Tự xây dựng một tập lệnh AT trên vi điều khiển

/\*-----------------------------------------------------\*/

Trong video này, mình sẽ chạy demo việc xây dựng tập lệnh AT

trên vi điều khiển qua giao tiếp UART để đọc, cài đặt thời gian, đọc nhiệt độ và điều khiển đèn led.

Các phần mềm mình sử dụng gồm:

- Proteus 8.6 để mô phỏng mạch

- Microchip studio để lập trình cho vi điều khiển

- Termite 3.4 để giao tiếp với vi điều khiển qua UART

- Virtual Serial Port Driver để tạo cổng COM ảo.

Vi điều khiển mình sử dụng là atmega328P, các ngoại vi dùng gồm:

- UART - nhân vật chính, trong đó có dùng thêm 2 thư viện phụ là

+ Ringbuffer: giúp quản lý dữ liệu tiện lợi hơn

+ CLI (command line interface): quản lý các lệnh

- Timer 1: tạo ngắt 1ms để chạy bộ lập lịch

- ADC: đọc cảm biến nhiệt độ lm35

- GPIO: để điều khiển led

Cấu trúc lệnh AT (Mình dựa vào tập lệnh AT của các chip quetel)

- AT+<cmd> : lệnh thực thi

- AT+<cmd>=? : lệnh trả về các tham số của lệnh

- AT+<cmd>=<...> : lệnh ghi

- AT+<cmd>? : lệnh đọc

Các bạn có thể tải project mô phỏng và file hex tại đây

[https://github.com/QuangPhuo.../share/tree/main/cli\_template](https://github.com/QuangPhuoc1998/share/tree/main/cli_template?fbclid=IwAR1Eo019jhS9ES_oIG8v0WJ-ye7zlDIoppg6ujFJozYQT-AyHNr20fEc29A)

𝐓𝐢̀𝐦 𝐡𝐢𝐞̂̉𝐮 𝐛𝐨𝐨𝐭𝐥𝐨𝐚𝐝𝐞𝐫 𝐭𝐫𝐞̂𝐧 𝐯𝐢 đ𝐢𝐞̂̀𝐮 𝐤𝐡𝐢𝐞̂̉𝐧 - 𝐏𝐡𝐚̂̀𝐧 𝟏

Chào các bạn, trong chuỗi bài viết này chúng ta sẽ tìm hiểu về ứng dụng của bootloader và cách triển khai bootloader trong 1 số tình huống cụ thể

✔𝟏. 𝐏𝐡𝐮̛𝐨̛𝐧𝐠 𝐩𝐡𝐚́𝐩 𝐜𝐚̣̂𝐩 𝐧𝐡𝐚̣̂𝐭 𝐟𝐢𝐫𝐦𝐰𝐚𝐫𝐞 𝐜𝐡𝐨 𝐯𝐢 đ𝐢𝐞̂̀𝐮 𝐤𝐡𝐢𝐞̂̉𝐧

Hiện nay có 3 cách chủ yếu để cập nhật firmware cho vi điều khiển:

- Sử dụng debugger chuyên dụng như St-linh cho stm32, Jlink, Ulink cho các dòng ARM, Pickit cho pic

+ Ưu điểm: support tận răng, nạp và debug thoải mái

+ Nhược điểm: cồng kền, giá thành cao (Jlink, Ulink hàng real cũng 15 củ khoai)

- Sử dụng chức năng boot của vi điều khiển hay bootloader mặc định (Điển hình như esp32)

+ Ưu điểm: nhỏ gọn, nạp qua UART, I2C, SPI tùy phiên bản, chỉ cần 1 mạch USB-UART là có thể update firmware được

+ Nhược điểm: Chức năng hạn chế trong giới hạn của bootloader mặc định, thường chỉ debug bằng log, khó tùy biến

- Sử dụng bootloader hay second bootloader

+ Ưu điểm: nhỏ gọn, tùy biến cao tùy thuộc theo nhu cầu

+ Nhược điểm: thương chỉ debug bằng log

✔𝟐. 𝐕𝐢𝐞̣̂𝐜 𝐠𝐢̀ 𝐬𝐞̃ 𝐱𝐚̉𝐲 𝐫𝐚 𝐤𝐡𝐢 𝐯𝐢 đ𝐢𝐞̂̀𝐮 𝐤𝐡𝐢𝐞̂̉𝐧 𝐤𝐡𝐨̛̉𝐢 đ𝐨̣̂𝐧𝐠

- Khi đặt vi điền khiển khỏi động , bộ đếm chương trình hoặc program counter được tải với địa chỉ

0x00000000 (địa chỉ đầu tiên này chứa địa chỉ của đỉnh ngăn xếp hoặc giá trị sẽ được tải vào con trỏ ngăn xếp chính.)

- Tải giá trị được lưu tại địa chỉ 0x00000000 vào MSP (Main stack pointer)

- PC di chuyển đến địa chỉ tiếp theo 0x00000004 và giá trị tại địa chỉ 0x00000004 chứa một con trỏ để trỏ tới hàm reset handler

- Chạy chức năng của hàm reset handler (Vô hiệu hóa tất cả ngắt, khởi tạo đoạn dữ liệu, khởi tạo phân đoạn .Bss, khởi tạo

phân đoạn ngăn xếp, kích hoạt ngắt,…)

- Gọi hàm main()

(Các bạn có thể phân tích file start-up.s để biết thêm chi tiết)

✔𝟑. 𝐂𝐡𝐮̛́𝐜 𝐧𝐚̆𝐧𝐠 𝐜𝐮̉𝐚 𝟏 𝐛𝐨𝐨𝐭𝐥𝐨𝐚𝐝𝐞𝐫

- Bootloader là một chương trình cho phép chúng ta tải các chương trình khác thông qua giao diện thuận tiện hơn như cáp USB.

Khi bật nguồn hoặc reset bo mạch vi điều khiển, bootloader sẽ kiểm tra xem có yêu cầu cập nhật firmware (ví dụ như nhấn nút) hay không.

Nếu có, nó sẽ tải lên chương trình mới và ghi nó vào bộ nhớ Flash.

Nếu không, nó sẽ bắt đầu chạy chương trình cuối cùng mà bạn đã tải vào trước đó (Chương trình Application).

- Bootloader thương được đặt tại ví trị bắt đầu 0x00000000 (Với stm32 là 0x08000000).

