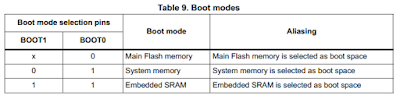
**NẠP CODE VỚI BOOT LOADER STM32**

1. Boot loader là gì

[](https://2.bp.blogspot.com/-9Fm0vRT13ZI/WsNHhRMgv5I/AAAAAAAAAps/6OjQVPTy24A06UvzlogfpfC-sfdz9TosACKgBGAs/s1600/hoi.jpg)

**Nhìn sơ đồ chân của vi điều khiển STM32 có khi nào các bạn thắc mắc chân BOOT0, BOOT1 để làm gì không?? Không tự dưng mà nhà sản xuất lại thiết kế chúng mà không có ứng dụng gì. Mục đích của chân này để phân chia các chuẩn nạp code cho MCU tùy thuộc vào mục đích của người sử dụng. Các loại STM32Fxx khác cũng gần giống như STM32F1, hệ thống các chuẩn nạp được phân chia như sau:**

[](https://4.bp.blogspot.com/-ZmiP4oMfO1Q/WsM85PTCXYI/AAAAAAAAAn8/xUl_Lwt7ZOUfRiVTwFQbnmvgpSlyMikugCKgBGAs/s1600/boot_01.png)

**Có 3 chuẩn mode như hình ở trên:**

* + Main flash memory: chuẩn này là chuẩn nạp code thông thường của MCU(dùng các mạch nạp như Stlink, Jlink….) thiết kế của nó là chân BOOT0 sẽ được nối đất. Vì lý do này nên chân BOOT đươc thiết kế là nối xuống GND thông qua trở kéo nguồn. Không có sự kết nối này, các mạch nạp sẽ không nhận dạng được IC được nạp và sẽ bị báo lỗi không tìm thấy MCU.
  + System memory : đây là chuẩn nạp code đặc biệt của MCU thông qua những chương trình nạp mà nhà sản xuất cung cấp. Tùy vào dòng MCU mà có các ngoại vi nạp khác nhau như : UART, SPI, USB, I2C.. Ở STM32F103C8T6 chuẩn nạp này chỉ được áp dụng ở UART1 trên chân PA9,PA10. Điều kiện là chân BOOT0 nối nguồn 3V3 và chân BOOT1(PB2) nối GND.
  + Embedded SRAM: chuẩn này cho phép truy cập địa chỉ SRAM nội bên trong IC. Điều kiện là chân BOOT0 nối nguồn 3V3 và chân BOOT1(PB2) nối 3V3. Chuẩn này mình chưa sử dụng, hy vọng có ai đó bổ sung chuẩn này.

1. Sử dụng boot loader với chuẩn UART.

**Chương trình nạp: flash loader demo.**

**Module UART: các module UART có bán trên thị trường : USB sang UART dùng PL2303, CH340, CP2102, FT232RL….**

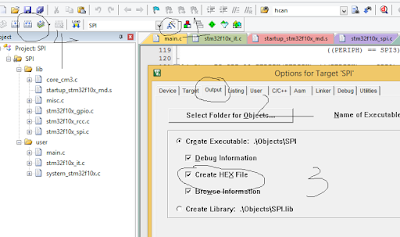
**Yêu cầu phần cứng:**

* + Máy tính nhận cổng COM của các module UART(đã cài đặt driver….)
  + Cấp nguồn cho module UART, board STM32.
  + Chân BOOT0 nối 3V3, chân BOOT1 nối GND.
  + Có file code nạp đuôi ở 3 dạng file .s19; .hex; .bin.
  + Kiểm tra các bus nối đã được kết nối, không bị lỏng bất cứ chân nào.

**Các bước thực hiện:**

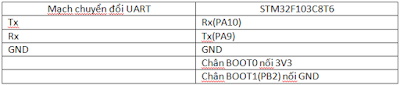
* + Tạo file hex trong Keil C.

**Mở keil C lên vào options for Target bằng các kích vào biểu tượng hoặc project -> Options for target, vào tab Output, click vào ô Create hex file sau đó nhấn Rebuild(F7).**

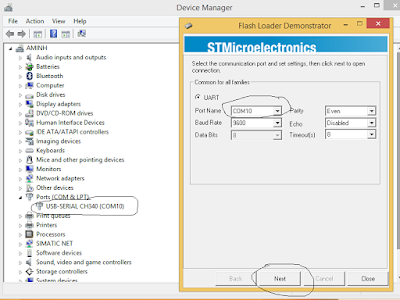
[](https://1.bp.blogspot.com/-4seTnoc2K-k/WsM8_f3ZsyI/AAAAAAAAAoE/BUiQW07GpwYoiG_M5BwdbYC31GAB1vkKACKgBGAs/s1600/boot_02.png)

**Như vậy là file hex đã được tạo, kiểm tra trong thư mục Object của project.**

* + Đấu nối các dây kết nối.

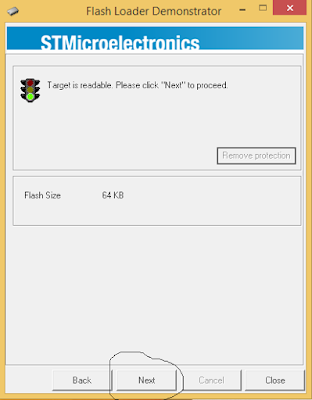
[](https://3.bp.blogspot.com/-vLG8habXOa8/WsM9NWFtImI/AAAAAAAAAoM/zEuKJ5qEFq0tjid_OI5ydIDBpy2Mr3_JgCKgBGAs/s1600/boot_03.png)

* + Mở phần mềm flash loader demo và thực hiện theo như sau.
    - Chọn cổng COM tương ứng với cổng COM của module UART. Các thông số khác giữ nguyên, ngoại trừ baud rate(có thể thay đổi). Nhấn Next để tiếp tục.

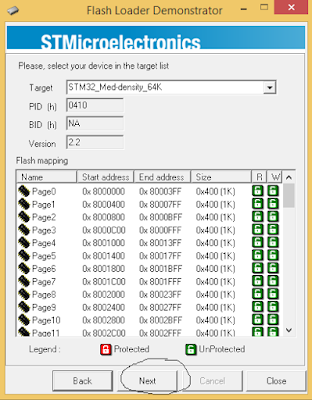
[](https://2.bp.blogspot.com/-MkMQwEo2_Cs/WsM9GrP2WAI/AAAAAAAAAoI/IVHM9TnI5e8Au1X3ajaoN9jNnxdHt92KACKgBGAs/s1600/boot_04.png)

Lưu ý: Bạn có thể mất vài lần cắm nếu phần mềm không kết nối được với MCU do một vài lỗi(Cắm dây lỏng, đấu nối dây sai hay thiếu, mạch bị hư…). Hãy reset lại mạch và kiểm tra kỹ lại.

