TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA ĐIỆN TỬ - VIỄN THÔNG

BÁO CÁO ĐỒ ÁN

MÔN: THỰC HÀNH HỆ THỐNG NHÚNG

GIÁO VIÊN HƯỚNG DẪN:

TH.S HOÀNG ANH TUẤN

THÀNH VIÊN NHÓM 17:

Nguyễn Quốc Khánh	20200229
Hồ Viết Đức Huy	20200218
Lê Đình Huy	20200219

Thành phố Hồ Chí Minh, ngày 24 tháng 12 năm 2023

Lời cảm ơn

Lời đầu tiên, chúng em xin chân thành cảm ơn tới thầy ThS.Hoàng Anh Tuấn và công ty Bosch Global Software đã hỗ trợ board mạch để chúng em có cơ hội nghiên cứu sâu hơn, học tập nhiều hơn về lĩnh vực này. Đây chắc chắn là kiến thức sẽ đóng vai trò then chốt trong việc định hình sự nghiệp tương lai của chúng em nói chung và các bạn sinh viên trong ngành.

Đồ án này dựa trên nền tảng các kiến thức đã được học trên lớp và chúng em đã ứng dụng được vào trong đồ án. Mặc dù đã hoàn thiện các yêu cầu của đề, nhưng trực quan thì vẫn còn nhiều thiếu sót và không hoàn hảo để so với các dự án doanh nghiệp công ty. Đây cũng là điểm thiếu sót của nhóm em từ việc thiếu kinh nghiệm làm đồ án trong các doanh nghiệp và cần rèn luyện,mài dũa nhiều với đồ án doanh nghiệp để có thể tiến bộ và hoàn thiện hơn. Vì vậy, mong thầy có thể nhìn nhận và đưa ra những phản hồi tích cực để chúng em dần hoàn thiện và phát triển tiếp trong tương lai.

Một lần nữa, chúng em xin chân thành cảm ơn sự giảng dạy nhiệt huyết của thầy để chúng em có thể hoàn thành đồ án thành công.

MỤC LỤC

I/ Giới thiệu

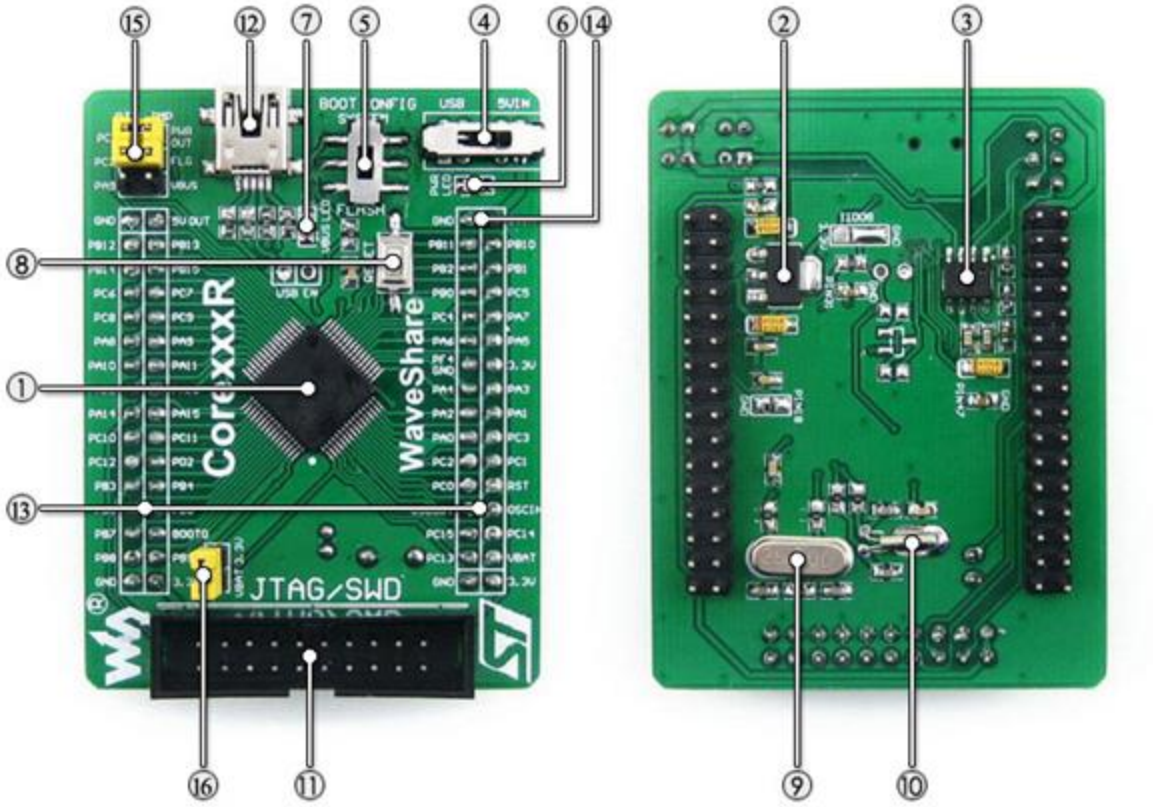
1. Giới thiệu về Open405R-C

Tổng quan:

Open405R-C là bo mạch phát triển **STM32** được thiết kế cho vi điều khiển **STM32F405RGT6,** bao gồm bo mạch chủ và bo mạch lõi **MCU core board Core405R**.

Open405R-C hỗ trợ mở rộng hơn thông qua khả năng tương thích với nhiều tùy chọn phụ kiện bảng khác nhau được thiết kế riêng cho các ứng dụng cụ thể. Thiết kế mô-đun và mở rộng ý tưởng để bắt đầu phát triển ứng dụng với bộ điều khiển dòng **STM32F2**, từ đó thúc đẩy sự dễ dàng và khả năng thích ứng trong quá trình phát triển.

Các thông số trên Core405R:



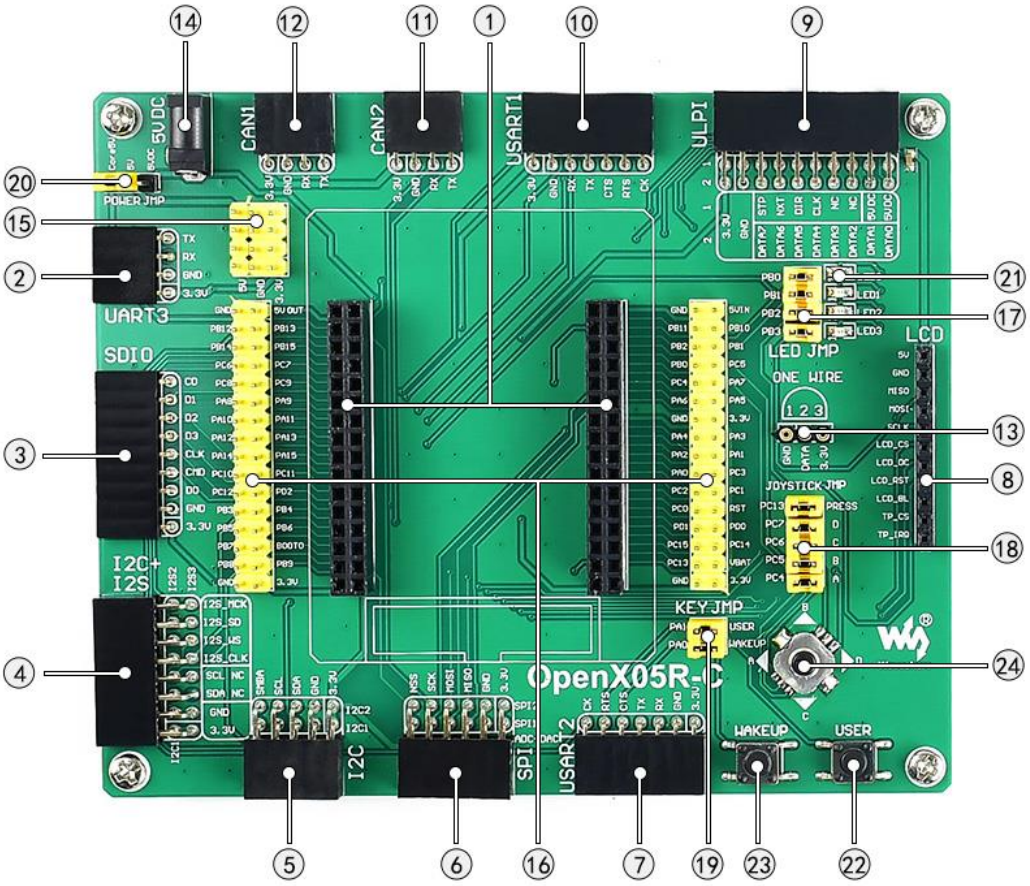
1. **STM32F405RGT6:** Vi điều khiển STM32 hiệu suất cao tính năng:
    - **Core:** Cortex-M4 32-bit RISC

- **Operating Frequency (Tần số hoạt động):** 168MHz, 210 DMIPS/1.25 DMIPS/MHz
- **Operating Voltage (Điện áp hoạt động):** 1.8V-3.6V
- **Package (Đóng gói):** LQFP64
- **Memories (Bộ nhớ):** 1MB Flash, 192+4kB SRAM
- **MCU communication Interfaces (Bộ giao tiếp Vi Điều Khiển):**
    - 3 x SPI, 4 x USART, 2 x UART, 2 x I2S, 3 x I2C, 1 x SDIO, 2 x CAN
    - 1 x USB 2.0 HS/FS device/host/OTG controller with dedicated DMA, on-chip full-speed PHY
    - 1 x USB HS ULPI (external PHY required)
- **AD & DA converters (Bộ chuyển đổi AD & DA):** 3 x AD (12-bit, 1µs, shares 16 channels); 2 x DA (12-bit)
- **Debugging/Programming (Gỡ lỗi/ Lập trình):** supports JTAG/SWD (serial wire debug) interfaces, supports IAP

2. **AMS1117-3.3:** ổn áp 3.3V
3. **MIC2075-2:** thiết bị quản lý nguồn USB tích hợp.
4. **Power supply switch,** cấp nguồn 5Vin hoặc kết nối USB
5. **Boot mode selection,** định cấu hình chân BOOT0
6. **Power indicator**
7. **VBUS LED**
8. **Reset button**
9. **8M crystal**
10. **32.768K crystal,** dành cho RTC bên trong tiêu chuẩn
11. **JTAG/SWD interface:** gỡ lỗi / lập trình
12. **USB connector,** sử dụng thiết lập giao tiếp USB giữa PC và phát triển STM32 board
13. **MCU pins expander,** VCC, GND và tất cả các chân I/O đều có thể mở rộng trên các đầu nối mở rộng để mở rộng hơn.
14. **5Vin pinheader,** nguồn điện 5V khi sử dụng USB HOST/OTG
15. **USB jumper**
    - Rút ngắn jumper khi sử dụng USB
    - Mở jumper để ngắt kết nối cổng I/O

16. **VBAT selection jumper**
    - Rút ngắn jumper để sử dụng nguồn điện hệ thống
    - Mở jumper để kết nối VBAT với nguồn ngoài, chẳng hạn như PIN

Các thông số trên Board?