✔𝟒. 𝐇𝐚̀𝐦 𝐧𝐡𝐚̉𝐲 𝐭𝐨̛́𝐢 𝐜𝐡𝐮̛𝐨̛𝐧𝐠 𝐭𝐫𝐢̀𝐧𝐡 𝐜𝐡𝐢́𝐧𝐡

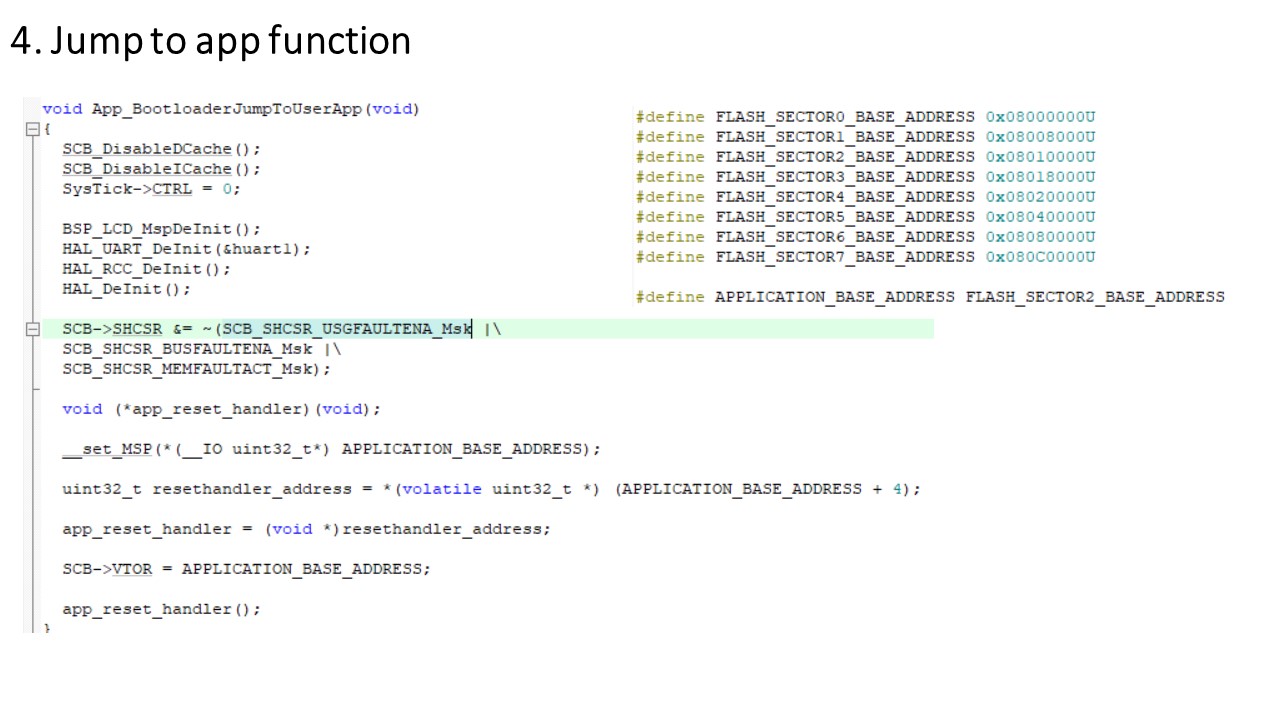
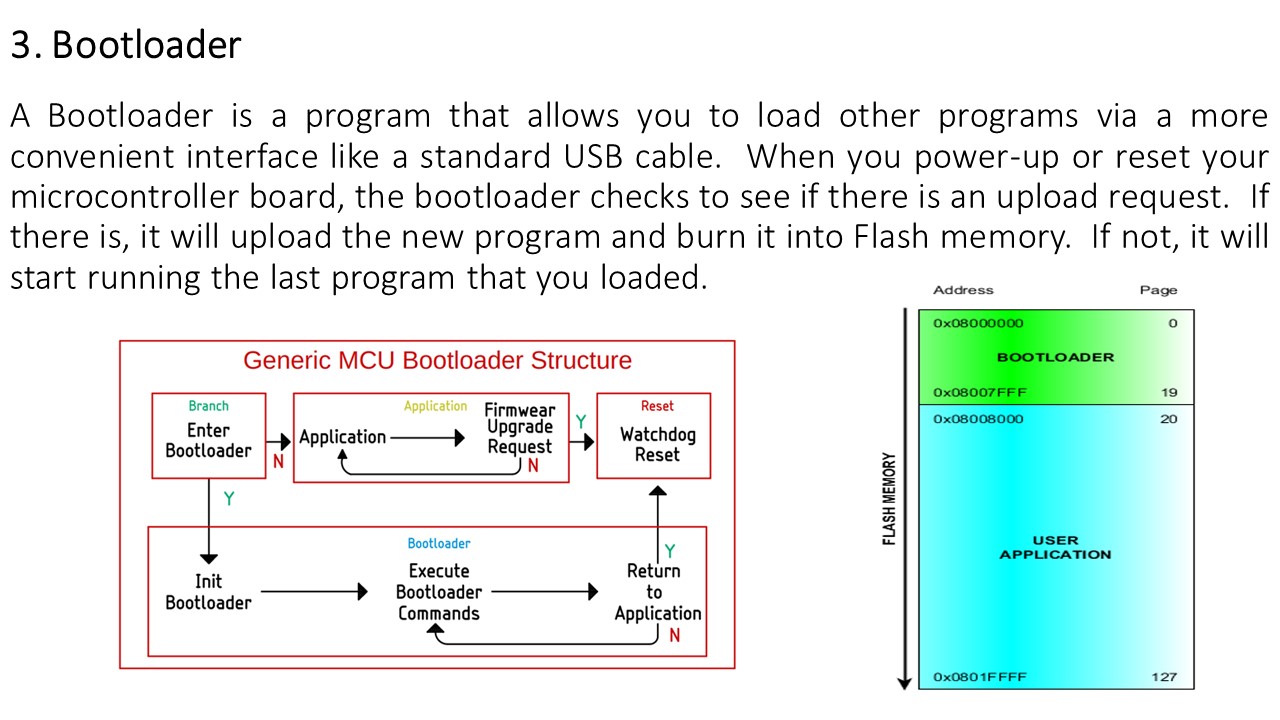
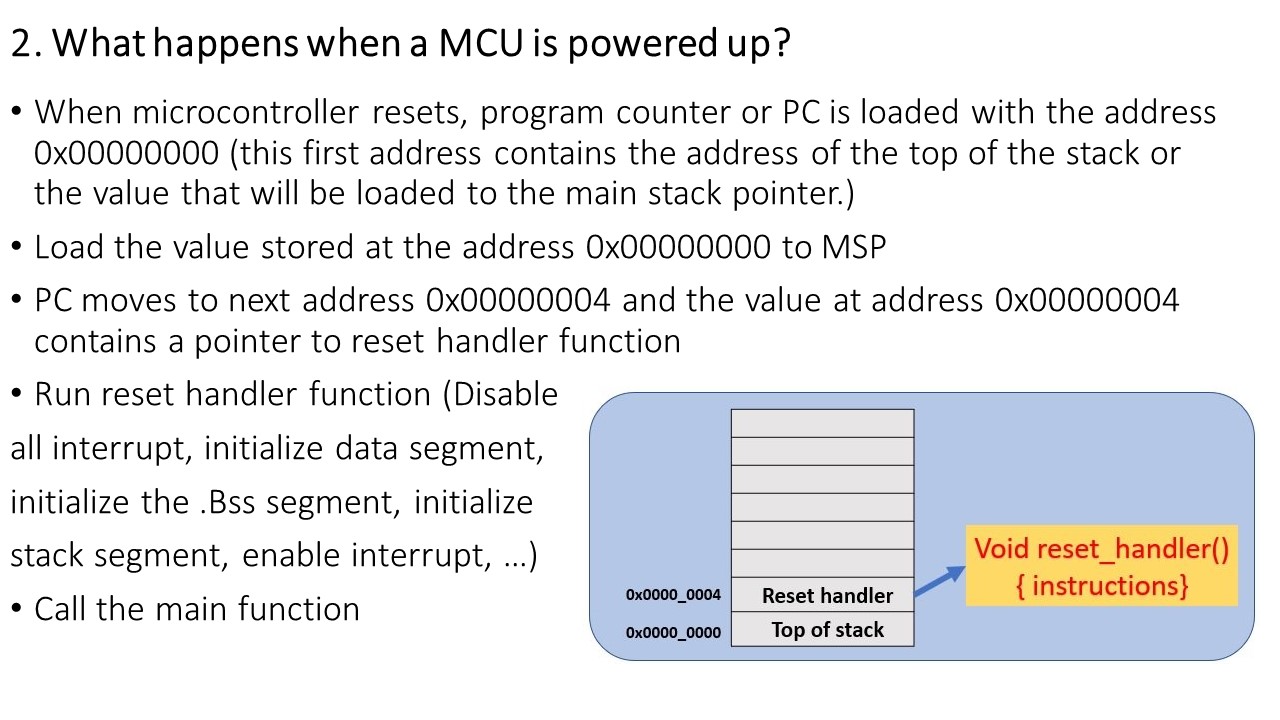
