

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HỒ CHÍ MINH**

**KHOA ĐÀO TẠO CHẤT LƯỢNG CAO**

🙟🕮🙝



**MÔN: HỆ THỐNG NHÚNG**

**ĐỀ TÀI:**

**CẢM BIẾN VÂN TAY**

**VÀ ỨNG DỤNG**

**(QUẢN LÍ VÀ MỞ KHÓA CỬA)**

**(Báo cáo cuối kì )**

**GVHD : ThS. ĐINH CÔNG ĐOAN**

**SVHD: DIỆP GIA HỮU 17110158**

**ĐOÀN QUỐC HÙNG 17110154**

**LÂM GIA KHÁNH 17110160**

**TP.Hồ Chí Minh, ngày 10 tháng 12 năm 2018**

**MỤC LỤC**

[**PHẦN MỞ ĐẦU** 4](#_Toc532315528)

[**1.** **Tóm tắt ý tưởng nội dung báo cáo:** 4](#_Toc532315529)

[**2.** **Đặt vấn đề:** 4](#_Toc532315530)

[***2.1 Những nghiên cứu trong và ngoài nước liên quan đến đề tài.*** 4](#_Toc532315531)

[***2.2 Một số tài liệu liên quan đến đề tài*** 5](#_Toc532315532)

[***2.3 Tính cấp thiết cần nghiên cứu của đề tài*** 5](#_Toc532315533)

[***2.4 Mục tiêu đề tài*** 5](#_Toc532315534)

[***2.5 Đối tượng phạm vi nghiên cứu*** 5](#_Toc532315535)

[***2.6 Phương pháp nghiên cứu*** 6](#_Toc532315536)

[***2.7 Nội dung đề tài*** 6](#_Toc532315537)

[**PHẦN NỘI DUNG** 7](#_Toc532315538)

[**Chương 1: GIỚI THIỆU VI XỬ LÍ** 7](#_Toc532315539)

[***1.1. Giới thiệu dòng Vi xử lí ARM Cortex M3*** 7](#_Toc532315540)

[**1.1.1 Dòng vi xử lý Cortex của ARM** 7](#_Toc532315541)

[**1.1.2 Kiến trúc ARMV7 và tập lệnh Thumb-2** 8](#_Toc532315542)

[**1.1.3 Giới thiệu ARM Cortex M3** 9](#_Toc532315543)

[**1.1.4 Bộ điều khiển vecto ngắt lồng nhau (NVIC)** 11](#_Toc532315544)

[**1.1.5 Đơn vị bảo vệ bộ nhớ (MPU)** 13](#_Toc532315545)

[***1.2. Vi xử lí STM32*** 14](#_Toc532315546)

[**1.2.1 Giới thiệu** 14](#_Toc532315547)

[**1.2.2 Các đặc điểm nổi bật của STM32** 14](#_Toc532315548)

[**1.2.3 Bộ nhớ trong STM32** 17](#_Toc532315549)

[**Chương 2: CẤU TRÚC CHUNG CỦA KIT** 19](#_Toc532315550)

[***2.1*** ***Giới thiệu Kit STM32F103C8T6*** 19](#_Toc532315551)

[**Chương 3: NHỮNG KIẾN THỨC LIÊN QUAN** 22](#_Toc532315552)

[***3.1. Giới thiêu sơ lược về protocol USART, module Fingerprint và LCD16x02*** 22](#_Toc532315553)

[**3.1.1 Sơ lược về lý thuyết USART** 22](#_Toc532315554)

[**3.1.2** **Sơ lược về module Fingerprint R305** 23](#_Toc532315555)

[**3.1.3** **Giao tiếp với lcd16x02** 30](#_Toc532315556)

[**Chương 4: ỨNG DỤNG** 31](#_Toc532315557)

[**4.1 Ứng dụng trong đời sống** 31](#_Toc532315558)

[**4.2 Mô tả ứng dụng** 31](#_Toc532315559)

[**4.3 Cách nạp chương trình vào KIT** 32](#_Toc532315560)

[**4.4. Chương trình** 36](#_Toc532315561)

[**4.4.1** **Chương trình chính** 36](#_Toc532315562)

[**4.4.2** **Chương trình con** 37](#_Toc532315563)

[**4.4.3** **Thư viện mẫu** 40](#_Toc532315564)

[**4.4.4** **Hàm thực thi ngắt và hàm con của lcd** 44](#_Toc532315565)

[**KẾT LUẬN** 49](#_Toc532315566)

[**I.** **Kết quả đạt được** 49](#_Toc532315567)

[**II.** **Ưu, nhược điểm** 49](#_Toc532315568)

[**1.** **Ưu điểm** 49](#_Toc532315569)

[**2.** **Nhược điểm** 49](#_Toc532315570)

[**III.** **Hướng phát triển** 49](#_Toc532315571)

[**PHẦN BỔ SUNG** 50](#_Toc532315572)

[**1.** **Sơ đồ nối module đọc vân tay với STM32** 50](#_Toc532315573)

[**2.** **Mô hình thực tế** 51](#_Toc532315574)

[**3.** **Giải thích thư viện và hàm con** 51](#_Toc532315575)

# **PHẦN MỞ ĐẦU**

1. **Tóm tắt ý tưởng nội dung báo cáo:**

 Cuộc cách mạng khoa học kỹ thuật thế kỷ XX bắt nguồn từ những nhu cầu càng lúc càng tăng cao của con người trong suốt tiến trình lịch sử trong khi sức lực và khả năng (sinh học) của con người có hạn không thể đáp ứng tất cả các nhu cầu ngày càng tăng ấy, đồng thời tài nguyên thiên nhiên, vật liệu tự nhiên về số lượng và tính chất cũng có giới hạn, không thể đáp ứng những yêu cầu mới nảy sinh trong cuộc sống.