1. **MCU core board connector:** Để dễ dàng kết nối với Core405R.
2. **UART3 interface:** Để dễ dàng kết nối với RS232, USB TO 232, etc.
3. **SDIO interface:** Để kết nối với mô-đun Micro SD,đặc điểm có tốc độ truy cập nhanh hơn nhiều so với SPI.
4. **I2S2/I2S3/I2C1:** Để kết nối với thiết bị ngoại vi I2S, chẳng hạn như : mô-đun Audio (âm thanh).
5. **I2C1/I2C2 interface:** Để kết nối dễ dàng với thiết bị ngoại vi I2C như là bộ mở rộng I/O (PCF8574), FRAM (FM24CLxx), vv.

6. **SPI1/SPI2 + AD/DA interface**
    - Để dễ dàng kết nối với các thiết bị ngoại vi SPI chẳng hạn như: DataFlash (AT45DBxx), SD card, mô-đun MP3, vv.
    - SPI1 có chức năng thay thế AD/DA ,hỗ trợ kết nối tốt đối vớimô-đun AD/DA.
7. **USART2 interface:** Dễ dàng kết nối với RS232, RS485, USB TO 232, vv.
8. **LCD interface:** Để kết nối với màn hình LCD cảm ứng.
9. **ULPI interface:** Để kết nối thiết bị ngoại vi USB tốc độ cao (the STM32F405R tích hợp bộ điều khiển USB HS mà không cần thiết bị PHY).
10. **UART1 interface:** Để dễ dàng kết nối với RS232, USB TO 232, vv.
11. **CAN2 interface:** Giao tiếp với các BOARD có thiết bị CAN một cách thuận tiện.
12. **CAN1 interface:** Giao tiếp với các BOARD có thiết bị CAN một cách thuận tiện.
13. **ONE-WIRE interface:** dễ dàng kết nối với các thiết bị ONE-WIRE (TO-92 package), chẳng hạn như là cảm biến đo nhiệt độ (DS18B20), số đăng kí điện tử (DS2401), vv.
14. **5V DC jack (giách cắm DC 5V)**
15. **5V/3.3 V power input/output:** Thường được sử dụng làm nguồn ra, cũng là đầu nối đất chung với board mạch người dùng khác.
16. **MCU pins connector:** VCC, GND, và tất cả các cổng I/O đều có thể truy cập được trên các đầu nối mở rộng .
17. **LEDs jumper**
    - Rút ngắn jumper để kết nối với I/Os mặc định được sử dụng trong code mẫu.
    - Mở jumper để kết nối với I/Os tùy chỉnh thông qua các dây jumper.
18. **Joystick jumper**
    - Rút ngắn jumper để kết nối với I/Os mặc định được sử dụng trong code mẫu.
    - Mở jumper để kết nối với I/Os tùy chỉnh thông qua các dây jumper.
19. **User key/Wake-Up button jumper**
    - Rút ngắn jumper để kết nối với I/Os mặc định được sử dụng trong code mẫu.
    - Mở jumper để kết nối với I/Os tùy chỉnh thông qua các dây jumper.
20. **5V power selection jumper:** Cấp nguồn từ Core 5V hoặc 5V DC
21. **LEDs:** Thuận tiện cho việc chỉ báo trạng thái I/O và/hoặc trạng thái chương trình chạy.
22. **User key:** convenient for I/O input and/or interact with running code

23. **Wake-Up button:** Chuyển vi điều khiển STM32 từ chế độ ngủ sang trạng thái hoạt động, cũng được sử dụng như là khóa người dùng thông thường.

24. **Joystick:** Thuận tiện cho đầu vào I/O (ứng với 5 vị trí).

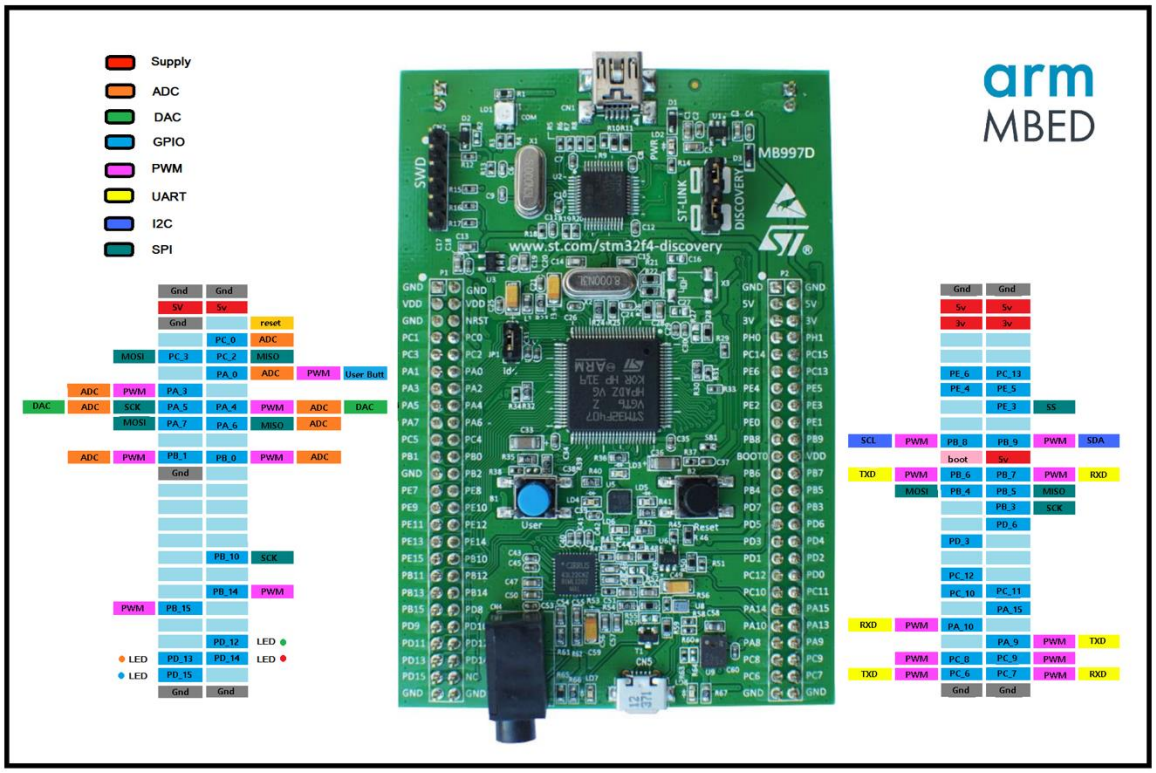
II/ Mô tả vi điều khiển:

1. Vi điều khiển STM32:

a. Vi điều khiển STM32F405xx:

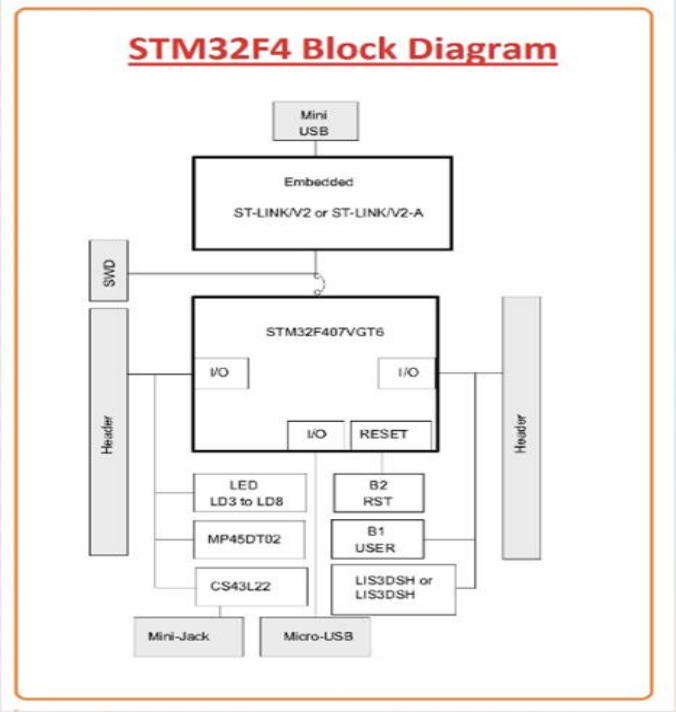
STM32F405xx thuộc họ Vi điều khiển ARM 32-bits có hiệu suất cao được tạo bởi STmicrocontroller. Họ **STM32F405xx** cung cấp các tính năng khác nhau nhưng tất cả các bộ điều khiển này đều sử dụng bộ vi điều khiển 32 bit. **STM32F405xx** cung cấp với tần số lên tới 168MHz. Họ **STM32F405xx** có khả năng kết hợp tính năng nhúng tốc độ cao có bộ nhớ flash 1Megabyte, SRAM có dung lượng lên tới 192 kilobyte, SRAM dự phòng lên tới 4 byte và các thiết bị ngoại vi kết nối với 2 kiến trúc APB buses, 3 kiến trúc AHB buses và 1 kiến trúc 32-bit AHB bus ma trận đa chiều. Họ **STM32F405xx** cung cấp đa dạng các thiết bị với nhiều gói khác nhau từ 64 chân tới 176 chân.Với những tính năng đa dạng của họ vi điều khiển **STM32F405xx** phù hợp, tương thích với phần lớn với các thiết bị ngoại vi nên thường được sử dụng rộng rãi.

