermittent execution.
Unlike traditional firmware, here developer must think in terms of energy cycles instead of clock cycles.

2. ARCHITECTURE OF BATTERY-FREE IOT NODE

The architecture of a battery-free IoT node is carefully designed so that **every block consumes minimum energy** and can operate with **unstable and very small power levels** coming from environmental sources. Unlike battery powered nodes, here the architecture is tightly coupled with firmware behavior because energy availability directly controls system operation.
A typical battery-free node contains the following main building blocks:

2.1 Energy Source and Energy Harvester

This is the primary block that collects energy from surroundings. Common sources are:

Component	Function	Power Consideration
Energy Harvester	Collect energy from environment	Output is unstable
Energy Storage (Capacitor/Supercap)	Temporary storage	Limited capacity
Ultra-low power MCU	Processing unit	Must support deep sleep
Sensors	Data collection	Should have sleep mode
RF Communication	Data transmission	Most power consuming
Non-volatile Memory	State retention	Needed for intermittent run

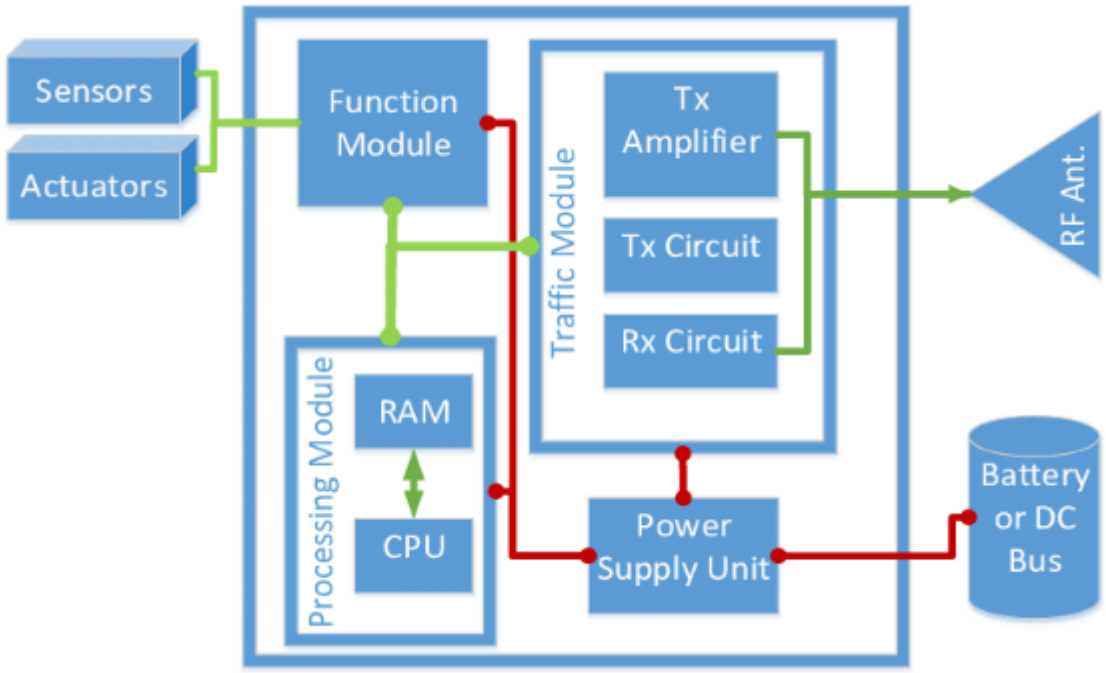

Figure 1: Block diagram of battery-free IoT node

3. CHALLENGES IN FIRMWARE DESIGN FOR BATTERY-FREE IOT NODES

Designing firmware for battery-free IoT nodes is fundamentally different from traditional embedded systems. In conventional systems, developers assume **continuous and stable power**. In contrast, battery-free nodes operate on **tiny, unpredictable, and intermittent energy**, which forces firmware to be written with an **energy-first mindset** rather than a time-first mindset.

The following challenges define how firmware must be architected.