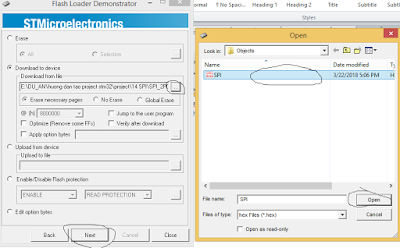
* + - Kích thước bộ nhớ flash của MCU sẽ xuất hiện. Nhấn next để tiếp tục.

[](https://1.bp.blogspot.com/-S4Ma0fiuNfM/WsM9SoBwAqI/AAAAAAAAAoQ/0els3U6Kc_IKqFfzv8F8lBcjRcFiM3LugCKgBGAs/s1600/boot_05.png)

* + - Loại MCU và các page nhớ xuất hiện, nhấn Next để tiếp tục.

[](https://4.bp.blogspot.com/-LcFHowgYG-g/WsM9WA5RF7I/AAAAAAAAAoU/OD2InU96kNAuxgapk38d-jfZ567yy8kBQCKgBGAs/s1600/boot_06.png)

* + - Xuất hiện Tab tiếp theo với nhiều lựa chọn, ở đây mình chọn là Nạp code mới và chọn đường dẫn đến file hex mà mình đã chuẩn bị. Nhấn Next để tiếp tục

[](https://3.bp.blogspot.com/-6RJEL54ykuA/WsM9a9jt6GI/AAAAAAAAAoY/gUsdgI1fPuEFKCvRZa42WONdDTa7vkpZwCKgBGAs/s1600/boot_07.png)

* + - Nhấn Next để nạp, sau khi nạp xong sẽ có thông báo đã nạp xong.

[](https://4.bp.blogspot.com/-X_XTjw8GarY/WsM9dxhRv5I/AAAAAAAAAoc/sX6-o8aF9lYXpYqaTYGuykuBlCAVi2pEQCKgBGAs/s1600/boot_08.png)

* + - Sau khi nạp xong, đưa mạch trở về trạng thái Main flash memory(BOOT0 nối đất, BOOT1 tùy chỉnh). Reset mạch, mạch chạy bình thường.

**Ưu điểm của phương pháp này là không cần những mạch nạp đắt tiền vẫn nạp được code cho MCU, ứng dụng để tiết kiệm chân VĐK(PA14, PA13, PB3.. những chân cho mạch nạp) mà vẫn đầy đủ các chức năng cơ bản. Đó cũng là công nghệ giúp bảo mật các thiết kế(tìm hoài trên mạch không thấy chân nạp thông thường….).**

**Nhược điểm: khá nhiều thao tác(cắm, cắm, reset, lỏng dây..), rườm rà, không debug được lỗi, dễ xảy ra sai sót, cũng như không nhiều chức năng được thêm vào. Và thường chỉ được sử dụng khi không có mạch nạp.**

//////////////////////////////////////////////////////

///////////////////////////////////////////////

Ngoài trường hợp thiết bị được cấp nguồn hoặc người dùng nhấn vào nút Reset của thiết bị thì ARM cung cấp hàm NVIC\_SystemReset() để thực hiện khởi động lại vi xử lý. Trong ứng dụng cập nhật chương trình từ xa, hàm này sẽ được gọi tại chương trình ứng dụng khi phát hiện một phiên bản chương trình mới và tại chương trình OTA khi thực hiện xong quá trình OTA. Sau khi hàm NVIC\_SystemReset() được thực thi thì vi xử lý sẽ khởi động lại và thực hiện chương trình Bootloader. Trong bài viết này, mình sẽ trình bày cho các bạn về thiết kế Bootloader cho vi điều khiển STM32 trong ứng dụng cập nhật chương trình từ xa theo thiết kế thực tế mà mình đã thực hiện.

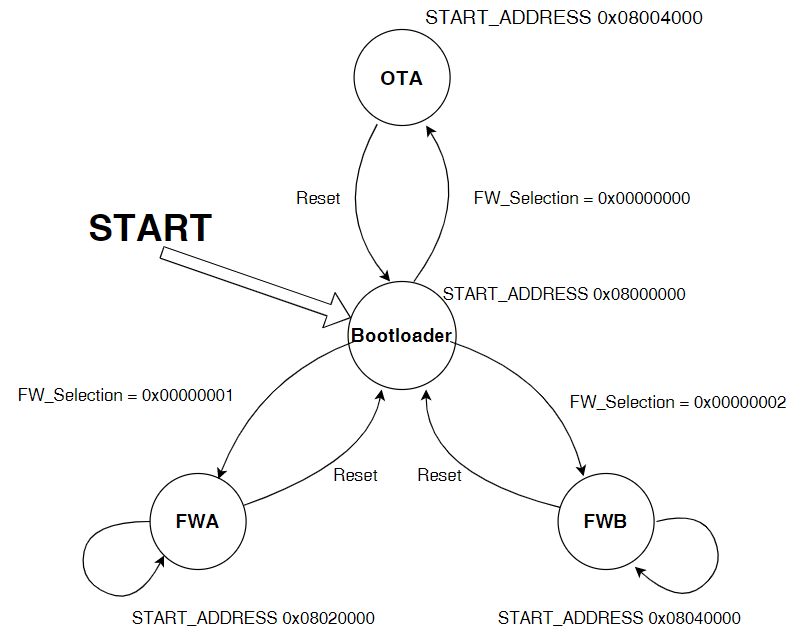
Để hiểu rõ về cập nhật chương trình từ xa và các nội dung của bài viết này, mời các bạn xem trước các bài viết có liên quan:

* [Cập nhật chương trình từ xa – Một tính năng quan trọng](https://tapit.vn/cap-nhat-chuong-trinh-tu-xa-mot-tinh-nang-quan-trong/)
* [Mô hình và quá trình cập nhật chương trình từ xa cho thiết bị nhúng, IoT](https://tapit.vn/mo-hinh-va-qua-trinh-cap-nhat-chuong-trinh-tu-xa-cho-thiet-bi-nhung-iot/)
* [Tổ chức bộ nhớ vi điều khiển lõi ARM Cortex M3/4 – Áp dụng FOTA cho STM32](https://tapit.vn/chuc-bo-nho-vi-dieu-khien-loi-arm-cortex-m3-4-ap-dung-fota-cho-stm32/)
* [Cấu trúc chương trình trong bộ nhớ vi điều khiển khi triển khai cập nhật chương trình từ xa](https://tapit.vn/mo-hinh-va-qua-trinh-cap-nhat-chuong-trinh-tu-xa-cho-thiet-bi-nhung-iot/)

Nhiệm vụ chính quan trọng hàng đầu của chương trình Bootloader là kiểm tra các điều kiện tại các biến liên kết để lựa chọn thực thi một trong các chương trình: Chương trình cập nhật từ xa OTA, Chương trình ứng dụng A hoặc Chương trình ứng dụng B mỗi khi CPU khởi động.