Có rất nhiều đầu ra sơ đồ chân trên mô-đun này có thể được vận hành như đầu vào và đầu ra và cũng có thể được sử dụng để cung cấp lập trình cho bo mạch. Có rất nhiều ứng dụng được cung cấp bởi mô-đun này, các loại thiết bị liên lạc khác nhau có thể được gắn với bộ điều khiển này để liên kết các loại thiết bị điện tử khác nhau như máy dò, động cơ. v.v. Bo mạch này bao gồm nhiều thiết bị khác được sử dụng để liên lạc và giao tiếp với các dự án khác nhau mà không cần sử dụng thiết bị bên ngoài. Mô-đun này còn đi kèm với một số thành phần khác như bộ chuyển đổi kỹ thuật số sang analog, bộ chuyển đổi analog sang kỹ thuật số, cổng âm thanh, tất cả các thành phần này tạo nên một bo mạch hoàn chỉnh.

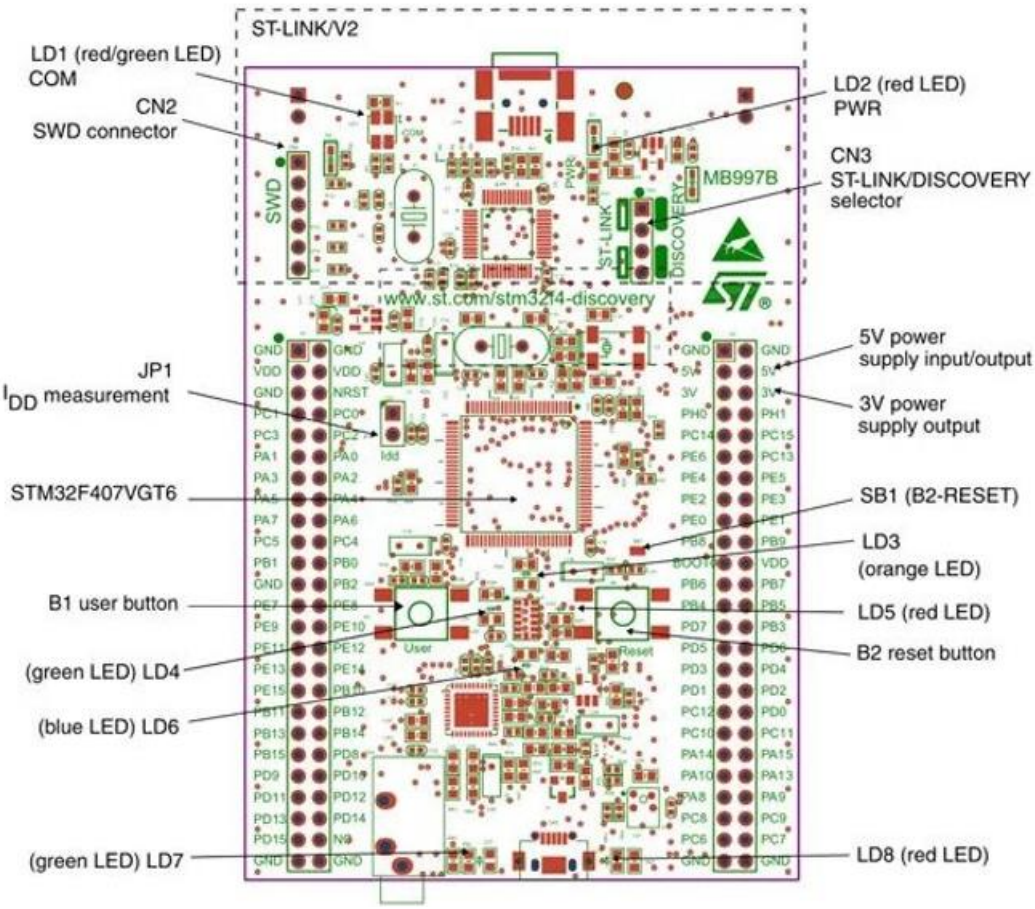


b. Đặc điểm STM32F405xx:

- Đây là một số tính năng quan trọng của STM32F4 được mô tả chi tiết.
- Bao gồm bộ xử lý trung tâm Cortex 32-bit của ARM với tần số 180Mhz
- Bao gồm bộ RAM có dung lượng tới 190kB.
- Ngoài ra, còn có bộ nhớ flash 1MB.
- Điện áp hoạt động từ 5V.
- Không có cổng debuging .
- Nguồn của mô-đun có thể được cấp qua cổng USB.
- Có tổng cộng 4 LED trên board với các màu cam, lục, đỏ và dương.
- Có 2 nút nhấn trên board, đầu tiên là reset và còn lại là dành cho người dùng.



c. Sơ đồ chân STM32F4xx:



❖ Sơ đồ chân nguồn

- Có rất nhiều sơ đồ chân trên bo mạch này để cung cấp điện áp 5 volt cho các thành phần khác nhau được kết nối.
- Chân này được sử dụng làm chân nguồn đầu vào trong nhóm P1 và P2 được coi là chân thứ ba và chân thứ tư trong nhóm P1 và chân thứ ba và thứ tư trong nhóm P2 được sử dụng làm chân ra đầu vào.
- Sơ đồ chân cung cấp nguồn điện ở đầu ra trong phạm vi 3 volt và 5 volt.
- Các chân cung cấp 5 volt là chân 3 và 4 trong nhóm P2 và các chân cung cấp 3 volt là các chân 5, 6, 15 và 16 trong nhóm P2.

❖ Sơ đồ chân GND

- Trong nhóm P1, các chân được sử dụng làm điểm nối đất là 1, 2, 5,23, 49 và 50. và các chân được sử dụng làm điểm nối đất trong nhóm P2 là 1, 2, 49 và 50.
- Sơ đồ chân dao động
- Không có xung đồng hồ tinh thể bên trong tồn tại trong mô-đun này. Có 4 sơ đồ chân xung đồng hồ bên ngoài, trong đó có 2 hoạt động cho tinh thể 32 kilohertz và 2 hoạt động cho tinh thể tần số giá trị lớn.
- Các chân ra dùng làm bộ tạo dao động nằm trong nhóm 2 và được đặt tên là.
- GPIO7, GPIO, GPIO GPIO10.

❖ Sơ đồ chân GPIO

- Có sáu cổng tồn tại trong bảng này được ký hiệu là A, B, C, D, E và H. Tất cả các cổng này đều có điện trở kéo lên bên trong và hoạt động như đầu vào và đầu ra.
- Các chân thoát này trong nhóm P1 là GPIO7 đến GPIO22 và GPIO24 đến GPIO47 và các chân này trong nhóm P2 có phạm vi từ GPIO7 đến GPIO21.

→ **Ứng dụng của STM32F4:**

- Đây là một số ứng dụng quan trọng của các mô-đun này được mô tả chi tiết ở đây.
- Nó được sử dụng trong các loại dự án nhúng và robot khác nhau.
- Vì nó cung cấp nhiều chức năng nên nó được sử dụng trong công nghiệp để điều khiển máy móc.
- Nó cũng được sử dụng trong các dự án truyền âm thanh. GPIO Pinouts.

2. HAL driver files

Cấu trúc dữ liệu HAL

Mỗi trình điều khiển HAL có thể chứa các cấu trúc dữ liệu sau:

- ❖ Cấu trúc tay cầm ngoại vi
- ❖ Cấu trúc khởi tạo và cấu hình
- ❖ Cấu trúc quy trình cụ thể.
- ❖ Cấu trúc xử lý ngoại vi:

Các API có kiến trúc đa phiên bản chung theo mô-đun cho phép làm việc đồng thời với nhiều phiên bản IP.

PPP_HandleTypeDef *xử lý là cấu trúc chính được triển khai trong trình điều khiển HAL. Nó xử lý cấu hình ngoại vi/mô-đun và đăng ký cũng như nhúng tất cả các cấu trúc và biến cần thiết để tuân theo luồng thiết bị ngoại vi.

- Tay cầm ngoại vi được sử dụng cho các mục đích sau:
- Hỗ trợ đa phiên bản: mỗi phiên bản ngoại vi/mô-đun có bộ điều khiển riêng. Kết quả là tài nguyên cá thể là độc lập.
- Giao tiếp liên lạc quy trình ngoại vi: tay cầm được sử dụng để quản lý tài nguyên dữ liệu được chia sẻ giữa các quy trình quy trình.
- Ví dụ: con trỏ toàn cục, bộ điều khiển DMA, máy trạng thái.
- Lưu trữ : tay cầm này cũng được sử dụng để quản lý các biến toàn cục trong trình điều khiển HAL nhất định.

Cấu trúc khởi tạo và cấu hình:

- ✓ Các cấu trúc này được xác định trong tệp tiêu đề trình điều khiển chung khi nó chung cho tất cả các số bộ phận. Khi chúng có thể thay đổi từ số bộ phận này sang số bộ phận khác, cấu trúc được xác định trong tệp tiêu đề mở rộng cho mỗi số bộ phận.
- ✓ Cấu trúc cấu hình được sử dụng để khởi tạo các mô-đun phụ hoặc phiên bản phụ.

Cấu trúc quy trình cụ thể:

- ✓ Các cấu trúc quy trình cụ thể được sử dụng cho quy trình cụ thể (các API chung). Chúng được xác định trong tệp tiêu đề trình điều khiển chung.

III/ LAB:

LAB 01 + 02:

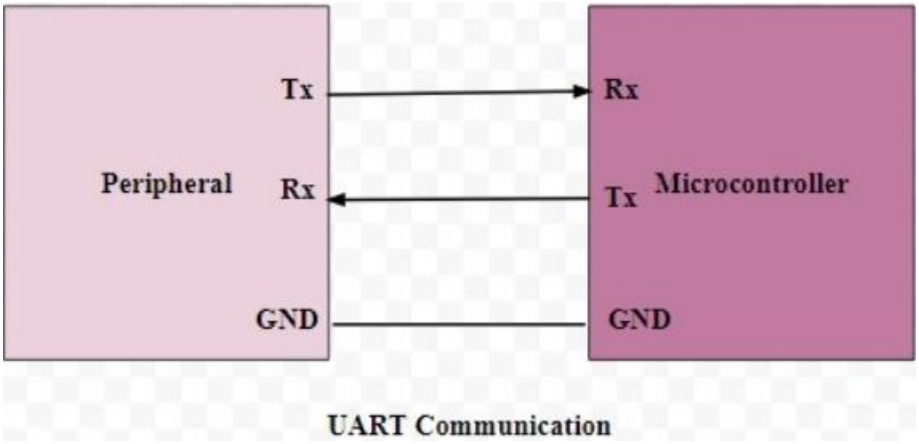
- **Yêu cầu:**
  - ✓ **Cấu hình giao thức UART truyền dữ liệu nhạc bất gì trên máy tính xuống STM.**
  - ✓ **Cấu hình USB host ghi dữ liệu file nhạc đó xuống USB storage.**

Bài làm:

1.1 Khái quát:

a. Giao thức UART là gì?