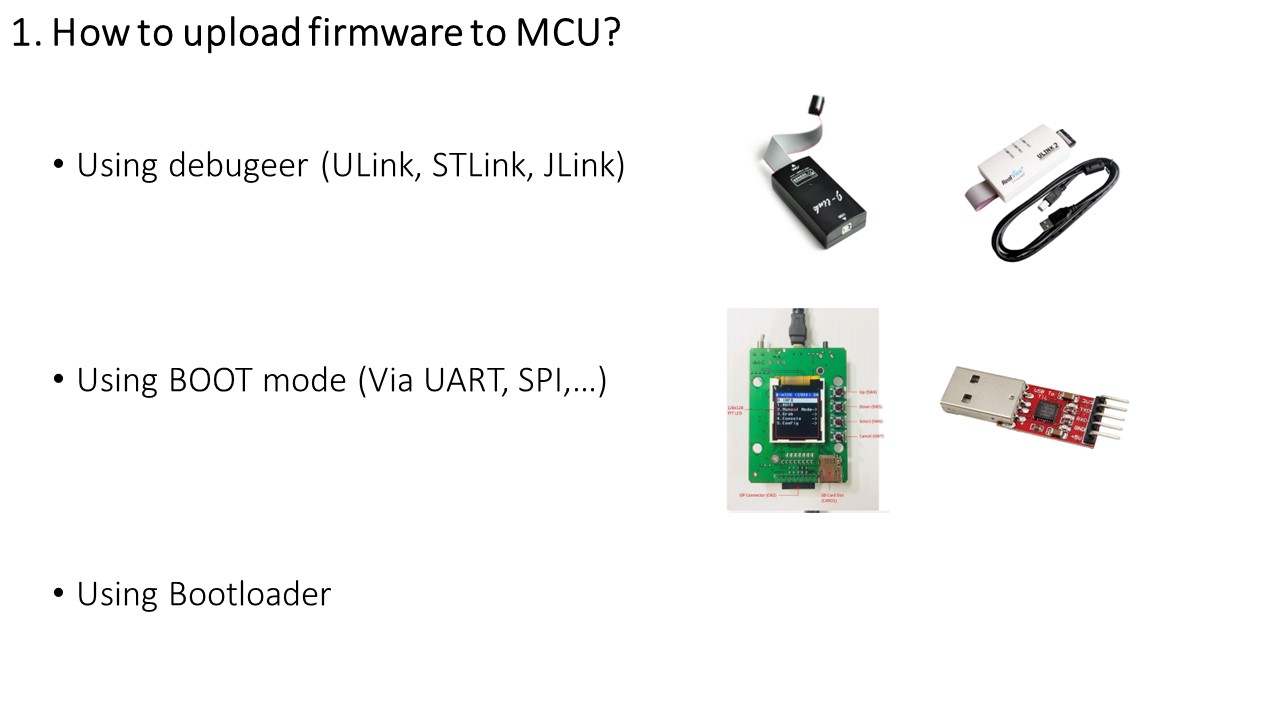
- Một trong những phần quan trọng nhất của bootloader là hàm nhảy tới chương trình chính. Nó có chức năng nhảy tới vị trí chưa chương trình