Ví dụ đối với trường hợp thử nghiệm cập nhật chương trình từ xa do mình thực hiện. Sau khi khởi động, tại chương trình Bootloader sẽ kiểm tra giá trị biến FW\_Selection, nếu:

* FW\_Selection == 0x00000000 thì Vi xử lý sẽ thực thi chương trình OTA được lưu từ địa chỉ 0x08004000
* FW\_Selection == 0x00000001 thì Vi xử lý sẽ thực thi chương trình Ứng dụng FWA được lưu từ địa chỉ 0x08020000
* FW\_Selection == 0x00000002 thì Vi xử lý sẽ thực thi chương trình Ứng dụng FWB được lưu từ địa chỉ 0x08040000

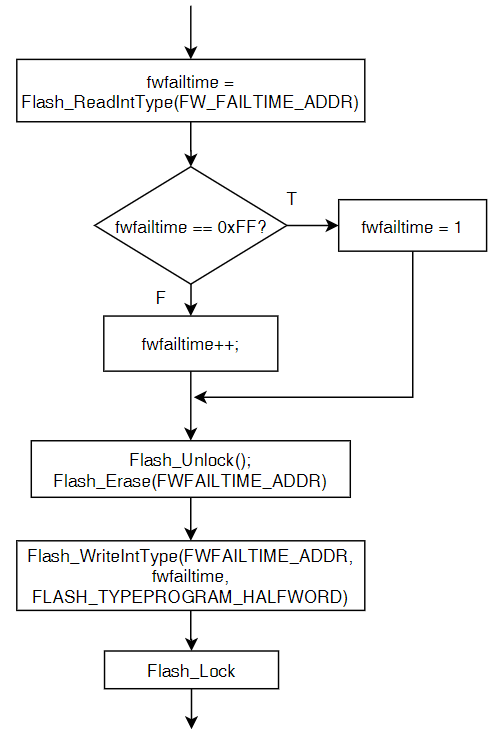


Hình 1. Sơ đồ trạng thái các chương trình trong vi điều khiển

Bên cạnh việc kiểm tra giá trị biến FW\_Selection để quyết định chương trình được thực thi thì Bootloader còn được mình thiết kế để đảm bảo phát hiện lỗi chương trình ứng dụng sau khi cập nhật, từ đó chương trình Bootloader sẽ lựa chọn thực thi chương trình ứng dụng chạy được trước đó – quay trở về (rollback) phiên bản cũ.   
Nếu chương trình ứng dụng FWA hoặc FWB mới được cập nhật tồn tại lỗi làm cho thiết bị khởi động lại liên tục (fwfailtime) quá 3 lần thì thuật toán đảm bảo thiết bị hoạt động với phiên bản trước của chương trình để đảm bảo các chức năng ứng dụng và có khả năng cập nhật bảng mới hơn [1][2]. Mỗi lần khởi động, chương trình Bootloader sẽ tăng fwfailtime lên 1 đơn vị, nếu chương trình ứng dụng chạy thành công đến cuối chương trình thì xem như chương trình được chạy thành công và đặt lại giá trị fwfailtime là 0.

Vị trí của chương trình Bootloader thường được bắt đầu tại địa chỉ đầu tiên của bộ nhớ Flash, đây là địa chỉ mặc định sẽ được CPU thực thi sau khi reset. Với dòng vi điều khiển STM32 thì vị trí bắt đầu của bộ nhớ Flash là 0x0800\_0000. Chương trình Booloader này sẽ được người dùng lập trình và nạp thủ công vào vi điều khiển. Thuật toán khởi động vi điều khiển phải được thiết kế như sau:

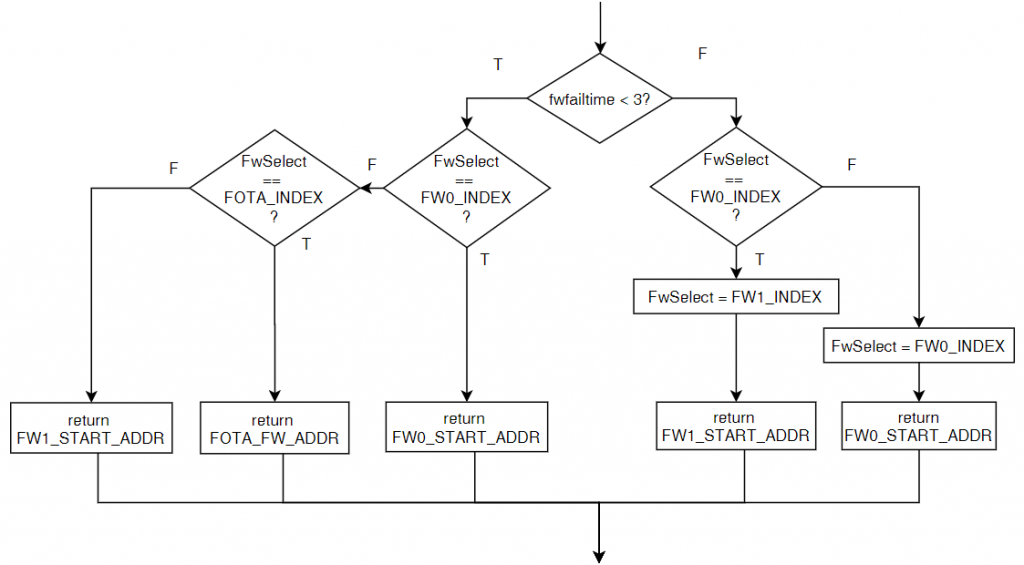
**Bước 1:** Khi thiết bị khởi động, chương trình Bootloader sẽ được thực hiện. Tại Bootloader, tăng biến đếm fwfailtime lên 1 đơn vị.



Hình 2. Giản đồ quy trình kiểm tra và làm việc với biến fwfailtime

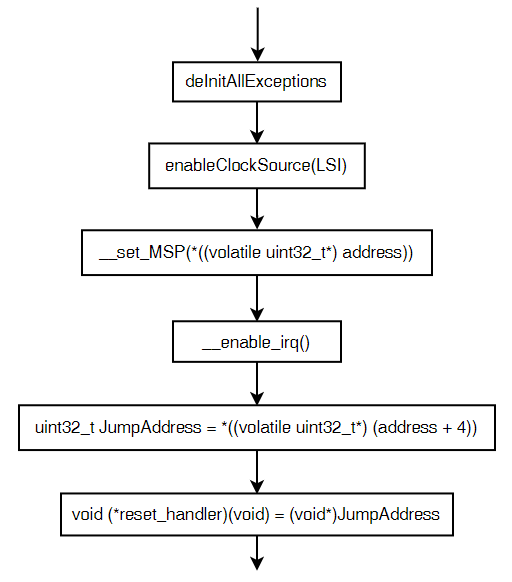
**Bước 2:** Xác định chương trình cần chạy trong lần khởi động này là chương trình nào thông qua giá trị biến liên kết được đọc ra từ bộ nhớ Flash.

