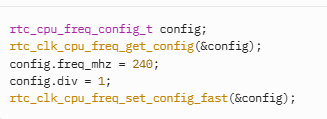
**Các Kỹ Thuật Tăng Thông Lượng**

Dưới đây là các phương pháp tối ưu hóa thông lượng được sử dụng trong code, với phân tích chi tiết về cách chúng hoạt động và hậu quả nếu không áp dụng.

**1. Ép Xung CPU (Overclocking to 240 MHz)**

* **Cách thức**:
  + Code thiết lập tần số CPU của ESP32 lên 240 MHz (tối đa) trong hàm *setup()* bằng cách sử dụng *rtc\_clk\_cpu\_freq\_set\_config\_fast():*



* ASCON-128 thực hiện các phép toán nặng (12 round cho initialization/finalization, 6 round cho associated data/plaintext) trên trạng thái 320-bit (5 word 64-bit). Mỗi round bao gồm các phép XOR, AND, NOT, và rotation, yêu cầu nhiều chu kỳ CPU.
* Tăng tần số từ 160 MHz lên 240 MHz giảm thời gian thực thi mỗi instruction khoảng 33% (1/240 MHz so với 1/160 MHz). Ví dụ, nếu một phép XOR mất 1 chu kỳ, thì tại 240 MHz, thời gian thực thi giảm từ 6.25 ns xuống 4.17 ns.
* Thông lượng được tính là :

Đồng nghĩa với việc kích thước gói tin và thông lượng tỷ lệ thuận với nhau.

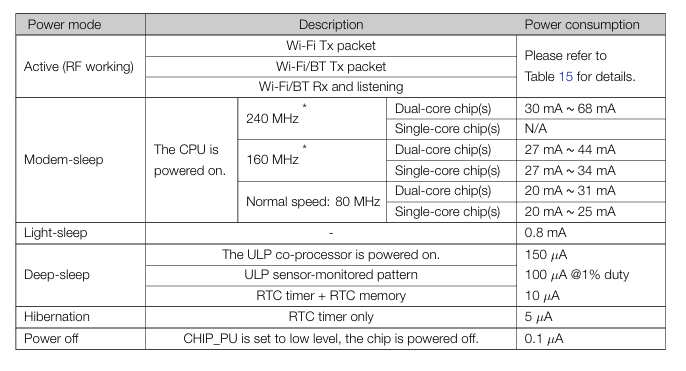
Trong IOT kích thước của dữ liệu thực chỉ rơi vào vài chục Byte nên để thông lượng mà ít ảnh hưởng đến phần cứng nhất chỉ có thể là padding.

**2. Tắt WiFi và Bluetooth**

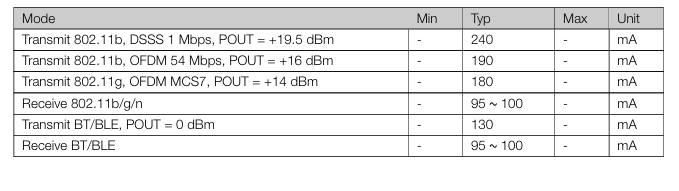
* **Cách thức**:
  + Code tắt WiFi và Bluetooth để giải phóng tài nguyên CPU và giảm gián đoạn



* Khi Wi-Fi/BT chạy, CPU còn phải xử lý ngắt (interrupts) và ngăn xếp giao thức mạng.Vì vậy nó mất 1 phần nhỏ của chu kì để xử lí những tác vụ liên quan đến Wi-Fi/BT.
* Ngoài ra khi bật Wi-Fi/BT đồng nghĩa với việc CPU ở chế độ Active (RF working) cần dòng tiêu thụ cao.Điều này CPU tăng nhiệt nhanh hơn mức bình thường.Giảm hiệu suất cũng như độ chính xác của CPU khi mã hóa



Bảng Năng lượng tiêu thụ theo chế độ(ESP Datasheet- Espressif Systems )



Bảng tiêu thụ năng lượng ở chế độ RF(ESP Datasheet- Espressif Systems)

**3. Sử Dụng Bộ Nhớ Nhanh (IRAM và heap\_caps\_malloc)**

* **Cách thức**:
  + Code cấp phát bộ nhớ cho *message, ciphertext, decrypted* bằng *heap\_caps\_malloc()* với *flag MALLOC\_CAP\_8BIT | MALLOC\_CAP\_INTERNAL*, đảm bảo lưu trữ trong IRAM (RAM nội bộ nhanh của ESP32). Hàm *benchmark\_core()* cũng được đánh dấu *IRAM\_ATTR* để chạy từ IRAM:



* + IRAM có độ trễ thấp hơn PSRAM (external RAM), và code trong IRAM chạy trực tiếp từ RAM thay vì flash (chậm hơn).
  + ASCON-128 truy cập bộ nhớ thường xuyên trong hàm LOAD/STORE (đọc/ghi t rạng thái 320-bit) và permutation. IRAM có latency ~1-2 cycle so với ~10-20 cycle của PSRAM, giảm thời gian truy cập dữ liệu.
  + Nếu truy cập PSRAM, mỗi byte mất thêm ~10 cycle, tổng cộng ~120 cycle cho 12 byte. Với IRAM, chỉ mất ~12-16 cycle, tăng tốc ~5x.
  + Chạy code từ IRAM giảm cache misses, tăng thông lượng ~2-3x so với flash, vì flash có độ trễ cao hơn (đặc biệt với SPI flash).