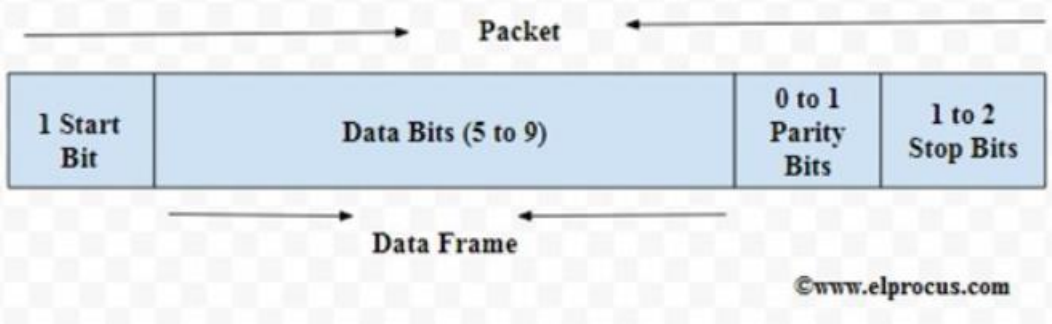
UART (Universal Asynchronous Reciver/Transmister**)** là chuẩn giao tiếp bất đồng bộ giữa vi điều khiển và các thiết bị ngoại vi. **UART** cung cấp một phạm vi tốc độ truyền rất rộng bằng cách sử dụng bộ tạo tốc độ truyền phân đoạn



UART Communication

Cách thức hoạt động của UART

UART là giao thức truyền thông không đồng bộ(không có xung Clock, các thiết bị có thể hiểu được nhau nếu các Setting giống nhau). **UART** là truyền thông song công(Full duplex) nghĩa là tại một thời điểm có thể truyền và nhận đồng thời. Trong đó quan trọng nhất là *Baund rate* (tốc độ *Baund*) là khoảng thời gian dành cho 1 bit được truyền. Phải được cài đặt giống nhau ở gửi và nhận.



Start – Bit

Start-bit còn được gọi là bit đồng bộ hóa được đặt trước dữ liệu thực tế. Khi đang ở chế độ "nhàn rỗi" thì đường tín hiệu được đưa lên mức cao (1). Để bắt đầu truyền dữ liệu, đường dữ liệu sẽ được kéo từ mức điện áp cao (1) xuống mức điện áp thấp (0), Báo cho bên nhận biết là sắp truyền dữ liệu, đây chính là bit Start.

Stop – Bit

Bit dừng được đặt ở phần cuối của gói dữ liệu. Thông thường, bit này dài 2 bit nhưng thường chỉ sử dụng 1 bit. Sau khi truyền xong dữ liệu, thì đường dữ liệu sẽ được giữ ở mức cao tương đương 1 hoặc 2 bit.

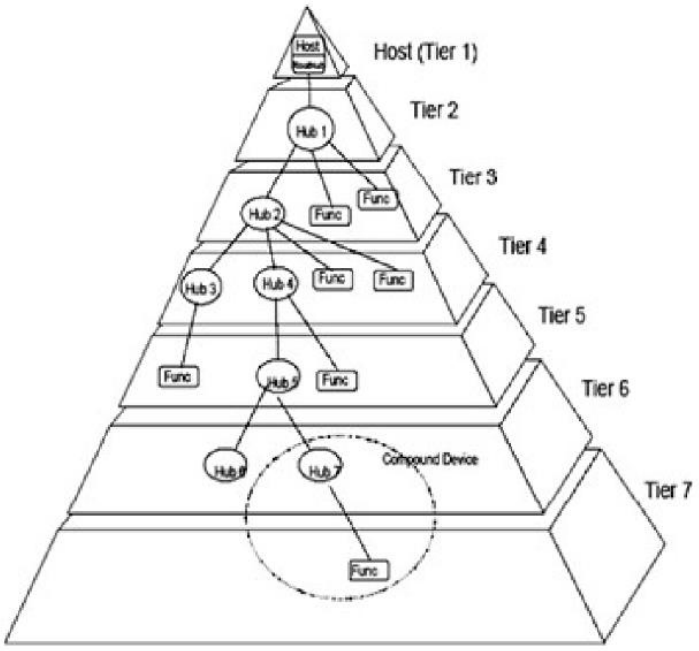
Partity Bit

Bit chẵn lẻ cho phép người nhận đảm bảo liệu dữ liệu được thu thập có đúng hay không. Đây là một hệ thống kiểm tra lỗi cấp thấp. Trên thực tế, bit này không được sử dụng rộng rãi nên không bắt buộc.

Data frame

Các bit dữ liệu bao gồm dữ liệu thực được truyền từ người gửi đến người nhận. Độ dài khung dữ liệu có thể nằm trong khoảng 5 & 8. Nếu bit chẵn lẻ không được sử dụng thì chiều dài khung dữ liệu có thể dài 9 bit. Bit LSB sẽ được truyền trước.

b. Kiến trúc USB:



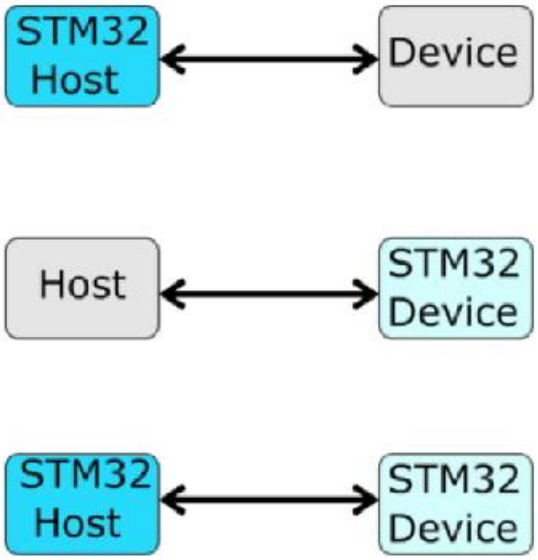
Một hệ thống USB được mô tả bởi 3 định nghĩa: USB host, USB device, USB connect.

Kiến trúc bus: về mặt vật lý, kiến trúc Bus USB là một tầng sao, với HOST là trung tâm. Mỗi tia là kết nối giữa HOST với HUB, HOST với DEVICE hoặc DEVICE với HUB. Với 7 bit địa chỉ, một HOST có thể quản lý 127 thiết bị trong mạng lưới của nó.

Mỗi vi điều khiển STM32 bao gồm cả USB có thể hỗ trợ:

- Thiết bị ở tốc độ FS.
- OTG (kép: thiết bị và máy chủ) trong tốc độ FS.
- OTG theo tốc độ HS.

Hệ thống USB cơ bản có thể được thiết lập bởi máy chủ và thiết vị đước gắn bằng cáp USB

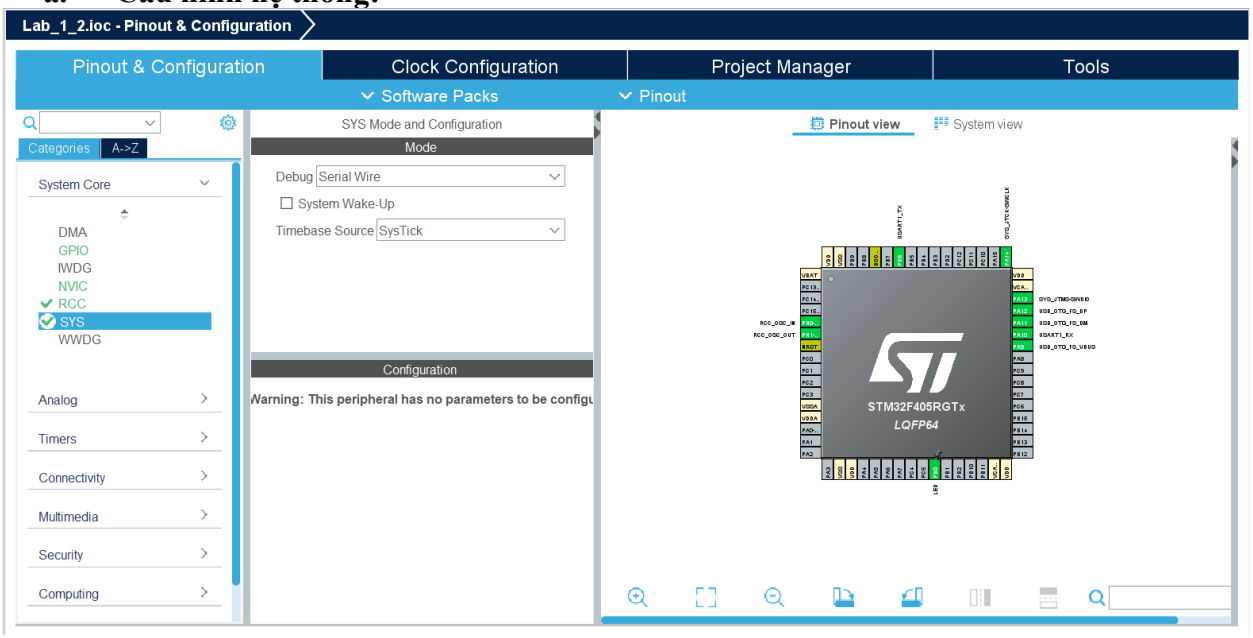


1.2 Ý tưởng:

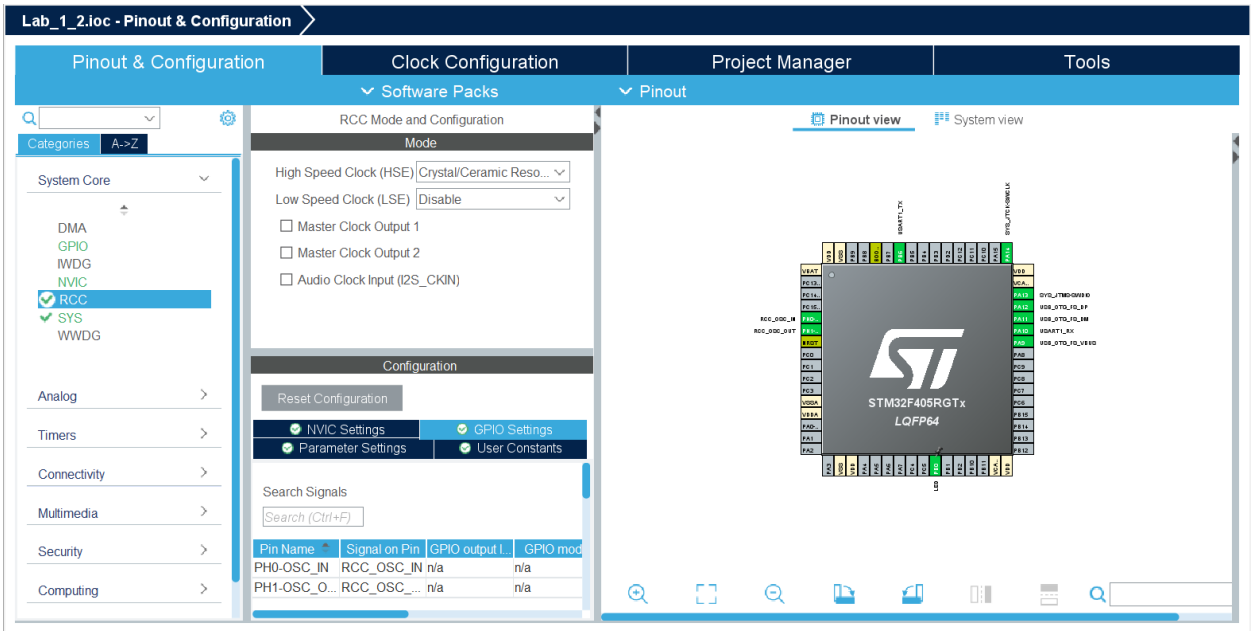
- ✓ Cấu hình UART và USB để truyền dữ liệu từ máy tính xuống board STM32 và lưu trữ dữ liệu trong USB.
- ✓ Chạy file python *music.py* truyền dữ liệu nhạc sẽ được truyền đi *(*dấu hiệu nhận biết: led thay đổi trạng thái: từ tắt sang sáng).
- ✓ Dữ liệu sẽ được lưu vào biến *buffer*
- ✓ Cuối cùng, board sẽ tạo file trong usb từ dữ liệu trên và lưu dữ liệu vào trong đó.

1.3 Cấu hình UART và USB trên STM32:

a. Cấu hình hệ thống:

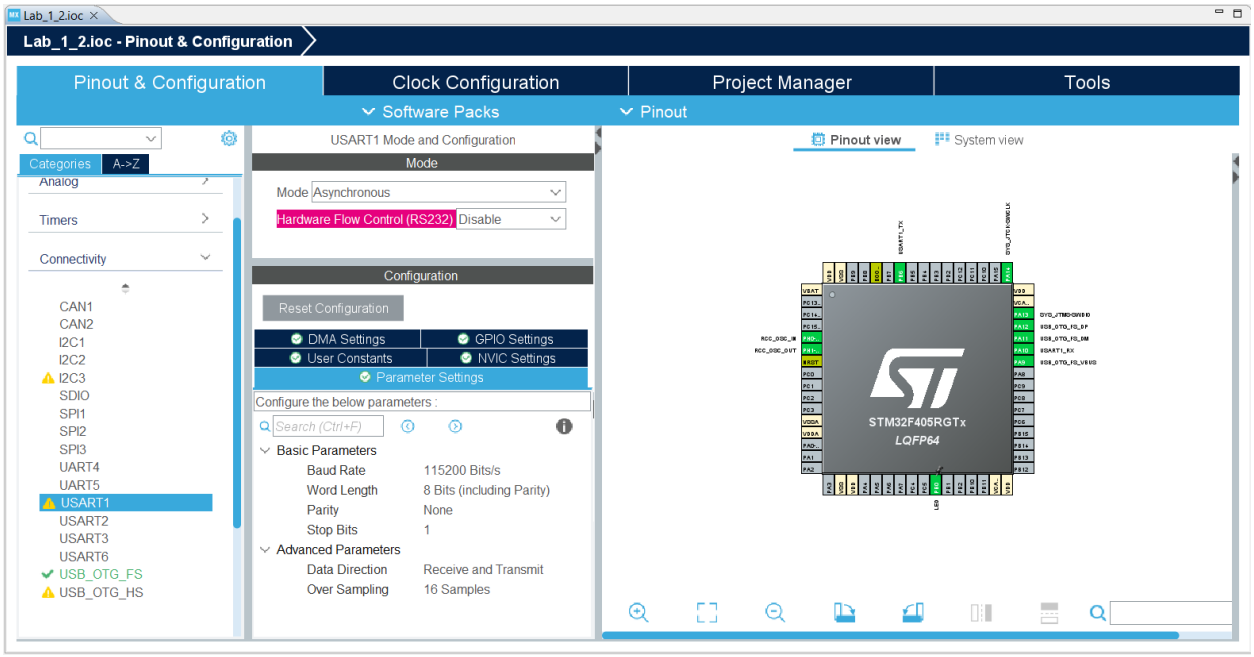


Cấu hình biên dịch code lên board theo chuẩn dây Serial.



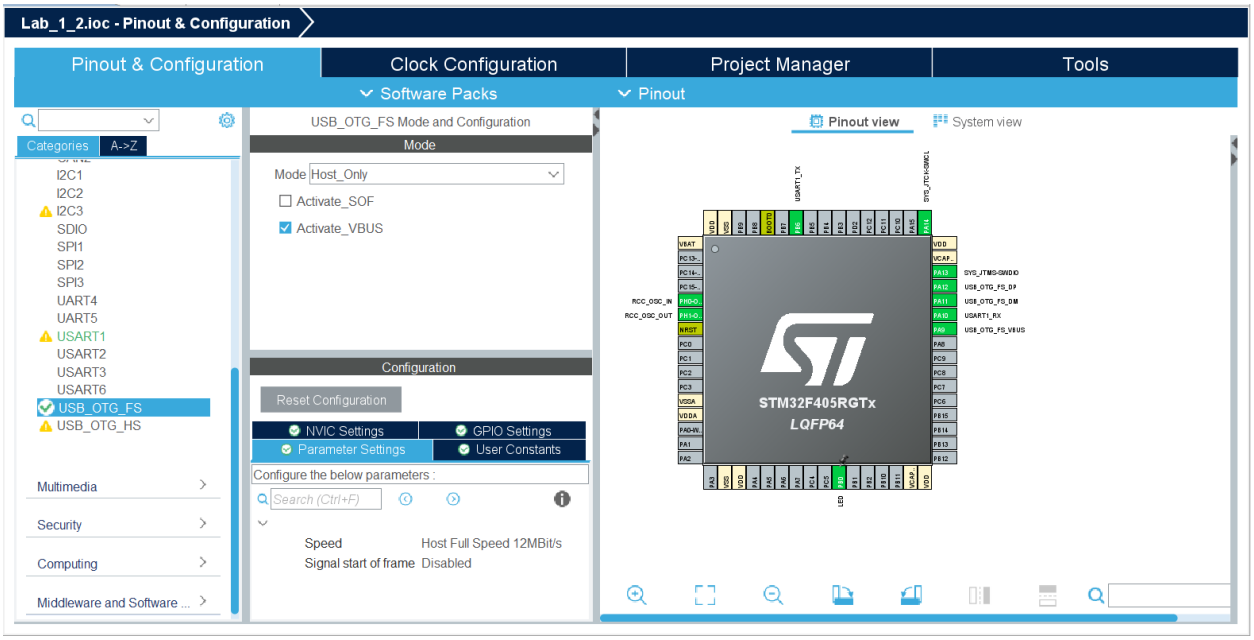
Sử dụng Crystal/Ceramic Resonator.

b. Cấu hình UART:

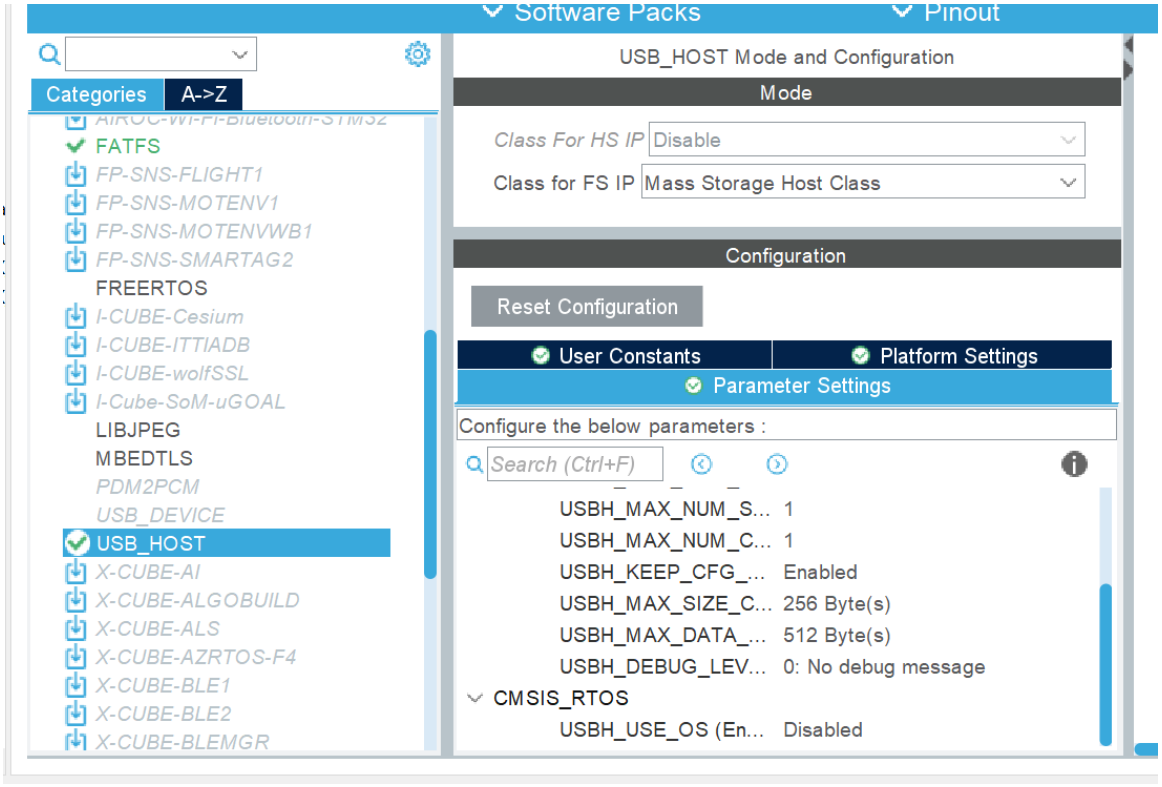


- Chế độ : bất đồng bộ
- Tốc độ baud:115200 bit/s
- Độ dài khung dữ liệu (Word Length): 8 bit
- Parity bit: Không
- Stop bits : 1 bit
- Chế độ: nhận và truyền dữ liệu.

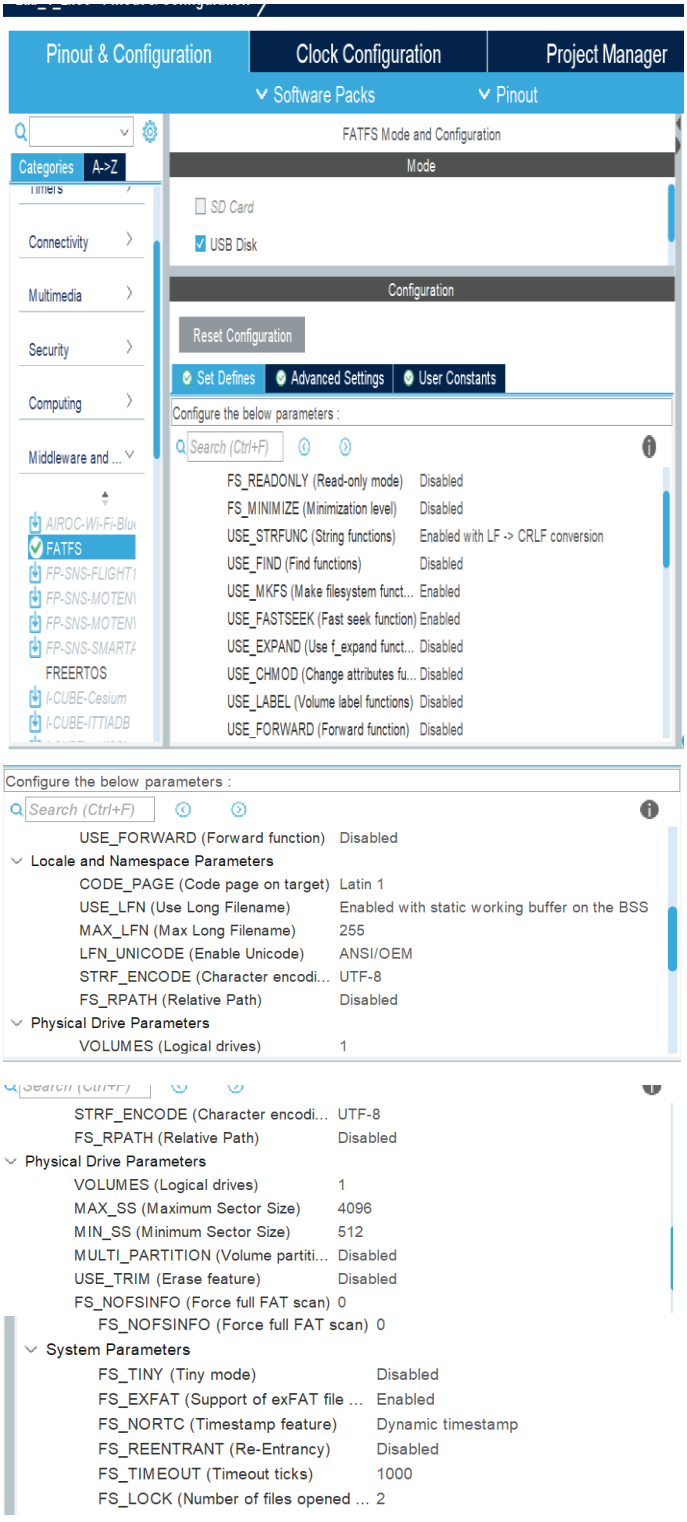
c. Cấu hình USB:



Cấu hình USB_OTG_FS,chọn chế độ Host_only và tick vào Activate_VBUS.

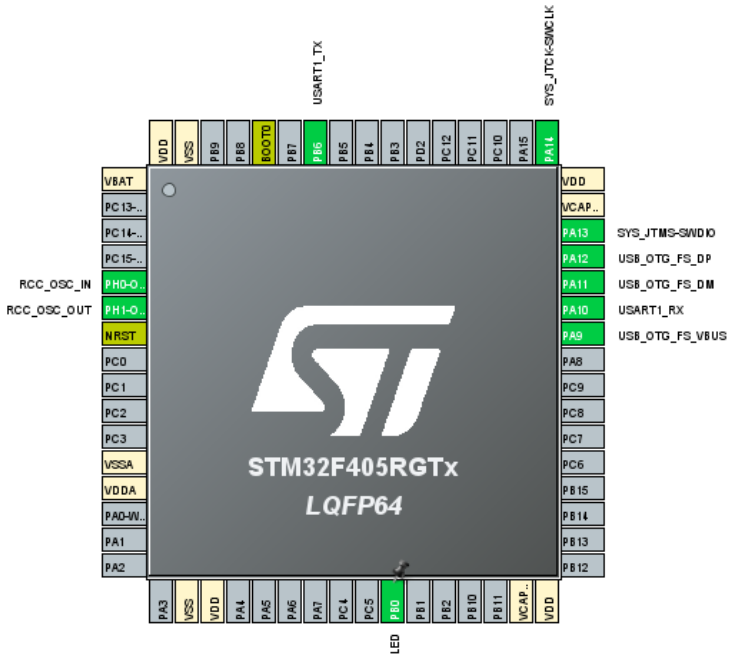


Cấu hình USB_HOST



Cấu hình FATFS

d. Sơ đồ chân:



- UART: Sử dụng chân PB6(USART1_TX) và PA10(USART1_RX).
- LED: Sử dụng chân PB0 cho LED.
- USB: Sử dụng chân PA9(USB_OTG_FS_VBUS), PA11(USB_OTG_FS_DM), PA12(USB_OTG_FS_DB).

1.4 Code và giải thích:

```
void Mount_USB (void)
{
	fresult = f_mount(&USBHFatFS, USBHPath, 1);
	if (fresult != FR_OK) Send_Uart ("ERROR!!! in mounting USB ...\n\n");
	else Send_Uart("USB mounted successfully...\n");
}
```

Giải thích:

→ Hàm Mount_USB này thường được gọi đầu tiên khi bắt đầu sử dụng thiết bị USB, để đảm bảo rằng hệ thống tệp tin của USB đã được kết nối và có thể được truy cập để thực hiện các thao tác đọc, ghi và quản lý tệp tin và thư mục.

- _mount: Hàm f_mount là một hàm trong thư viện FatFs, được sử dụng để gắn kết một hệ thống tệp tin vào một đường dẫn (path) cụ thể, hàm f_mount được gọi với tham số &USBHFatFS (con trỏ đến cấu trúc File System cho thiết bị USB), USBHPath (đường dẫn được sử dụng cho thiết bị USB), và 1 (được sử dụng để gắn kết hệ thống tệp tin).
- Kiểm tra kết quả: Sau khi gọi f_mount, hàm kiểm tra giá trị trả về fresult. Nếu fresult không bằng FR_OK (không có lỗi), hàm sẽ gửi thông báo lỗi qua UART thông qua hàm Send_Uart. Nếu fresult bằng FR_OK, điều này ngụ ý rằng quá trình gắn kết hệ thống tệp

tin từ thiết bị USB đã thành công. Hàm sẽ gửi một thông báo qua UART cho biết rằng thiết bị USB đã được gắn kết thành công.

```
void Check_USB_Details (void)
{
    /* Check free space */
    f_getfree("", &fre_clust, &pUSBHFatFS);

    total = (uint32_t)((pUSBHFatFS->n_fatent - 2) * pUSBHFatFS->csize *
0.5);
    char *buf = malloc(30*sizeof(char));
    sprintf (buf, "USB  Total Size: \t%lu\n",total);
    Send_Uart(buf);
    free(buf);
    free_space = (uint32_t)(fre_clust * pUSBHFatFS->csize * 0.5);
    buf = malloc(30*sizeof(char));
    sprintf (buf, "USB Free Space: \t%lu\n",free_space);
    Send_Uart(buf);
    free(buf);
}
```

Giải thích:

→ Hàm Check_USB_Details này cung cấp thông tin về dung lượng tổng và dung lượng trống của thiết bị USB thông qua UART, giúp người dùng biết được thông tin về không gian trống còn lại và dung lượng sử dụng trên thiết bị USB.

- _getfree: Hàm f_getfree trong thư viện FatFs được sử dụng để lấy thông tin về không gian trống trên thiết bị lưu trữ (thông qua tham số "" cho đường dẫn). Kết quả của hàm này được lưu trữ trong các biến fre_clust (số cluster trống) và pUSBHFatFS (con trỏ đến cấu trúc File System cho thiết bị USB).
- Tính toán thông tin không gian: Sau khi lấy thông tin về không gian trống, hàm tiếp tục tính toán thông tin về dung lượng tổng và dung lượng trống của thiết bị USB. total: Tổng dung lượng của thiết bị USB được tính dựa trên số lượng cluster trên thiết bị. free_space: Dung lượng trống của thiết bị USB được tính dựa trên số lượng cluster trống.
- Gửi thông tin qua UART: Hàm sử dụng sprintf để định dạng thông tin về tổng dung lượng (total) và dung lượng trống (free_space) vào một chuỗi ký tự và lưu vào biến buf. Sau đó, thông tin này được gửi đi qua UART bằng cách sử dụng hàm Send_Uart. Cuối cùng, sau khi gửi thông tin, bộ nhớ đã được cấp phát cho biến buf sẽ được giải phóng bằng lệnh free(buf).

```
FRESULT Scan_USB (char* pat)
{
    DIR dir;
    UINT i;
    char *path = malloc(20*sizeof (char));
    sprintf (path, "%s",pat);

    fresult = f_opendir(&dir, path);                       /* Open the
directory */
```

```
    if (fresult == FR_OK)
    {
        for (;;)
        {
            fresult = f_readdir(&dir, &USBHfno);                    /* Read a
directory item */
            if (fresult != FR_OK || USBHfno.fname[0] == 0) break;  /* Break
on error or end of dir */
            if (USBHfno.fattrib & AM_DIR)     /* It is a directory */
            {
                  if (!(strcmp ("SYSTEM~1", USBHfno.fname))) continue;
                  if (!(strcmp("System Volume Information", USBHfno.fname)))
continue;
                  char *buf = malloc(30*sizeof(char));
                  sprintf (buf, "Dir: %s\r\n", USBHfno.fname);
                  Send_Uart(buf);
                  free(buf);
                i = strlen(path);
                sprintf(&path[i], "/%s", USBHfno.fname);
                fresult = Scan_USB(path);                     /* Enter the
directory */
                if (fresult != FR_OK) break;
                path[i] = 0;
            }
            else
            {   /* It is a file. */
               char *buf = malloc(30*sizeof(char));
               sprintf(buf,"File: %s/%s\n", path, USBHfno.fname);
               Send_Uart(buf);
               free(buf);
            }
        }
        f_closedir(&dir);
    }
    free(path);
    return fresult;
}
```

Giải thích:

→ Hàm Scan_USB này giúp bạn đọc thông tin về tất cả các thư mục và tệp tin có trên thiết bị USB, và gửi thông tin này qua giao diện UART để theo dõi các thành phần trên thiết bị lưu trữ USB của bạn.