**Bước 3:** Xác định địa chỉ chương trình cần chạy: Kiểm tra giá trị fwfailtime, nếu giá trị fwfailtime bé hơn 3 thì tùy vào giá trị ở bước 2 sẽ xác định được địa chỉ bắt đầu của chương trình cần nhảy đến. Nếu giá trị fwfailtime bằng 3 thì địa chỉ bắt đầu của chương trình cần nhảy đến là địa chỉ chương trình của phiên bản trước.

  
Hình 3. Giản đồ quy trình lựa chọn chương trình cần thực thi

**Bước 4:** Thưc hiện theo các chỉ dẫn của ARM [3] để nhảy đến và thực thi chương trình đã xác định bao gồm:

* Tắt hết các ngoại vi, các ngoại lệ.
* Bật bộ dao động nội tần số thấp để cấp cho ngoại vi RTC.
* Đặt giá trị Main Stack Pointer
* Gán địa chỉ của hàm main() của chương trình đó cho thanh ghi PC



Hình 4. Giản đồ quy trình nhảy đến một chương trình khác từ bootloader

Qua bài viết này, mình đã chia sẻ một trường hợp thiết kế xây dựng Bootloader thực tế ứng dụng cho tính năng cập nhật chương trình từ xa, ngoài tính năng chính là lựa chọn chương trình thực thi thì bootloader còn được thiết kế để đảm bảo thiết bị vẫn hoạt động khi cập nhật một chương trình lỗi chương trình bị lỗi.

////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////

Lập trình Bootloader Update Firmware (P1)

Nguyễn Dũng

# ****I. Kiến thức cần chuẩn bị****

## **1. Chương trình Bootloader**

Bootloader là một chương trình và là chương trình đầu tiên chạy khi Chip hoạt động.

Nhiệm vụ của Bootloader là kiểm tra các điều kiện để lựa chọn thực thi một trong các chương trình: **FOTA**(bản Firmware update mới nhất), **Factory Firmware**(bản Firmware được nạp vào do hãng cung cấp trước khi xuất ra thị trường) hoặc **Current Firmware**(bản Firmware hiện tại đang chạy) mỗi khi CPU reset. Vị trí của chương trình Bootloader thường được bắt đầu tại địa chỉ đầu tiên của bộ nhớ Flash, đây là địa chỉ mặc định sẽ được CPU thực thi sau khi Reset. Với dòng vi điều khiển STM32 thì vị trí bắt đầu của bộ nhớ Flash là **0x08000000**.

Chương trình này sẽ được người dùng lập trình và nạp thủ công vào Chip.

Dưới đây là mô hình bộ nhớ đơn giản khi sử dụng chương trình Bootloader mà mình sử dụng với vi điều khiển STM32:

Bootloader và Application chạy riêng biệt lưu trên 2 vùng nhớ khác nhau . Địa chỉ mặc định khi chạy của Chip. Các bài trước khi ta tạo Project thì địa chỉ mặc định của App sẽ lưu tại File “**\*.sct**” trong thư mục Build. Tại đây địa chỉ mặc định là **0x08000000**.

Các thanh ghi liên quan đến quá trình khởi động và thực thi chương trình.

* **Stack Pointer (SP):** Thanh ghi lưu trữ giá trị trỏ tới vùng nhớ hiện tại trong bộ nhớ Stack. Mỗi chương trình sẽ có giá trị khởi tạo SP khác nhau, giá trị khởi tạo của SP sẽ được lưu tại địa chị bắt đầu của chương trình.
* **Program Counter (PC):** Thanh ghi chứa địa chỉ câu lệnh tiếp theo sẽ được thực thi. Thanh ghi này sẽ tự động tăng lên mỗi khi thực hiện xong một lệnh. Khi muốn chuyển sang chương trình mới, thanh ghi PC phải lấy giá trị từ địa chỉ bắt đầu chương trình + 4. Vì đây là nơi lưu trữ địa chỉ của Reset Handler, và hàm main() chương trình sẽ được thực thi từ đây.

Để một chương trình được thực thi thì cần có bảng **Vector Table** và mặc định khi không cấu hình thay đổi thì bảng **Vector Table** này nằm ở phần đầu tiên của bộ nhớ của bộ nhớ Flash. Bảng **Vector Table** chứa giá trị Reset của con trỏ Stack Pointer và địa chỉ cho tất cả exception handlers. Với chương trình Bootloader thì chúng ta không cần thay đổi vị trí bảng **Vector Table** vì Bootloader nằm ngay ở vị trí đầu tiên của bộ nhớ Flash.(Address = 0x08000000). Còn các chương trình còn lại gồm FOTA, Current Firmware và Factory Firmware nằm ở các vị trí khác trên bộ nhớ Flash nên chúng ta cần thay đổi địa chỉ bảng **Vector Table** tương ứng với vị trí bắt đầu của từng chương trình.

Sau khi chương trình Bootloader thực hiện xong ghi giá trị cho thanh ghi SP và CP thì chương trình sẽ nhảy tới một trong các chương trình Firmware tùy điều kiện.

Sau đó bắt buộc phải dời bảng vector ngắt ngay khi bắt đầu thực thi các chương trình ứng dụng.

**Ý tưởng Demo : Mình sẽ nạp 2 chương trình vào bộ nhớ Flash của chip STM32F411, chương trình 1 là chương trình Bootloader bắt đầu tại địa chỉ 0x8000000. Chương trình 2 là chương trình Blink LED đặt tại địa chỉ 0x8004000 trong Flash. Khi mình Reset Kit, chương trình Bootloader được thực thi, tại đây mình sẽ sử dụng lệnh để nhảy tới chương trình Blink LED.**

## **II. Lập trình**

## **1.    Tạo chương trình Blink LED.**

Vẫn như mọi bài Application sẽ tạo bằng CubeMX. Các bạn có thể xem lại bài thực hành STM32F4 với GPIO nhé, ở đây mình sẽ không nhắc lại các bước nữa vì nó khá đơn giản.

Mình sẽ đặt địa chỉ ghi chương trình **Blink LED** tại **0x8004000**nhé.

Tiếp theo mình cần cấu hình bảng Vector Table tới địa chỉ **0x8004000**. Ở đây mình sẽ có 2 cách để thực hiện việc này.

**Cách 1**: Các bạn mở file **system\_stm32f4xx.c** và sửa thành

**#define VECT\_TAB\_OFFSET 0x4000**

**Cách 2**: Ngay sau khi vào hàm main các bạn thêm dòng code sau

Tiếp đó, các bạn Build Project và nhớ đường dẫn tới output file **Blink.hex** để tiến hành nạp xuống KIT nhé.

Để nạp, các bạn tải và cài đặt phần mềm STM32 ST-LINK Utility theo link sau:

<https://www.st.com/en/development-tools/stsw-link004.html>

Sau khi cài xong, các bạn mở lên và làm theo các bước sau.

Chọn **Target → Connect**.

Chọn **Target → Erase Chip** để xóa toàn bộ Flash Chip.

Kết quả được như hình.

Để nạp chương trình **Blink.hex** vào địa chỉ **0x8004000**, các bạn chọn **Target → Program…**

Chọn **Browse** và tìm đến file **Blink.hex,**sau đó ấn **Start** và đợi phần mềm nạp xong nhé !

Kết quả thành công như hình

## **2. Tạo chương trình Bootloader**

Tiếp theo chúng ta cùng nhau tạo chương trình Bootloader. Nó cũng chỉ là một chương trình bình thường nên các bạn sẽ tạo y các bước như chương trình Blink nhé !

Mở phần mềm STM CubeMX, chọn dòng chip bạn sử dụng. Ở đây mình chọn chip STM32F411VET.

Đối với các dòng chip STM32 đời 4, tất cả mọi câu lệnh khi sử dụng thư viện HAL đều giống nhau. Chỉ khác nhau phần cấu hình Clock phụ thuộc riêng vào mỗi Chip.

Cấu hình Chip Debug bằng mode SWD

Cấu hình Clock hoạt động. Ở đây mình chọn sử dụng nguồn Clock là xung Clock nội và chọn giá trị tối đa 16MHz. Clock đi qua bộ nhân tần PLLCLK để đạt được tần số hoạt động tối đa mà chip hỗ trợ HCLK = 100MHz. Việc còn lại Cube MX sẽ tự cấu hình cho các bạn.

Cuối cùng chọn file và sinh code cho Project.

Chọn những thư viện cần thiết để sinh code nhanh hơn và giảm dung lượng Project nhé.

Trong hàm main, mình sử dụng code như sau.

/*\* USER CODE BEGIN Header \**/

/**\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

\* @file : main.c

\* @brief : Main program body

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

***\* @attention***

***\*\* <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.***

***\* All rights reserved.</center></h2>\****

***\* This software component is licensed by ST under BSD 3-Clause license,***

***\* the "License"; You may not use this file except in compliance with the***

***\* License. You may obtain a copy of the License at:***

***\* opensource.org/licenses/BSD-3-Clause***

***\****

***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***

***/\* USER CODE END Header \*/***

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

***#include "main.h"***

***void SystemClock\_Config(void);***

***static void MX\_GPIO\_Init(void);***

***uint32\_t ADDRESS\_START\_BLINK\_LED\_APPLICATION = 0x08004000;***

***/\*\****

***\* @brief The application entry point.***

***\* @retval int***

***\*/***

***int main(void)***

***{***

***/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/***

***HAL\_Init();***

***/\* Configure the system clock \*/***

***SystemClock\_Config();***

***/\* Initialize all configured peripherals \*/***

***MX\_GPIO\_Init();***

***/\* Turn off Peripheral, Clear Interrupt Flag\*/***

***HAL\_RCC\_DeInit();***

***/\* Clear Pending Interrupt Request, turn off System Tick\*/***

***HAL\_DeInit();***

***/\* Turn off fault harder\*/***

***SCB->SHCSR &= ~( SCB\_SHCSR\_USGFAULTENA\_Msk |\***

***SCB\_SHCSR\_BUSFAULTENA\_Msk | \***

***SCB\_SHCSR\_MEMFAULTENA\_Msk ) ;***

***/\* Set Main Stack Pointer\*/***

***\_\_set\_MSP(\*((volatile uint32\_t\*) ADDRESS\_START\_BLINK\_LED\_APPLICATION));***

***uint32\_t JumpAddress = \*((volatile uint32\_t\*) (ADDRESS\_START\_BLINK\_LED\_APPLICATION + 4));***

***/\* Set Program Counter to Blink LED Apptication Address\*/***

***void (\*reset\_handler)(void) = (void\*)JumpAddress;***

***reset\_handler();***

***/\* Infinite loop \*/***

***/\* USER CODE BEGIN WHILE \*/***

***while (1)***

***{***

***/\* USER CODE END WHILE \*/***

***/\* USER CODE BEGIN 3 \*/***

***}***

***/\* USER CODE END 3 \*/***

***}***

***/\*\****

***\* @brief System Clock Configuration***

***\* @retval None***

***\*/***

***void SystemClock\_Config(void)***

***{***

***RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};***

***RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};***

***/\*\* Configure the main internal regulator output voltage \*/***

***\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();***

***\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);***

***/\*\* Initializes the CPU, AHB and APB busses clocks\*/***

***RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;***

***RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;***

***RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;***

***RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;***

***RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;***

***RCC\_OscInitStruct.PLL.PLLM = 8;***

***RCC\_OscInitStruct.PLL.PLLN = 100;***

***RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;***

***RCC\_OscInitStruct.PLL.PLLQ = 4;***

***if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)***

***{***

***Error\_Handler();***

***}***

***/\*\* Initializes the CPU, AHB and APB busses clocks\*/***

***RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK |RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;***

***RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;***

***RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;***

***RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;***

***RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;***

***if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_3) != HAL\_OK)***

***{***

***Error\_Handler();***

***}***

***}***

***/\*\****

***\* @brief GPIO Initialization Function***

***\* @param None***

***\* @retval None***

***\*/***

***static void MX\_GPIO\_Init(void)***

***{***

***GPIO\_InitTypeDef GPIO\_InitStruct = {0};***

***/\* GPIO Ports Clock Enable \*/***

***\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();***

***\_\_HAL\_RCC\_GPIOD\_CLK\_ENABLE();***

***/\*Configure GPIO pin Output Level \*/***

***HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_12|GPIO\_PIN\_13|GPIO\_PIN\_14|GPIO\_PIN\_15, GPIO\_PIN\_RESET);***

***/\*Configure GPIO pin : PA0 \*/***

***GPIO\_InitStruct.Pin = GPIO\_PIN\_0;***

***GPIO\_InitStruct.Mode = GPIO\_MODE\_IT\_RISING;***

***GPIO\_InitStruct.Pull = GPIO\_NOPULL;***

***HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);***

***/\*Configure GPIO pins : PD12 PD13 PD14 PD15 \*/***

***GPIO\_InitStruct.Pin = GPIO\_PIN\_12|GPIO\_PIN\_13|GPIO\_PIN\_14|GPIO\_PIN\_15;***

***GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;***

***GPIO\_InitStruct.Pull = GPIO\_NOPULL;***

***GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_HIGH;***

***HAL\_GPIO\_Init(GPIOD, &GPIO\_InitStruct);***

***/\* EXTI interrupt init\*/***

***HAL\_NVIC\_SetPriority(EXTI0\_IRQn, 0, 0);***

***HAL\_NVIC\_EnableIRQ(EXTI0\_IRQn);***

***}***

***/\* USER CODE BEGIN 4 \*/***

***/\* USER CODE END 4 \*/***

***/\*\****

***\* @brief This function is executed in case of error occurrence.***

***\* @retval None***

***\*/***

***void Error\_Handler(void)***

***{***

***/\* USER CODE BEGIN Error\_Handler\_Debug \*/***

***/\* User can add his own implementation to report the HAL error return state \*/***

***/\* USER CODE END Error\_Handler\_Debug \*/***

***}***

***#ifdef USE\_FULL\_ASSERT***

***/\*\****

***\* @brief Reports the name of the source file and the source line number***

***\* where the assert\_param error has occurred.***

***\* @param file: pointer to the source file name***

***\* @param line: assert\_param error line source number***

***\* @retval None***

***\*/***

***void assert\_failed(uint8\_t \*file, uint32\_t line)***

***{***

***/\* USER CODE BEGIN 6 \*/***

***/\* User can add his own implementation to report the file name and line number,***

***tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/***

***/\* USER CODE END 6 \*/***

***}***

***#endif /\* USE\_FULL\_ASSERT \*/***

***/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (C) COPYRIGHT STMicroelectronics \*\*\*\*\*END OF FILE\*\*\*\*/***

Tiếp theo, các bạn tiến hành Build và nạp code bằng STLink như bình thường.

Chú ý khi nạp có 1 số option như sau.

* **Erase Full Chip**: xóa toàn bộ Flash trước khi nạp, các bạn chọn Option này nó sẽ xóa luôn firmware Blink các bạn đã nạp ở phần 1.
* **Erase Sectors**: xóa Sector tính từ vị trí nạp Firmware, ở đây mình đặt Firmware Bootload ở 0x8000000. Vậy Option này sẽ xóa Sector 1 (từ 0x8000000 → 0x8003FFF). Nếu chọn Option này các bạn cần nạp Firmware Blink ngoài khoảng xóa, ở đây mình đặt chương trình Blink ở địa chỉ 0x8004000 nên vẫn an toàn.
* **Do not Erase**: không cần xóa, nạp luôn Firmware.

Cuối cùng mình Build chương trình và nạp xuống KIT.

Sau khi nạp, KIT của mình đã Blink được LED, chứng tỏ chương trình sau khi chạy vào Bootloader đã nhảy vào Blink Application thành công.

Cảm ơn các bạn đã theo dõi.

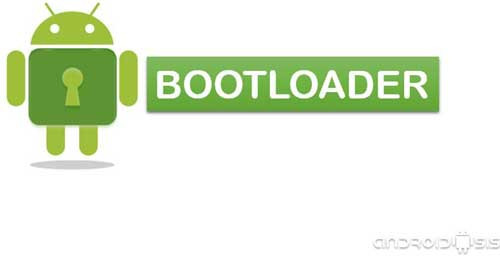
/////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////

////////////////////////////////

# Nạp chương trình cho stm32c8t6 qua bootloader

[27 Tháng Ba, 2020](http://iot47.com/2020/03/) [Đào Nguyện](http://iot47.com/author/daonguyen/) [Lập trình nhúng](http://iot47.com/category/lap-trinh-nhung/) [5](http://iot47.com/nap-chuong-trinh-cho-stm32c8t6-qua-bootloader/#mh-comments)



Khi làm 1 project có tính thương mại, việc update và bảo trì chương trình cho khách hàng nhất thiết phải có, bạn không thể đưa mạch nạp cho khách rồi bảo họ làm theo hướng dẫn để nạp file hex được. Thay vào đó chúng ta sẽ update firmwave khách qua chương trình bootloader

Một vài ứng dụng của chương trình bootloader

* Nạp code không cần mạch nạp gốc, chỉ cần 1 module UART-USB bất kì
* Làm mạch nạp offline (máy nạp code hàng loạt)
* Tạo chức năng FOTA (Firmware Over The Air) tức là cập nhật từ xa qua internet, wifi, bluetooth

Về phần lí thuyết các bạn có thể tìm hiểu về OTA tại đây:

* [**https://tapit.vn/cap-nhat-chuong-trinh-tu-xa-tren-vi-dieu-khien-stm32-p1/**](https://tapit.vn/cap-nhat-chuong-trinh-tu-xa-tren-vi-dieu-khien-stm32-p1/)
* [**https://tapit.vn/cap-nhat-chuong-trinh-tu-xa-tren-vi-dieu-khien-stm32-firmware-air-fota-p2/**](https://tapit.vn/cap-nhat-chuong-trinh-tu-xa-tren-vi-dieu-khien-stm32-firmware-air-fota-p2/)
* [**https://tapit.vn/cap-nhat-chuong-trinh-tu-xa-tren-vi-dieu-khien-stm32-firmware-air-fota-p3/**](https://tapit.vn/cap-nhat-chuong-trinh-tu-xa-tren-vi-dieu-khien-stm32-firmware-air-fota-p3/)
* [**https://tapit.vn/cap-nhat-chuong-trinh-tu-xa-tren-vi-dieu-khien-stm32-firmware-air-fota-p4/**](https://tapit.vn/cap-nhat-chuong-trinh-tu-xa-tren-vi-dieu-khien-stm32-firmware-air-fota-p4/)
* [**https://tapit.vn/cap-nhat-chuong-trinh-tu-xa-tren-vi-dieu-khien-stm32-firmware-air-fota-p5/**](https://tapit.vn/cap-nhat-chuong-trinh-tu-xa-tren-vi-dieu-khien-stm32-firmware-air-fota-p5/)

## DEMO chương trình bootloader

Do stm32f103c8t6 có 128 Page bắt đầu từ 0 và mỗi page là 128KB nên mình sẽ tạo 1 buffer đúng bằng kích cỡ của 1page, bởi mỗi lần ghi vào flash chúng ta phải ghi cả page chứ không thể ghi lẻ tẻ được.

Khơi tạo ram cho 1 page



|  |  |
| --- | --- |
| 1 | unsigned char code\_save[1024]; |

Chương trình xóa page, do mỗi lần ghi vào xóa page đó đi



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | void Erase\_PAGE(uint32\_t addr)  {  Flash\_Unlock();    while((FLASH->SR&FLASH\_SR\_BSY));    FLASH->CR |= FLASH\_CR\_PER; //Page Erase Set    FLASH->AR = addr; //Page Address    FLASH->CR |= FLASH\_CR\_STRT; //Start Page Erase    while((FLASH->SR&FLASH\_SR\_BSY));  FLASH->CR &= ~FLASH\_SR\_BSY;    FLASH->CR &= ~FLASH\_CR\_PER; //Page Erase Clear  Flash\_Lock();  } |

Chương trình đọc 1byte kiểu 16 ra từ 1 địa chỉ



|  |  |
| --- | --- |
| 1  2  3  4  5 | uint16\_t F\_read(uint32\_t addr)  {  uint16\_t\* val = (uint16\_t \*)addr;  return \*val;  } |

Chương trình ghi data vào 1 page + check lại, chúng ta sẽ cần nhập vào tham số là page vần ghi và size (thực ra cái này size mặc định là 1024 rồi)