3.1 Unpredictable Power Availability

Energy harvested from light, RF, vibration, or heat is highly variable:

- Sunlight changes with time and weather
- RF energy depends on distance from source
- Vibration depends on machine activity
- Thermal gradients fluctuate

This means the firmware **cannot assume** that enough energy will be present to complete a task.

Implications for firmware:

- Tasks must be broken into **very small executable chunks**
- Firmware must constantly check capacitor voltage using ADC
- Execution decisions must depend on available energy, not on time
- Graceful fallback to sleep when energy is insufficient

Traditional loop-based firmware fails here because it assumes the loop will complete.

3.2 Frequent Power Failures (Brownout Conditions)

A brownout happens when the capacitor voltage drops below MCU operating level during execution. This may occur **in the middle of an instruction**.

Consequences:

- Loss of CPU registers
- Loss of RAM content
- Corrupted program state
- Restart from beginning unless state is saved

Firmware challenges:

- Detect imminent brownout using voltage threshold interrupt
- Perform **checkpointing** before power loss
- Avoid long atomic operations
- Design code to be **restart-safe**

Firmware must be written assuming that **power can fail at any moment**.

3.3 Need for Fast Boot Time

Battery-free nodes may reboot hundreds or thousands of times per day. If boot sequence takes too long:

- Energy gets wasted in initialization
- No useful work is done before shutdown

Firmware design requirements:

- Minimal startup routines
- Skip unnecessary peripheral initialization
- Restore state from non-volatile memory quickly
- Avoid heavy RTOS bootloaders
- Use lightweight main loop or event-driven design

Goal: **Start doing useful sensing within few milliseconds of power up**.

3.4 Memory Retention Across Resets

Because RAM loses data during power loss, firmware cannot rely on volatile memory.

Problems:

- Variables reset to default
- Program loses track of last operation
- Data inconsistency after restart

Required solutions:

- Use FRAM/EEPROM to store critical variables
- Periodically store program counter and state
- Use non-volatile flags to indicate last completed task
- Implement idempotent code (safe to run multiple times)

This leads to concept of **non-volatile computing**.

3.5 Minimizing Communication Overhead

Wireless communication consumes **10–100× more energy** than sensing or computation. If firmware transmits data frequently:

- Capacitor drains immediately
- Node shuts down before useful operation

Firmware strategies needed:

- Data compression before transmission
- Send data in batches
- Transmit only when energy level is high
- Use acknowledgment-free protocols where possible
- Reduce packet size and headers

Firmware must treat communication as a **rare and precious operation**.

3.6 Accurate Task Scheduling Based on Energy

Traditional systems schedule tasks using timers, RTOS ticks, or deadlines. Battery-free nodes cannot follow strict time schedules.

Instead, tasks must be scheduled like:

- If energy high → Sense + Process + Transmit
- If energy medium → Sense + Store
- If energy low → Sleep

Challenges:

- Mapping capacitor voltage to energy budget
- Predicting if task can finish before energy runs out
- Prioritizing tasks dynamically
- Avoiding partial task execution

This requires an **energy-aware scheduler** inside firmware.

4. SLEEP SCHEDULING AND DUTY CYCLING

Sleep mode is the most important technique for reducing power consumption.

Mode	Typical Current	Usage
Active mode	3–8 mA	Processing
Idle mode	500 µA	Waiting for event
Deep sleep	<5 µA	Most of time
Shutdown	<1 µA	Energy critical

Firmware should keep MCU in deep sleep almost 99% of time and wake only on interrupts from sensors or timers.

Key Techniques:

- Use interrupt driven design instead of polling
- Disable unused peripherals
- Use low frequency clocks during sleep
- Wake up only when energy threshold is reached

5. INTERMITTENT COMPUTING AND CHECKPOINTING

Because energy is not stable, the MCU may lose power anytime. Firmware must save execution state periodically.

Checkpointing Process:

1. Save CPU registers

2. Store program counter
3. Save important variables to non-volatile memory
4. Resume execution after power returns

Step	Action	Memory Used
State save	Registers + stack	FRAM/EEPROM
Power loss	System off	—
Restart	Restore state	FRAM/EEPROM

This allows device to continue from where it stopped.

6. ENERGY AWARE TASK SCHEDULING

Firmware should schedule tasks based on available energy rather than time.