- Khởi tạo và mở thư mục: Hàm bắt đầu bằng việc khởi tạo một biến DIR để đại diện cho thư mục (DIR dir). Sau đó, nó cấp phát bộ nhớ cho biến path và gán giá trị đường dẫn được truyền vào hàm.
- Mở thư mục và quét: Sử dụng hàm f_opendir để mở thư mục với đường dẫn đã được truyền vào. Nếu mở thư mục thành công (fresult == FR_OK), hàm bắt đầu quét các tệp và thư mục bên trong thư mục đã mở. Vòng lặp for() sử dụng hàm f_readdir để đọc các mục trong thư mục đã mở. Nếu không còn mục nào hoặc xảy ra lỗi, vòng lặp sẽ thoát.
- Xử lý thư mục và tệp: Nếu mục đọc được là một thư mục (USBHfno.fattrib & AM_DIR), hàm sẽ gửi thông tin về thư mục đó qua UART và tiến hành đệ quy gọi lại chính nó

(Scan_USB) với đường dẫn mới (đường dẫn hiện tại cộng với tên thư mục đọc được). Nếu mục đọc được là một tệp tin, hàm cũng gửi thông tin về tệp tin đó qua UART.

- Đóng thư mục và giải phóng bộ nhớ: Sau khi hoàn thành quá trình quét, thư mục sẽ được đóng (f_closedir(&dir)) và bộ nhớ đã cấp phát cho biến path sẽ được giải phóng.

```
FRESULT Create_Dir (char *name)
{
    fresult = f_mkdir(name);
    if (fresult == FR_OK)
    {
      char *buf = malloc(100*sizeof(char));
      sprintf (buf, "*%s* has been created successfully\n\n", name);
      Send_Uart (buf);
      free(buf);
    }
    else
    {
      char *buf = malloc(100*sizeof(char));
      sprintf (buf, "ERROR No. %d in creating directory *%s*\n\n",
fresult,name);
      Send_Uart(buf);
      free(buf);
    }
    return fresult;
}
```

Giải thích:

→ Hàm Create_Dir này giúp bạn tạo thư mục mới trên thiết bị lưu trữ, và sau đó thông báo kết quả của quá trình tạo thư mục qua giao diện UART để theo dõi quá trình tạo thư mục trên thiết bị của bạn.

- f_mkdir: Hàm f_mkdir là một hàm từ thư viện FatFs, được sử dụng để tạo một thư mục mới với tên được chỉ định bởi đường dẫn name.
- Kiểm tra kết quả tạo thư mục: Hàm kiểm tra kết quả của việc tạo thư mục bằng cách kiểm tra giá trị trả về fresult. Nếu fresult bằng FR_OK (không có lỗi), nghĩa là thư mục đã được tạo thành công. Trong trường hợp này, hàm sẽ cấp phát bộ nhớ cho một chuỗi thông báo thành công, thông báo rằng thư mục đã được tạo thành công với tên đã chỉ định và gửi thông báo này qua giao diện UART. Nếu fresult không bằng FR_OK, nghia là xảy ra lỗi trong quá trình tạo thư mục, sau đó sẽ in ra một chuỗi thông báo lỗi, thông báo về mã lỗi cụ thể và tên thư mục mà quá trình tạo thư mục gặp sự cố và gửi thông báo lỗi này qua UART.
- Giải phóng bộ nhớ và trả về kết quả: Cuối cùng, sau khi gửi thông báo thành công hoặc thông báo lỗi, hàm giải phóng bộ nhớ đã cấp phát cho chuỗi thông báo và trả về kết quả của quá trình tạo thư mục.

```
FRESULT Create_File (char *name)
```

```
{
	fresult = f_stat (name, &USBHfno);
	if (fresult == FR_OK)
	{
		char *buf = malloc(100*sizeof(char));
		sprintf (buf, "ERROR!!! *%s* already exists!!!!\n use
Update_File \n\n",name);
		Send_Uart(buf);
		free(buf);
	    return fresult;
	}
	else
	{
		fresult = f_open(&USBHFile, name,
FA_CREATE_ALWAYS|FA_READ|FA_WRITE);
		if (fresult != FR_OK)
		{
			char *buf = malloc(100*sizeof(char));
			sprintf (buf, "ERROR!!! No. %d in creating file *%s*\n\n",
fresult, name);
			Send_Uart(buf);
			free(buf);
		    return fresult;
		}
		else
		{
			char *buf = malloc(100*sizeof(char));
			sprintf (buf, "*%s* created successfully\n Now use
Write_File to write data\n",name);
			Send_Uart(buf);
			free(buf);
		}

		fresult = f_close(&USBHFile);
		if (fresult != FR_OK)
		{
			char *buf = malloc(100*sizeof(char));
			sprintf (buf, "ERROR No. %d in closing file *%s*\n\n",
fresult, name);
			Send_Uart(buf);
			free(buf);
		}
		else
		{
			char *buf = malloc(100*sizeof(char));
			sprintf (buf, "File *%s* CLOSED successfully\n\n", name);
			Send_Uart(buf);
			free(buf);
		}
	}
    return fresult;
}
```

Giải thích:

→ Hàm Create_File trong đoạn mã được sử dụng để tạo một tệp tin mới trên thiết bị lưu trữ như USB, thẻ nhớ hoặc các thiết bị lưu trữ khác được hỗ trợ, sử dụng thư viện FatFs.

- Sử dụng hàm f_stat để kiểm tra xem tệp có tồn tại không.Nếu tệp đã tồn tại (f_stat trả về FR_OK), hàm sẽ cấp phát bộ nhớ cho một chuỗi thông báo lỗi thông báo rằng tệp đã tồn tại và hướng dẫn sử dụng hàm Update_File để cập nhật tệp, sau đó gửi thông báo lỗi qua UART và trả về kết quả.
- Tạo tệp mới: Nếu tệp chưa tồn tại, hàm sẽ tiếp tục bằng cách mở tệp để tạo mới (f_open với cờ FA_CREATE_ALWAYS). Nếu việc tạo tệp không thành công (f_open trả về giá trị khác FR_OK), hàm sẽ cấp phát bộ nhớ cho một chuỗi thông báo lỗi, gửi thông báo lỗi này qua UART và trả về kết quả lỗi.
- Thông báo về việc tạo tệp: Nếu tạo tệp thành công, hàm sẽ cấp phát bộ nhớ cho một chuỗi thông báo thành công, thông báo rằng tệp đã được tạo thành công và hướng dẫn sử dụng hàm Write_File để ghi dữ liệu vào tệp, sau đó gửi thông báo này qua UART.
- Đóng tệp và xử lý kết quả: Tiếp theo, hàm sẽ đóng tệp (f_close) sau khi đã tạo xong hoặc nếu quá trình tạo tệp không thành công. Nếu việc đóng tệp không thành công (f_close trả về giá trị khác FR_OK), hàm sẽ cấp phát bộ nhớ cho một chuỗi thông báo lỗi, gửi thông báo lỗi này qua UART và trả về kết quả lỗi.
- Giải phóng bộ nhớ và trả về kết quả: Cuối cùng, hàm sẽ giải phóng bộ nhớ không cần thiết và trả về kết quả của quá trình tạo tệp (có thể là FR_OK nếu không có lỗi, hoặc mã lỗi nếu có lỗi xảy ra). Đóng thư mục và giải phóng bộ nhớ: Sau khi hoàn thành quá trình quét, thư mục sẽ được đóng (f_closedir(&dir)) và bộ nhớ đã cấp phát cho biến path sẽ được giải phóng.

```
FRESULT Write_File(char *name, uint8_t *data, uint32_t size)
{
    /**** check whether the file exists or not ****/
    fresult = f_stat(name, &USBHfno);
    if (fresult != FR_OK)
    {
        char *buf = malloc(100 * sizeof(char));
        sprintf(buf, "ERROR!!! %s does not exist\n\n", name);
        Send_Uart(buf);
        free(buf);
        return fresult;
    }
    else
    {
        /* Create a file with read write access and open it */
        fresult = f_open(&USBHFile, name, FA_OPEN_EXISTING | FA_WRITE);
        if (fresult != FR_OK)
        {
            char *buf = malloc(100 * sizeof(char));
            sprintf(buf, "ERROR!!! No. %d in opening file *%s*\n\n", fresult,
name);
            Send_Uart(buf);
            free(buf);
            return fresult;
        }
        else
        {
            char *buf = malloc(100 * sizeof(char));
            sprintf(buf, "Opening file-->  %s  To WRITE data in it\n", name);
            Send_Uart(buf);
```

```
            free(buf);

            fresult = f_write(&USBHFile, data, size, &bw);
            if (fresult != FR_OK)
            {
                char *buf = malloc(100 * sizeof(char));
                sprintf(buf, "ERROR!!! No. %d while writing to the FILE
*%s*\n\n", fresult, name);
                Send_Uart(buf);
                free(buf);
            }

            /* Close file */
            fresult = f_close(&USBHFile);
            if (fresult != FR_OK)
            {
                char *buf = malloc(100 * sizeof(char));
                sprintf(buf, "ERROR!!! No. %d in closing file %s after writing
it\n\n", fresult, name);
                Send_Uart(buf);
                free(buf);
            }
            else
            {
                char *buf = malloc(100 * sizeof(char));
                sprintf(buf,    "File    %s    is    WRITTEN    and    CLOSED
successfully\n\n", name);
                Send_Uart(buf);
                free(buf);
            }
        }
        return fresult;
    }
}
```

Giải thích:

→ Hàm này làm việc với thư viện FatFs (file system) để kiểm tra, mở, ghi dữ liệu vào một tệp tin và gửi thông báo qua UART cho việc gỡ lỗi hoặc mục đích theo dõi quá trình ghi tệp USB.