**4. Sử Dụng Đa Lõi (Dual-Core Parallelism)**

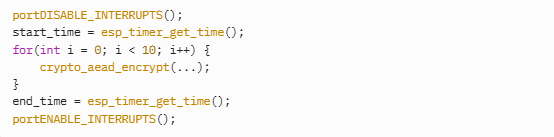
* **Cách thức**:
  + Code tạo hai task chạy *benchmark\_core()* trên core 0 và core 1 của ESP32, sử dụng *xTaskCreatePinnedToCore():*



* ESP32 có hai lõi Xtensa LX6, mỗi lõi xử lý độc lập. Chạy mã hóa/giải mã song song trên hai lõi gần như tăng gấp đôi thông lượng tổng (total\_enc ≈ core0 + core1).
* Ví dụ, nếu core 0 đạt 1 MB/s và core 1 đạt 1 MB/s, tổng thông lượng là ~2 MB/s, so với ~1 MB/s nếu chỉ dùng một core.
* Mutex (portENTER\_CRITICAL) đảm bảo đồng bộ core\_stats mà không gây bottleneck (do chỉ update vài giá trị nhỏ).

**5. Disable Interrupts Khi Đo Thời Gian**

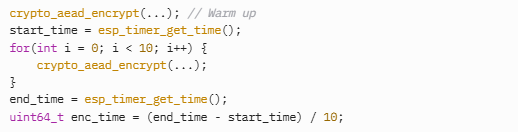
* **Cách thức**:
  + Code tắt interrupts trong giai đoạn đo thời gian để đảm bảo độ chính xác:



* Interrupts (timer, serial, system) làm gián đoạn CPU, tăng thời gian thực thi. Ví dụ, một interrupt mất 10 µs, với 100 interrupts/s, tổng thời gian mất là 1000 µs/s.
* Tắt interrupts đảm bảo CPU chạy liên tục trong vòng lặp mã hóa/giải mã, giảm thời gian đo, tăng thông lượng báo cáo chính xác hơn.

**6. Warm Up Cache và Multiple Iterations**

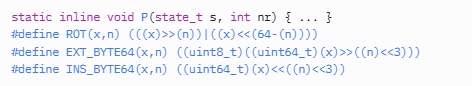
* **Cách thức**:
  + Code chạy một lần encrypt/decrypt trước benchmark để warm up cache, rồi đo trung bình qua 10 lần lặp:



* Lần đầu chạy ASCON, cache cold (misses cao) làm tăng thời gian do truy cập bộ nhớ chậm. Warm up đảm bảo trạng thái, key, nonce đã nằm trong cache L1 (~8 KB/core trên ESP32).
* Lặp 10 lần và lấy trung bình làm mịn noise từ hệ thống (jitter, interrupts nhỏ), đảm bảo thông lượng đo chính xác.
* Ví dụ, nếu lần đầu mất 1000 µs do cache miss, nhưng lần sau chỉ 600 µs, thì warm up và average giúp báo cáo đúng ~16.7 MB/s thay vì 10 MB/s.

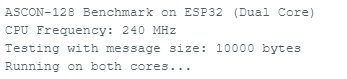
**7. Tối Ưu Hóa Code ASCON (Inline Functions và Macros)**

* **Cách thức**:
  + Hàm *permutation (P), LOAD, STOR*E được đánh dấu static inline để tránh *overhead* gọi hàm. Các phép rotation sử dụng *macro (ROT, EXT\_BYTE64, INS\_BYTE64):*



* Inline thay thế function call bằng code trực tiếp, giảm overhead (~10-20 cycle/call). Với hàng trăm lần gọi P/LOAD trong mã hóa, điều này tiết kiệm đáng kể.
* Macro ROT thực hiện rotation bit ngay trong code, tránh hàm phụ, giảm ~2-5 cycle mỗi rotation. ASCON có 10 rotation/round, với 12/6 round, tiết kiệm ~120-300 cycle/khối.
* Ví dụ, nếu mỗi round mất 100 cycle, inline/macros giảm xuống ~80 cycle, tăng thông lượng ~20% (từ 15 MB/s lên 18 MB/s).

**Kết quả khi chạy thử nhiệm (Được CPU trả về thông qua Serial Monitor):**

****

Thuật toán ASCON-128 được chạy trên 2 core

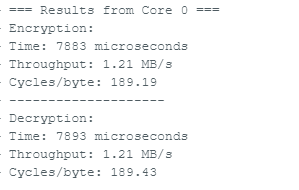
Tần số CPU:240 MHz

Kích thước dữ liệu được mã hóa: 10KB(12 Byte dữ liệu thật).

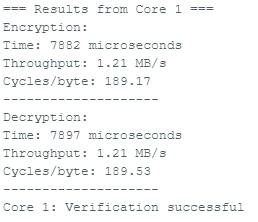
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Padding Data  (9988 Byte) | LDR Data  (1 Byte) | MOVE  Data  (1 Byte) | DHT- Humidity Data  (1 Byte) | DHT- Temperature  Data  (1 Byte) | Latitude  Data  (4 Byte) | Longitude  Data  (4 Byte) |

Thông lượng trên từng core:

* Core 0:



* Core 1:

****

* Thông lượng tổng:

