**Energy Saving Algorithm Implementation Based on Dynamic Clock Adjustment in Embedded Systems**  
**Cheng-Yi Chiu, Chi-En Ting**

**Abstract**

In most embedded systems, the clock frequency is set statically at task startup, which limits energy optimization under varying workloads. This work presents a method to dynamically adjust the clock frequency based on task characteristics to minimize energy consumption while maintaining performance. We implement a profiling-assisted frequency selection algorithm on the STM32L4 platform and classify tasks into compute-bound, I/O-bound, and unknown types. Experimental results show that our method can effectively select energy-efficient frequencies for compute-bound workloads, achieving significant energy-delay product (EDP) improvements.

**I. Introduction**

Power efficiency has become a critical concern in embedded system design. Traditionally, system clock frequency is fixed during task execution, leading to unnecessary energy consumption, especially when task behavior varies significantly over time. Existing dynamic voltage and frequency scaling (DVFS) methods often consider frequency and voltage together, but relatively few studies isolate the impact of frequency alone. Our goal is to implement a dynamic frequency adjustment algorithm that minimizes energy consumption by tailoring clock frequency to each task's runtime characteristics.

**II. Background and Motivation**

In real-world systems, tasks exhibit different phases—some are memory-intensive (I/O-bound), while others are computation-intensive (compute-bound). A fixed clock frequency cannot efficiently serve all types of tasks. Inspired by Power Clocks [1], a dynamic multi-clock management framework, and the Profiling-Assisted Decoupled Access-Execute (PDAE) model [2], we explore a software-level mechanism that dynamically selects the most energy-efficient frequency for each task phase.

We employ the Optimal Energy-Delay Product (OptEDP) metric to evaluate task performance under different frequencies. The selected frequency minimizes the EDP value, representing the trade-off between energy consumption and execution delay.

**III. Challenges in Implementation**

Our implementation is based on the STM32L4 microcontroller, which is inherently optimized for low power consumption. However, several practical challenges arise:

* **Lack of power estimation tools**: Unlike x86 platforms, STM32L4 lacks APIs like PAPI to retrieve Instructions Per Cycle (IPC), hindering accurate power modeling.
* **Difficulty separating static and dynamic power**: We rely on datasheet-based lookup tables to estimate energy use.
* **Inherent low power design**: Due to STM32L4's already optimized architecture, the energy savings from frequency tuning are harder to measure.

To address these issues, we designed large-scale tasks to amplify energy differences and simplified experimental procedures to focus on observing energy trends.

**IV. System Design**

**A. Task Types**

We categorize tasks into the following:

* **Compute-bound**: Dominated by arithmetic operations (e.g., repetitive additions and subtractions).
* **I/O-bound**: Dominated by serial output (e.g., printing messages).
* **Unknown**: Behavior undetermined before runtime.

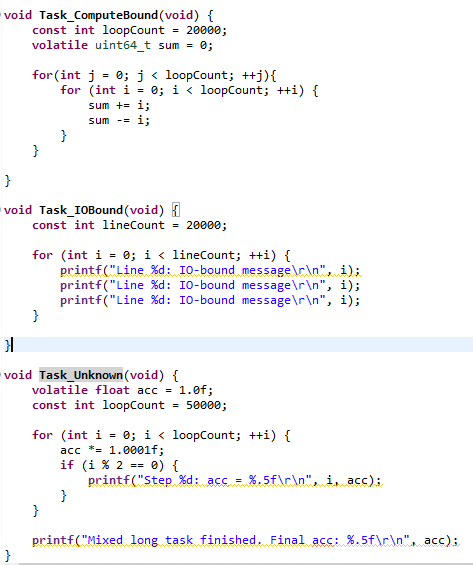


Fig 1. Code for task design

**B. Clock Configuration**

We utilize a Multi-Speed Internal oscillator (MSI) in combination with PLL to generate various clock frequencies. The system clock is computed as:

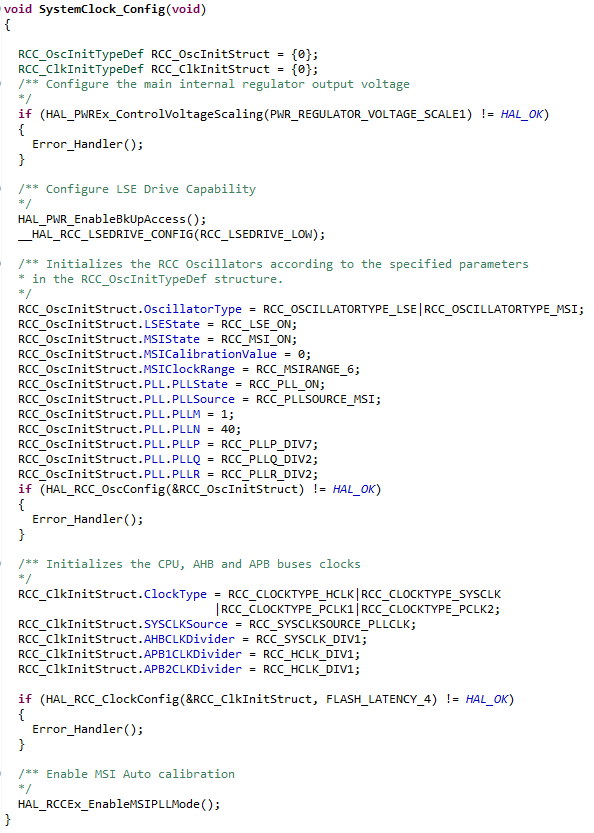
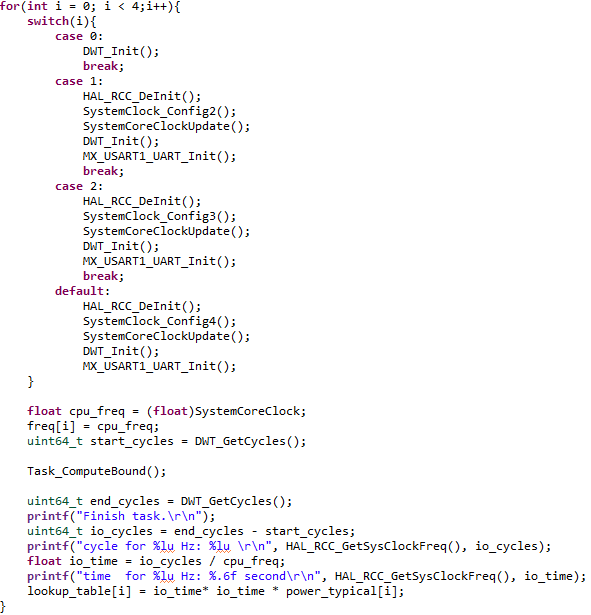


Fig 2. Code for clock structure

**C. Frequency Selection Algorithm**

For each task:

1. Execute at all candidate frequencies.
2. Record execution time and compute estimated energy consumption from the lookup table.
3. Calculate EDP = Energy × Delay.
4. Select the frequency with the minimum EDP for future execution.



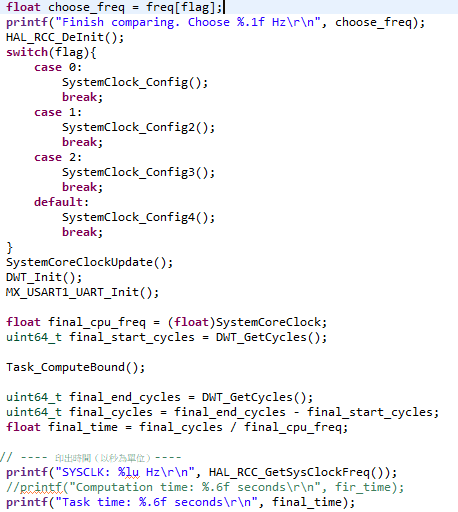


Fig 3. Code implementation for algorithm

**V. Experimental Results**

**A. Execution Time vs. Frequency**

Using a log-scale plot, we observe a clear inflection point:

* Below a threshold, the task is I/O-bound (frequency has little effect on delay).
* Above the threshold, the task is compute-bound (frequency significantly affects delay).

This threshold varies by task type.