- Kiểm tra sự tồn tại của tệp: gọi hàm f_stat để kiểm tra xem tệp có tồn tại hay không.Nếu tệp không tồn tại (f_stat trả về giá trị khác FR_OK), chương trình sẽ gửi thông báo lỗi qua UART và trả về kết quả lỗi.
- Mở tệp để ghi: Nếu tệp tồn tại, hàm tiếp tục bằng việc mở tệp để ghi dữ liệu (f_open). Nếu việc mở tệp không thành công (f_open trả về giá trị khác FR_OK), thông báo lỗi sẽ được gửi qua UART và hàm sẽ trả về kết quả lỗi.
- Ghi dữ liệu vào tệp:Nếu tệp được mở để ghi thành công, hàm sẽ gửi thông báo qua UART cho biết tệp đã được mở để ghi. Tiếp theo, nó sử dụng hàm f_write để ghi dữ liệu từ bộ đệm (data) vào tệp. Nếu việc ghi không thành công, thông báo lỗi sẽ được gửi qua UART.
- Đóng tệp và xử lý kết quả: Sau khi ghi dữ liệu xong, tệp sẽ được đóng (f_close). Nếu việc đóng tệp không thành công, thông báo lỗi sẽ được gửi qua UART. Nếu việc ghi và đóng tệp đều thành công, thông báo thành công sẽ được gửi qua UART.

- Giải phóng bộ nhớ và trả về kết quả:Cuối cùng, bộ nhớ được cấp phát và không còn cần thiết (buf) sẽ được giải phóng. Hàm sẽ trả về kết quả của quá trình ghi tệp (có thể là FR_OK nếu không có lỗi, hoặc mã lỗi nếu có lỗi xảy ra).

```
extern int Appli_state;

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint8_t rx_buffer[5268];

/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart1;

  /* USER CODE BEGIN 2 */

  HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, 0);
  HAL_UART_Receive(&huart1, rx_buffer, sizeof(rx_buffer), HAL_MAX_DELAY);
  HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
```

Giải thích:

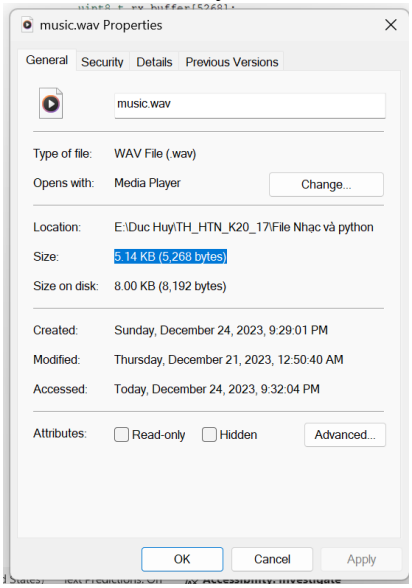
1. extern int Appli_state;

→ Khai báo biến Appli_state kiểu int.

2. uint8_t rx_buffer[5268];

→ Khai báo biến buffer có 5268 btyes mỗi byte có kiểu dữ liệu uint8_t.

→ Khai báo 5268 bytes vì file nhạc gửi vào có 5268 bytes.



2. HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, 0);

- Hàm này được sử dụng để viết giá trị vào chân GPIO để điều khiển một thiết bị, một đèn LED có kết nối với một chân GPIO cụ thể trên vi điều khiển STM32.Trong đó:
  - LED_GPIO_Port: Đây là con trỏ đến cổng GPIO mà đèn LED được kết nối.

- LED_Pin: Đây là mã số (hoặc bit) của chân GPIO mà đèn LED được kết nối.
- 0: Tham số thứ ba là trạng thái bạn muốn thiết lập cho chân GPIO, có nghĩa là GPIO_PIN_RESET hoặc chân GPIO sẽ được đặt về trạng thái thấp, tắt đèn LED.

3. HAL_UART_Receive(&huart1, rx_buffer, sizeof(rx_buffer), HAL_MAX_DELAY);

- Hàm này được sử dụng để nhận dữ liệu từ module UART (Universal Asynchronous Receiver/Transmitter) hoặc cổng giao tiếp serial trên vi điều khiển STM32.
  - &huart1: Đây là con trỏ đến cấu hình của UART được sử dụng. biến *huart1* đã được khai báo trước đó để sử dụng UART1.
  - rx_buffer: Đây là buffer (bộ đệm) để lưu trữ dữ liệu nhận được từ UART.
  - sizeof(rx_buffer): Độ dài của buffer nhận được, thường được thiết lập để nhận một lượng dữ liệu cố định.
  - HAL_MAX_DELAY: Thời gian chờ tối đa khi nhận dữ liệu, chờ đến khi dữ liệu được nhận hoặc có lỗi xảy ra.

4. HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);

- Hàm này được sử dụng để chuyển đổi trạng thái của chân GPIO từ trạng thái cao (HIGH) sang trạng thái thấp (LOW) hoặc ngược lại.
  - LED_GPIO_Port: Con trỏ đến cổng GPIO mà đèn LED được kết nối.
  - LED_Pin: Mã số (hoặc bit) của chân GPIO mà đèn LED được kết nối.

```
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */
  MX_USB_HOST_Process();

  /* USER CODE BEGIN 3 */
  switch(Appli_state)
     {
      case APPLICATION_READY:
          Mount_USB();

          Check_USB_Details();   // check space details

          Scan_USB("/");   // scan for files and directories

          Create_Dir("/Music");
          Create_File("/Music/Music_File.wav");


          Write_File("/Music/Music_File.wav", rx_buffer, sizeof(rx_buffer));

          Appli_state = APPLICATION_IDLE;
          break;
      case APPLICATION_IDLE:
      default:
          break;
     }
}
/* USER CODE END 3 */
```

Giải thích:

1. MX_USB_HOST_Process():

→ Hàm này là một phần của CubeMX generated code, thường được sử dụng để xử lý các sự kiện và giao tiếp với thiết bị USB.

- Switch case với biến Appli_state:
- Biến Appli_state đang được sử dụng để xác định trạng thái hiện tại của ứng dụng.

Các trường hợp ứng với `APPLICATION_READY`:

- Mount_USB(): Gắn kết thiết bị USB.
- Check_USB_Details(): Kiểm tra chi tiết không gian lưu trữ trên thiết bị USB.
- Scan_USB("/"): Quét các tệp và thư mục trên thiết bị USB từ thư mục gốc.
- Create_Dir("/Music"): Tạo thư mục /Music trên thiết bị USB.
- Create_File("/Music/Music_File.wav"): Tạo một tệp tin có tên Music_File.wav trong thư mục /Music.
- Write_File("/Music/Music_File.wav", rx_buffer, sizeof(rx_buffer)): Ghi dữ liệu từ rx_buffer (có kích thước sizeof(rx_buffer)) vào tệp tin vừa được tạo.

```python
import serial
import time

# Cổng UART - điều chỉnh theo cấu hình
uart_port = 'COM16'
uart_baudrate = 115200

# Mở cổng UART
ser = serial.Serial(uart_port, uart_baudrate)

# Tên tập tin âm thanh WAV
audio_file_path = 'music.wav'

# Đọc dữ liệu từ tập tin và gửi qua UART
with open(audio_file_path, 'rb') as audio_file:
    data = audio_file.read()
    ser.write(data)

# Đóng cổng UART
ser.close()
```

Giải thích:

File Python có chức năng:

- Kết nối với board thông qua hàm serial.Serial(cổng COM đang kết nối ms board, tốc độ baudrate).

- Mở tệp có tên music.wav sau đó đọc dữ liệu và gán cho biến data.
- Dùng hàm ser.write(data) để gửi biến data xuống board và trong board biến được gửi xuống được lưu trong tên biến buffer.

IV/ Tổng kết:

Việc hoàn thành các bài lab này rất có ý nghĩa quan trọng trong hành trình học tập của chúng em. Qua đó,trang bị cho chúng em những kỹ năng thực tế, củng cố kiến thức lý thuyết và nuôi dưỡng tinh thần hợp tác trong nhóm của chúng tôi. Chúng em rất biết ơn vì có cơ hội tham gia vào các bài tập này và mong muốn áp dụng những kỹ năng mới tìm được này vào các tình huống thực tế.

Nhóm của chúng em đã hoàn thành đầy đủ tất cả các yêu cầu trong 2 lab 01 và 02 được giao trong khung thời gian được chỉ định. Trong quá trình làm bài, với những yêu cầu vừa cơ bản vừa phong phú, mang đến cho chúng em sự hiểu biết thực tế về chủ đề này đồng thời hoàn thiện năng lực kỹ thuật của chúng em. Mỗi bài lan đều vừa là thách thức vừa là cơ hội học tập để chúng em trau chuốt thêm các khuyết điểm còn sót. Thông qua các bài tập này, chúng em đã có được kinh nghiệm thực tế trong việc áp dụng kiến thức lý thuyết vào các tình huống thực tế. Chúng em đã mài giũa kỹ năng giải quyết vấn đề của mình, học cách giải quyết những vấn đề phức tạp và phát triển sự hiểu biết sâu sắc hơn về các khái niệm được dạy trong lớp.

Sự hợp tác trong nhóm của chúng em là rất quan trọng trong việc hoàn thành các bài tập trong phòng thí nghiệm. Làm việc cùng nhau, chúng em tận dụng thế mạnh của nhau, chia sẻ hiểu biết sâu sắc và cùng nhau giải quyết những trở ngại gặp phải trong quá trình hoàn thành bài tập.

V/ Mô phỏng:

Lab 1+2: Demo_lab_1_2

VI/ Tài liệu tham khảo:

https://docs.scintilla.utwente.nl/cursus/MicrocontrollerCourse2015/STM32CubeF4-HAL-description.pdf

https://www.st.com/resource/en/user_manual/um1725-description-of-stm32f4-hal-and-lowlayer-drivers-stmicroelectronics.pdf

https://wiki.st.com/stm32mcu/wiki/Introduction_to_USB_with_STM32?fbclid=IwAR0f7H0A3RqCrsBLze9IxImDi5q1CN9fRu7hlNOG9omTkJECa3e12tRCwUM

https://www.st.com/resource/en/user_manual/um1734-stm32cube-usb-device-library-stmicroelectronics.pdf