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | char Write\_PAGE(uint32\_t size,uint32\_t page)  {  uint32\_t ADDRESS\_PAGE = ((uint32\_t)0x08000000 + (page\*1024)); //tinh dia chi cua page  uint32\_t addr;  uint16\_t data;  Flash\_Erase(ADDRESS\_PAGE);  Flash\_Unlock();  for(int i=0;i<size/2;i++)  {  addr = ADDRESS\_PAGE + (i\*2);  FLASH->CR |= FLASH\_CR\_PG; /\*!< Programming \*/  while((FLASH->SR&FLASH\_SR\_BSY));  data=((uint16\_t)code\_save[i\*2+1]<<8) | ((uint16\_t)code\_save[i\*2]);  \*(\_\_IO uint16\_t\*)addr = data;  while((FLASH->SR&FLASH\_SR\_BSY));  FLASH->CR &= ~FLASH\_CR\_PG;  }  //check lại  for(int i=0;i<size/2;i++)  {  addr = ADDRESS\_PAGE + (i\*2);  data=((uint16\_t)code\_save[i\*2+1]<<8) | ((uint16\_t)code\_save[i\*2]);  uint16\_t test2=F\_read(addr);  if(data != test2)  {  Flash\_Lock();  return 1;  }  }  Flash\_Lock();  return 0 ;  } |

Phương pháp giao tiếp với máy tính

Đầu tiên phần mềm sẽ gửi dữ liệu gồm số lượng byte của code, ở dưới đọc xong sẽ phẩn hồi OK cho app biết nó sẵn sàng nhận data. Sau đó app sẽ gửi liên tục gửi 1page (1024byte) xuống cho ở dưới nạp vào flash, sau khi nạp xong sẽ nhảy tới phần vùng thực thi code

Chương trình ngắt UART nhận dữ liệu



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart)  {  if(flag==0)  {  rx\_buff[count++]=rx\_data;  if(rx\_data == '\n')  {  flag=1;  int ts=1;  size\_program=0;size\_count=0;end=0;  page\_write=PROGRAM\_START\_PAGE;  for(count=count-2;count>0;count--)  {  if(rx\_buff[count] == '=')break;  size\_program+= (rx\_buff[count]-48)\*ts;  ts\*=10;  }  if(size\_program>1024)HAL\_UART\_Receive\_IT(&huart2,code\_save,1024);  else HAL\_UART\_Receive\_IT(&huart2,code\_save,size\_program);  HAL\_UART\_Transmit(&huart2,(uint8\_t \*)"OK MEN !\r\n",10,100);  }  else  {  HAL\_UART\_Receive\_IT(&huart2,(uint8\_t \*)&rx\_data,1);  }  }  else if(flag==1)  {  if(end==1)  {  HAL\_UART\_Receive\_IT(&huart2,(uint8\_t \*)&rx\_data,1);  if(Write\_PAGE(1024,page\_write) == 0)HAL\_UART\_Transmit(&huart2,(uint8\_t \*)"Read OK !\r\n",11,100);  else  {  flag=0;  HAL\_UART\_Transmit(&huart2,(uint8\_t \*)"Write Fall !\r\n",14,100);  HAL\_UART\_Receive\_IT(&huart2,(uint8\_t \*)&rx\_data,1);  }  clear\_buffer();  flag=0;  goto\_aplication=1; //chuyen toi ct BOOT  return;  }  size\_count+=1024;  if((size\_count+1024) < size\_program)  HAL\_UART\_Receive\_IT(&huart2,code\_save,1024);  else  {  HAL\_UART\_Receive\_IT(&huart2,code\_save,size\_program-size\_count);  end=1;  }  if(Write\_PAGE(1024,page\_write) == 0)HAL\_UART\_Transmit(&huart2,(uint8\_t \*)"Read OK !\r\n",11,100);  else  {  flag=0;  HAL\_UART\_Transmit(&huart2,(uint8\_t \*)"Write Fall !\r\n",14,100);  HAL\_UART\_Receive\_IT(&huart2,(uint8\_t \*)&rx\_data,1);  }  page\_write++;  clear\_buffer();  }  } |

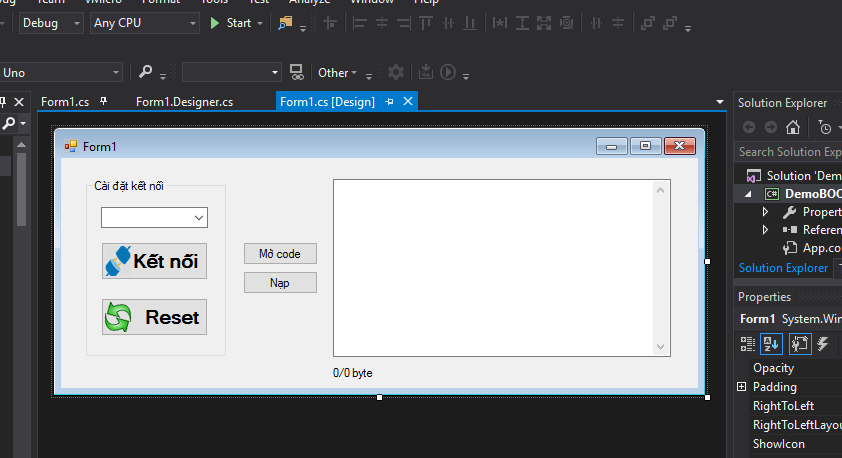
Sau khi nhận dầy đủ mình sẽ set cờ goto\_aplication=1

Ở hàm main check cờ và nhảy tới vùng code thực thi



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | if(goto\_aplication==1)  {  void (\*SysMemBootJump)(void);  volatile uint32\_t addr = PROGRAM\_START\_ADDRESS;  #if defined(USE\_HAL\_DRIVER)        HAL\_RCC\_DeInit();  #endif /\* defined(USE\_HAL\_DRIVER) \*/  #if defined(USE\_STDPERIPH\_DRIVER)  RCC\_DeInit();  #endif /\* defined(USE\_STDPERIPH\_DRIVER) \*/    SysTick->CTRL = 0;        SysTick->LOAD = 0;        SysTick->VAL = 0;    \_\_disable\_irq();    #if defined(STM32F4)        SYSCFG->MEMRMP = 0x01;  #endif  #if defined(STM32F0)  SYSCFG->CFGR1 = 0x01;  #endif    SysMemBootJump = (void (\*)(void)) (\*((uint32\_t \*)(addr + 4)));    \_\_set\_MSP(\*(uint32\_t \*)addr);    SysMemBootJump();    } |