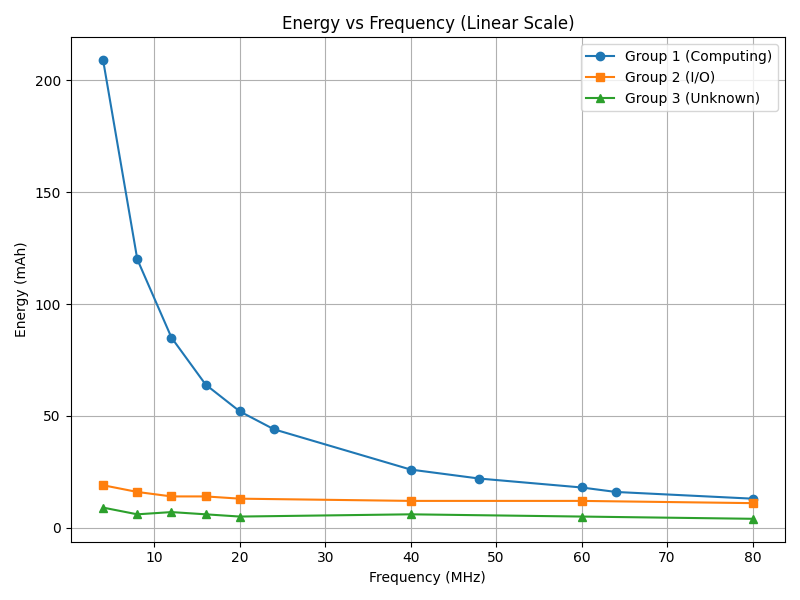
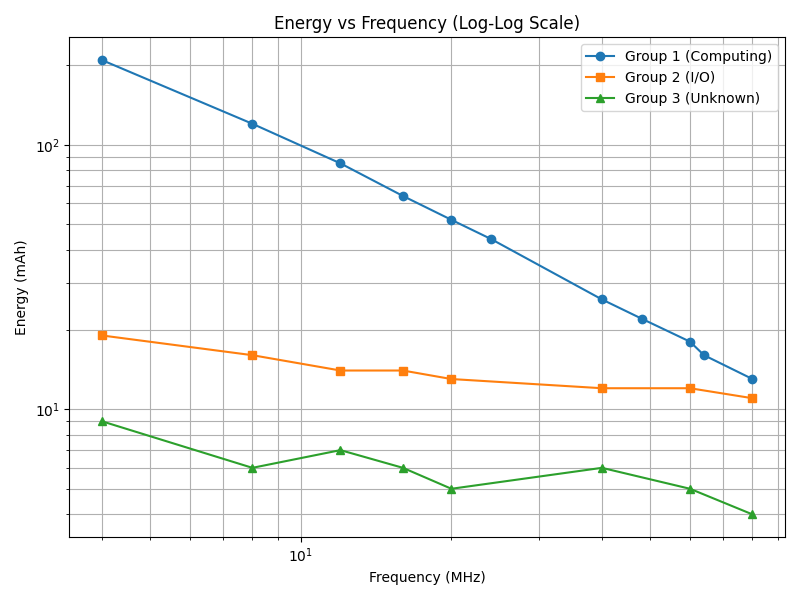


Fig 4. Energy-frequency relation. Left for log scale, and right for linear scale.

**B. Energy Consumption**

Measured energy data also shows log-linear behavior with respect to frequency. Our algorithm successfully selects the lowest-energy frequency for compute-bound tasks, validating its practical benefit.

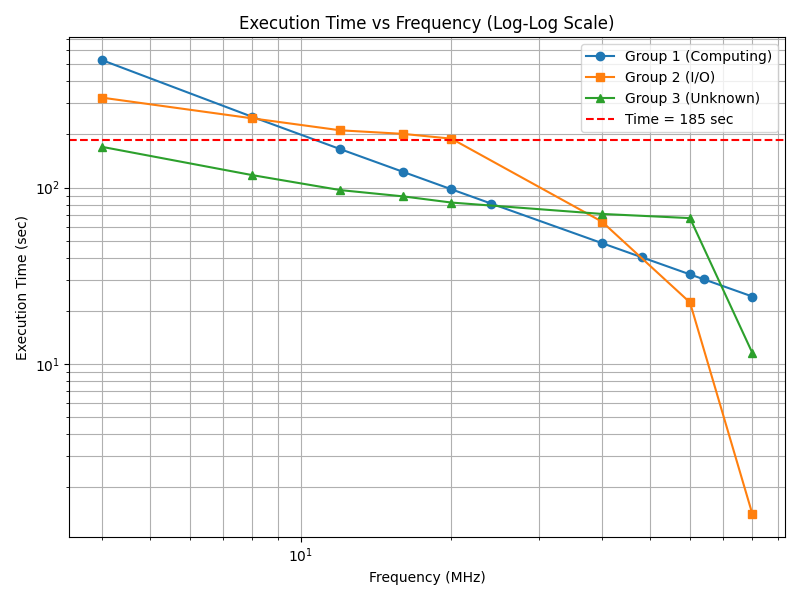
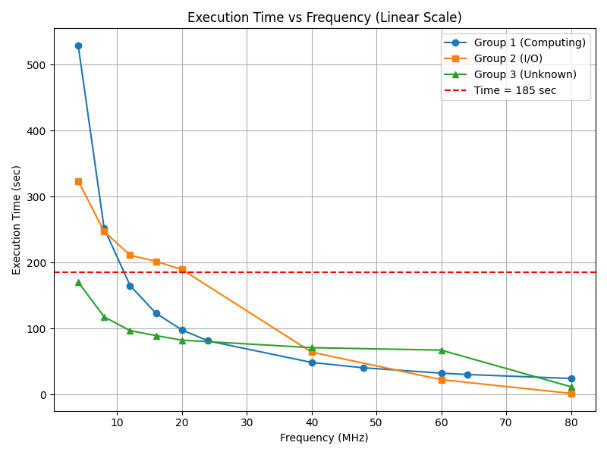