Example:

- If capacitor voltage > threshold → Perform sensing + transmission
- If medium energy → Only sensing
- If low energy → Go to sleep

This is called **energy driven execution model**.

7. MEMORY OPTIMIZATION TECHNIQUES

Memory access consumes power. Therefore:

- Use static memory instead of dynamic allocation
- Reduce stack size
- Use bit-fields and packed structures
- Avoid large buffers
- Use DMA where possible

Efficient memory usage reduces both execution time and power.

8. COMMUNICATION OPTIMIZATION

RF transmission consumes more energy than processing.

Operation	Energy Cost
Sensor read	Low
Data processing	Medium

Operation	Energy Cost
RF transmit	Very high

Firmware should:

- Compress data before sending
- Send data in bursts
- Reduce packet size
- Use low power protocols like BLE, LoRa, Zigbee
- Avoid frequent transmission

9. FAST BOOT AND LIGHTWEIGHT FIRMWARE

Since node may reboot many times, boot time must be minimal.

- Remove unnecessary initialization
- Use lightweight RTOS or no OS
- Store configuration in non-volatile memory
- Avoid long startup routines

10. EVENT DRIVEN PROGRAMMING MODEL

Polling wastes energy. Event driven design is required.

Events can be:

- Sensor interrupt
- Voltage threshold interrupt
- Timer interrupt
- RF receive interrupt

This approach ensures MCU wakes only when required.

11. USE OF NON-VOLATILE PROCESSORS (FRAM BASED MCUS)

FRAM based microcontrollers are very useful because they retain memory without power.

Advantages:

- Very fast write speed
- Low write energy

- High endurance
- Suitable for checkpointing

12. CASE STUDY: SOLAR POWERED ENVIRONMENTAL SENSOR NODE

Parameter	Value
Energy source	Small solar cell
Storage	100 mF capacitor
MCU	Ultra low power MSP430
Sensor	Temperature + Humidity
Communication	LoRa

Firmware behavior:

1. Wait till capacitor reaches 3V
2. Wake, read sensors
3. Store data
4. Transmit packet
5. Sleep again

This cycle repeats based on sunlight availability.

13. Firmware Design Flow

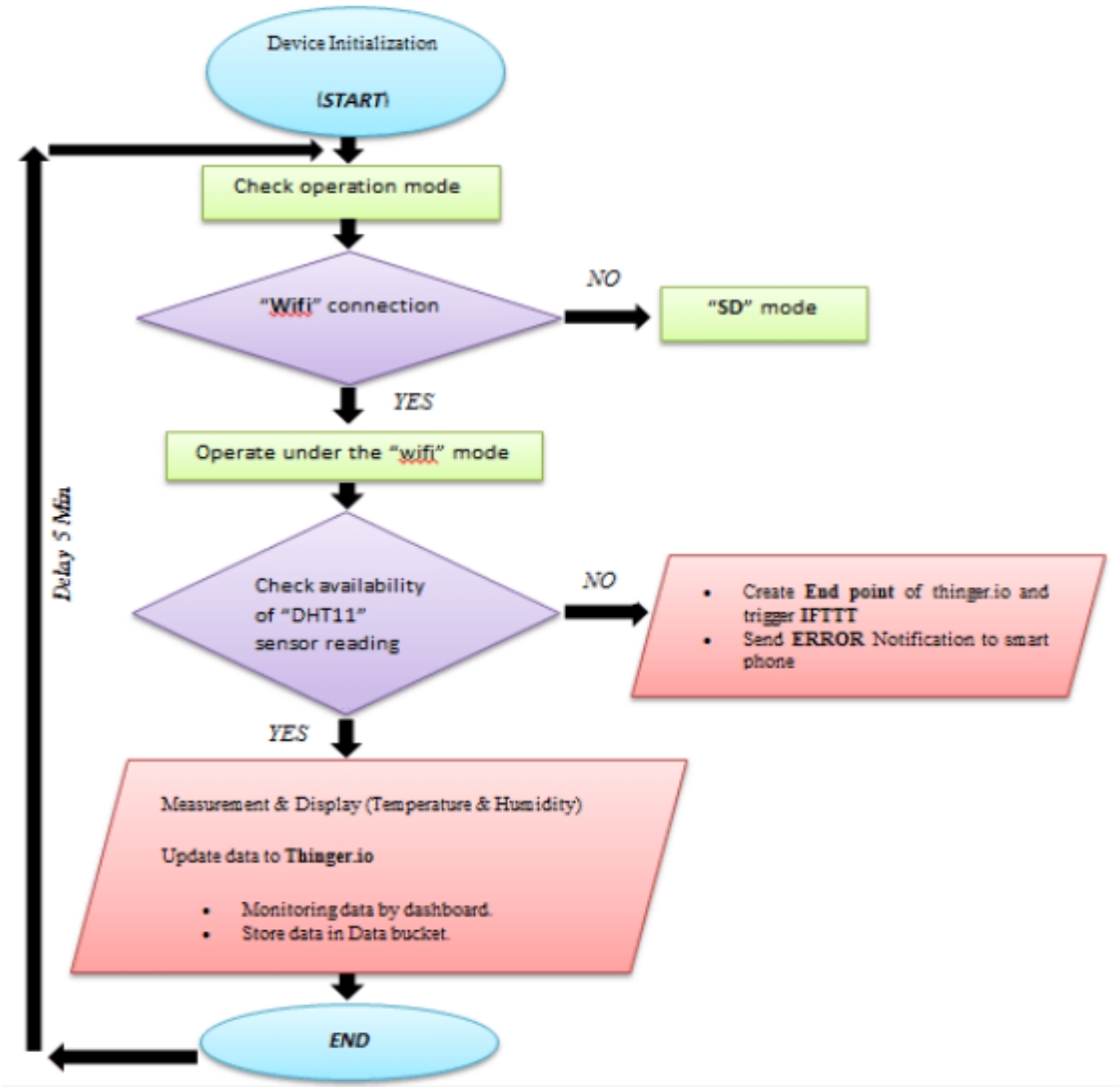

Figure 2: Firmware flow for battery-free node

14. Comparison with Traditional Firmware

Feature	Traditional IoT	Battery-Free IoT
Power source	Battery	Harvested energy
Execution	Continuous	Intermittent
Scheduling	Time based	Energy based
Reset handling	Rare	Frequent
Memory usage	Flexible	Highly optimized

15. FUTURE TRENDS

- AI based energy prediction
- Ultra low power RISC-V cores
- Non-volatile processors
- Self-adaptive firmware
- Standard firmware frameworks for battery-less systems

CONCLUSION

Ultra low power firmware design is the backbone of battery-free IoT nodes. Without proper firmware strategies, even best hardware cannot sustain operation on harvested energy. Techniques like deep sleep scheduling, intermittent computing, checkpointing, memory optimization, and energy aware scheduling are necessary. As energy harvesting technology improves, firmware approaches will also evolve. Designers must think in terms of energy availability rather than continuous power. Battery-free IoT will play very big role in future smart environments and sustainable sensing networks.

REFERENCES

1. A. Kansal, J. Hsu, “Power Management in Energy Harvesting Sensor Networks,” IEEE, 2018.
2. B. Lucia, K. Maeng, “Intermittent Computing: Challenges and Opportunities,” ACM, 2017.
3. S. Ransford et al., “Mementos: System Support for Long-Running Computation,” ASPLOS, 2019.
4. Texas Instruments, “Ultra Low Power MSP430 Microcontroller Guide,” 2020.
5. P. Dutta, D. Culler, “Energy Aware Wireless Sensor Platforms,” IEEE Sensors, 2016.
6. J. Hester, J. Sorber, “The Future of Battery-Free Computing,” IEEE Pervasive Computing, 2019.
7. ARM Ltd., “Low Power Firmware Techniques for IoT,” Whitepaper, 2021.
8. L. Yang, “Energy Harvesting for Embedded Systems,” Springer, 2018.
9. Microchip Technology, “Low Power Design Tips for Embedded Firmware,” 2020.
10. S. Sudevalayam, P. Kulkarni, “Energy Harvesting Sensor Nodes: Survey,” Ad Hoc Networks Journal, 2017.