## Demo chương trình c# gửi dữ liệu

Thiết kế giao diện

Hàm mở code



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | private void button3\_Click(object sender, EventArgs e)          {              OpenFileDialog diaglog = new OpenFileDialog();              diaglog.Filter = "HEX files|\*.hex";              if (diaglog.ShowDialog() == DialogResult.OK)              {                  for (int i = 0; i < max\_size; i++)                      data\_send[i] = 0xFF;  //clear bộ đệm gửi                  program\_size = 0;                    hex\_data = File.ReadAllText(diaglog.FileName);                  textBox1.Text = hex\_data;                    int \_2c\_line1 = hex\_data.IndexOf(":");                  int \_2c\_line\_end = hex\_data.LastIndexOf(":");                  hex\_data = hex\_data.Substring(\_2c\_line1+1, \_2c\_line\_end-1); // bỏ dòng 1 và dòng cuối                    \_2c\_line1 = hex\_data.IndexOf(":");                  \_2c\_line\_end = hex\_data.LastIndexOf(":");                  hex\_data = hex\_data.Substring(\_2c\_line1, \_2c\_line\_end - \_2c\_line1 - 1 ); // bỏ dòng 1 và dòng cuối                  //tiến hành lọc lấy phần mã hex                  while (true)                  {                      int count = 0;                      count = hex\_data.IndexOf(":"); //lấy vị tri bắt đầu                      if (count != -1) //nếu còn dữ liệu                      {                          byte[] data\_num = StringToByteArrayFastest(hex\_data.Substring(count + 1, 2)); //kiểm tra số lượng byte trong hàng                          int byte\_num = (int)data\_num[0];                          byte[] data\_line = new byte[byte\_num]; //số lượng byte data trong line đó                          data\_line = StringToByteArrayFastest(hex\_data.Substring(count + 9, byte\_num \* 2));                          //đưa data vào mảng                          for(int i=0;i< byte\_num;i++)                              data\_send[program\_size++] = data\_line[i];                          //lấy xong data thì cắt bỏ luôn                          hex\_data =hex\_data.Substring(count+1);                      }                      else                      {                          label1.Text = "0/" + program\_size.ToString() + " byte";                          return;                      }                  }              }          } |

Hàm nạp code xuống



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | private async void button4\_Click(object sender, EventArgs e)          {              // nạp xuống từng byte data một              int dem = 0;              //gửi thông điệp upload code              serialPort.Write("Upload=" + program\_size.ToString() + "\n");              flag\_check = 0;              await Task.Delay(50); //chờ phản hồi từ mạch              if (flag\_check == 1)              {                  while (true)                  {                      if(program\_size/1024 != dem)serialPort.Write(data\_send, dem \* 1024, 1024); //tryền từng mảng 1024 byte 1 lần                      else serialPort.Write(data\_send, dem \* 1024, program\_size-(1024\*dem)); //truyền nốt phần data cuối                      flag\_check = 0;                      timer1.Enabled = true;                      await Task.Delay(500); //chờ phản hồi                      if (flag\_check == 0)                      {                          MessageBox.Show("Lỗi");                          return;                      }                      if (flag\_check == 2)                      {                          MessageBox.Show("Lỗi khi ghi");                          return;                      }                      dem++;                      String vl = (dem \* 1024).ToString() + "/" + program\_size.ToString() + " byte";                      if (label1.InvokeRequired)                          label1.Invoke(new Action(() => label1.Text = vl));                      else                      { label1.Text = vl; Application.DoEvents(); } //cập nhật quá trình nạp                        if (dem\*1024 > program\_size)                      {                          label1.Text = "Nạp thành công !";                          MessageBox.Show("Xong");                          return;                      }                  }              }              else              {                  MessageBox.Show("Kiểm tra kết nối !");              }          } |

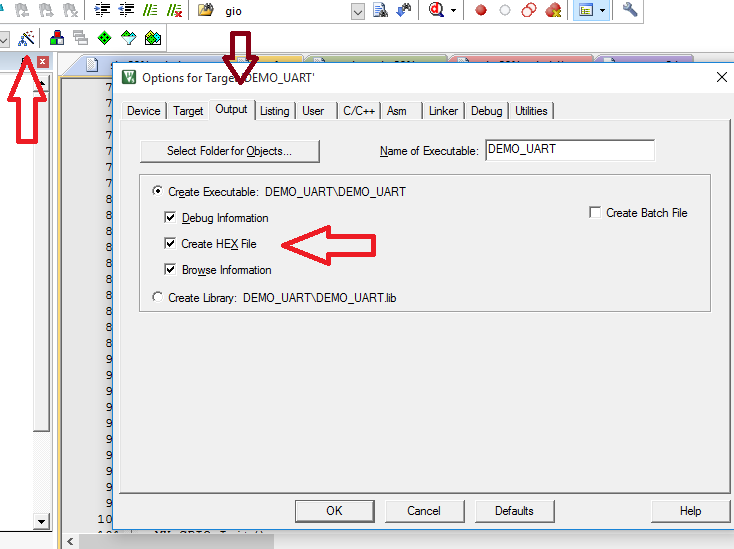
## Chương trình demo

Chương trình code demo mình sẽ cho gửi 1 đoạn văn bản lên mỗi 500 mili giây. Do phần vùng của chương trình này bắt đầu ở địa chỉ 0x08002000 nên ngay khi vào hàm main, chúng ta dời bảng vector ngắt đến địa chỉ này, đồng thười cũng bật isr lên luôn

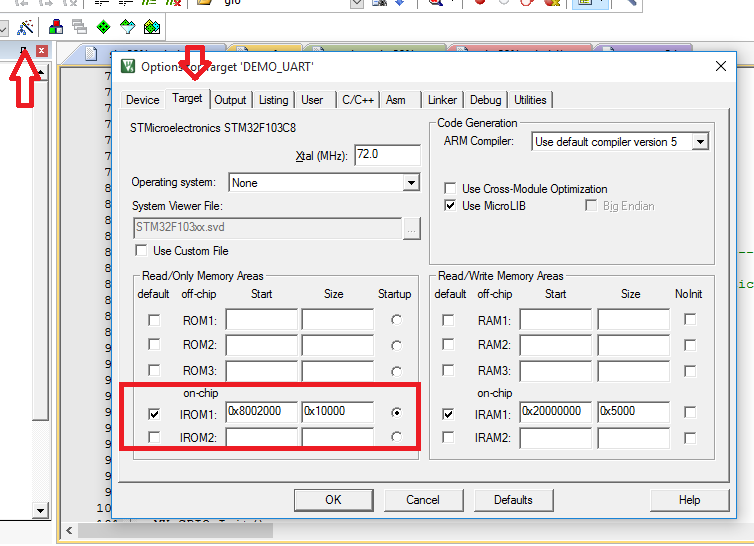


|  |  |
| --- | --- |
| 1  2  3  4  5  6 | int main(void)  {    /\* USER CODE BEGIN 1 \*/    SCB->VTOR = (uint32\_t)0x08002000;    \_\_enable\_irq();    /\* USER CODE END 1 \*/ |

Khi biên dịch các bạn tích chọn creat file hex trong build option



Trong thẻ tagret sửa địa chỉ bắt đầu thành 0x8002000



Sau đó ấn build để lấy file hex

Khi khởi động mạch, chip sẽ chạy chương trình boot trước, sau khi nạp code xuống thành công chíp sẽ nhảy sang chương trình chính ! Các bạn có thể thêm 1 nút nhấn để khi khởi động chương trình boot sẽ kiểm tra nút nhấn để quyết định xem nó chạy ở bootloader hay nhảy tới phân vùng code chính