Fig 5. Energy- execution time relation. Left for linear scale, and right for log scale.

**VI. Conclusion and Future Work**

We implemented a dynamic frequency tuning algorithm based on the OptEDP framework on an STM32L4 MCU. Despite measurement limitations, the method effectively selects energy-efficient frequencies for compute-heavy tasks. Overall, these results confirm that Compute-bound workloads benefit from almost linear speedup as frequency increases; I/O-bound workloads see minimal gains until the clock surpasses the I/O latency threshold; and Unknown workloads transition from I/O-limited to compute-limited as frequency rises. This insight validates our dynamic frequency tuning strategy: by profiling each task’s behavior at multiple clock rates and choosing the frequency that minimizes the Energy‐Delay Product (EDP), we can run compute-heavy code at higher clocks while keeping I/O-bound code at just above its bottleneck frequency—thereby meeting performance requirements without wasting power.

Future work includes:

* Extending to full DVFS with voltage control.
* Applying machine learning to automate task classification and frequency prediction.

**References**

[1] J. Waern, P. Ekemark, K. Koukos, S. Kaxiras, A. Jimborean, "Power Clocks: Dynamic Multi-Clock Management for Embedded Systems", *IEEE Trans. on Embedded Systems*, 2020.

[2] J. Waern et al., "Profiling-Assisted Decoupled Access-Execute", *Proc. of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

**Appendix**

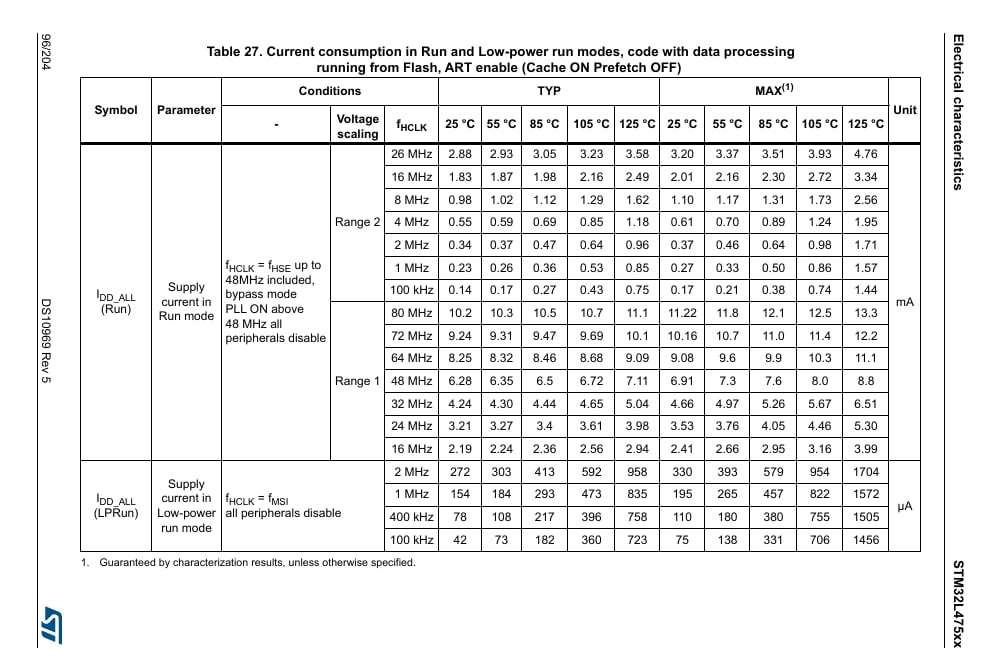


Table 1. current value for various temperature and frequency for stm32l4 series