chính mà bạn muốn thực thi.

- Chương trình chính thường được đặt tại sector (Nhiều vi điều khiển gọi là Block) tiếp theo so với bootloader (Tùy vào kính thước bootloader)

- Nhìn chung, hàm này thường có các thành phần như: deinit tất cả các ngoại vi đang sử dụng, đặt lại main stack pointer và nhảy tới hàm reset handler.

(Xem ảnh để biết được 1 hàm nhảy tới chương trình chính viết cho stm32)



𝑻𝒊̀𝒎 𝒉𝒊𝒆̂̉𝒖 𝒃𝒐𝒐𝒕𝒍𝒐𝒂𝒅𝒆𝒓 𝒕𝒓𝒆̂𝒏 𝒗𝒊 đ𝒊𝒆̂̀𝒖 𝒌𝒉𝒊𝒆̂̉𝒏 - 𝑷𝒉𝒂̂̀𝒏 2

Tiếp tục series tìm hiểu về bootloader trên vi điều khiển, trong bài viết này mình sẽ

xây dựng 1 chương trình bootloader hỗ trợ nạp firmware cho stm32 qua cổng UART

1. 𝑻𝒐̂̉𝒏𝒈 𝒒𝒖𝒂𝒏

Vi điều khiển sử dụng: stm32f769nih6 (STM32f769i-disco)

IDE: Keil C

Desktop application: Qt framework

2. 𝑪𝒉𝒖̛𝒐̛𝒏𝒈 𝒕𝒓𝒊̀𝒏𝒉 𝒃𝒐𝒐𝒕𝒍𝒐𝒂𝒅𝒆𝒓

Chương trình được viết theo kiểu State Machine

Gồm các lớp:

- Application

+ App\_Booloader.c/h: thực hiện chức năng nhảy đến chương trình chính

- Middleware

+ Mid\_PCudeProtocol.c/h: gửi/giải mã gói tin từ PC gửi xuống qua UART

+ Mid\_FlashControl.c/h: thực hiện chức năng đọc ghi vào bộ nhớ flash

+ Mid\_KeyControl.c/h: tạo các sự kiện cho nút nhấn

- Device

+ Dev\_Key.c/h: đọc input

- HAL

+ Thư viện HAL của STM32

- Operation & system timer

+ main.c: chương trình chính

+ Op\_HeaderFlag.h chứa các cờ được sử dụng trong chương trình

+ Op\_HeaderFunction.c/h chứa các hàm chung được sử dụng trong chương trình

+ Op\_HeaderDefine.h chứa các macro được sử dụng trong chương trình

+ Op\_HeaderMain.h include các Op\_Header

+ Op\_HeaderVariable.h chứa tất cả các biến toàn cục dùng trong chương trình

+ sys\_timer.c/h tạo sự kiện thời gian

3. 𝑪𝒉𝒖̛𝒐̛𝒏𝒈 𝒕𝒓𝒊̀𝒏𝒉 𝒏𝒂̣𝒑 𝒇𝒊𝒓𝒎𝒘𝒂𝒓𝒆 𝒕𝒓𝒆̂𝒏 𝒎𝒂́𝒚 𝒕𝒊́𝒏𝒉

Gồm các lớp:

- Operation & system timer

+ Op\_HeaderFlag.h chức các cờ được sử dụng trong chương trình

+ Op\_HeaderFunction.c/h chứa các hàm chung được sử dụng trong chương trình

+ Op\_HeaderDefine.h chứa các macro được sử dụng trong chương trình

+ Op\_HeaderVariable.h chứa tất cả các biến toàn cục dùng trong chương trình

+ Op\_interruptmanager.cpp/h giao tiếp với file giao diện (Qml)

- Middleware

+ Mid\_PCubeProcess.cpp/h gửi/giải mã các gói tin gửi từ vi điều khiển

+ Mid\_ReadFile.cpp/h thực hiện chức năng đọc file

- HAL

+ Thư viện của Qt

- GUI (Qml)