Ngày nay, khoa học kỹ thuật phát triển như vũ bão. Hệ thống thiết bị điện tử, mạch điện tử, điện tử kỹ thuật số đã làm thay đổi sâu sắc toàn bộ hoạt động sản xuất của con người. Kỹ thuật điện tử số đã và đang thay thế dần các kỹ thuật tương tự và còn đóng vai trò then chốt trong cuộc cách mạng kỹ thuật và công nghệ. Hiện nay, hệ thống kiểm soát thang máy, máy chấm công vân tay và kiểm soát ra vào ngày càng được sử dụng nhiều do có ưu việt về độ chính xác, mỹ quan. Có thể sử dụng cho tòa cao ốc, công xưởng có quy mô nhỏ hay khu công nghiệp có quy mô lớn hàng vạn công nhân cần chấm công, kiểm soát ra vào.

Hệ thống máy chấm công vân tay kiểm soát ra vào là một cỗ máy làm thay nhiệm vụ chấm công của 1 nhân viên nhân sự làm nhiệm vụ chấm công. Nhưng vì là máy móc nên sẽ không có sự nể nang hay sai sót trong việc chấm công tính lương và kiểm soát giờ giấc làm việc của nhân viên.

1. **Đặt vấn đề:**

## ***2.1 Những nghiên cứu trong và ngoài nước liên quan đến đề tài.***

Tình hình nghiên cứu về công nghệ nhận dạng vân tay Con người đã biết sử dụng dấu vân tay từ rất sớm. Vào thời cổ đại, các thương gia ở Babylon đã biết dùng dấu vân tay được in lên viên đất sét trong trao đổi hàng hóa. Ở Trung Quốc, người ta cũng đã tìm thấy các ngón tay cái được in lên các con dấu đất sét. Nhưng bắt đầu từ thế kỷ 19, dấu vân tay mới được đưa vào nghiên cứu chính thức. • Năm 1823, nhà phẫu thuật Jan Evangelista Purkyne thuộc trường đại học Breslau đã trình bày trong luận án của mình về 9 mẫu vân tay. • Năm 1858, William Herschel đã dựa vào vết vân tay để nhận dạng tù nhân. Năm 1880, bác sĩ Người Anh Henry Faulds đưa ra kiến nghị lấy dấu vân tay của tội phạm tại hiện trường xảy ra vụ án và đưa ra lý luận gien vân tay. Năm 1882, theo sáng kiến của A. Bertion, lần đầu tiên cảnh sát Paris đã áp dụng lăn ngón tay trên các hồ sơ căn cước. Trang 11 • Năm 1892, Francis Galton là người đầu chia vân tay thành 3 nhóm: xoáy, móc, sóng. Việc sử dụng các nghiên cứu khoa học của dấu vân tay ở thế kỷ 19 đã làm tiền đề sau này cho việc ứng dụng rộng rãi công nghệ nhận dạng vân tay trong nhiều lĩnh vực của cuộc sống. • Năm 1924, FBI (Federal Bureau of Investigation) đã thu thập và lưu trữ hơn 250 triệu dấu vân tay của người dân để cho việc điều tra tội phạm và nhận dạng những người bị giết. • Nước Anh cũng sớm sử dụng biện pháp này và đến năm 1944, họ đã lưu trữ tới hơn 90 triệu dấu vân tay của tất cả binh lính và những người dân. Với việc sử dụng dấu vân tay để nhận dạng, cảnh sát có thể truy tìm tung tích tội phạm, người chết, mất thẻ căn cước hoặc mắc bệnh tâm thần lú lẫn, … • Năm 1977, chương trình IAI's Certified Latent Print Examiner ra đời được áp dụng để xác nhận phạm nhân trong tòa án. Với sự phát triển ngày càng nhanh chóng của khoa học kỹ thuật, cho tới nay các ứng dụng công nghệ này được áp dụng rộng rãi khá thành công trên thế giới

## ***2.2 Một số tài liệu liên quan đến đề tài***

- Nội dung trên 1 số trang web: https://tinhte.vn/, <http://www.st.com>, http://codientu.org...

- Đinh Công Đoan, Bài giảng Hệ thống nhúng, khoa CNTT đại học SPKT TP.HCM

- Joseph Yiu, “The Definitive Guide to the ARM Cortex-M3”, Elsevier Newnes, 2007

- <http://www.rhydolabz.com/documents/finger-print-module.pdf> User manual of R305

## ***2.3 Tính cấp thiết cần nghiên cứu của đề tài***

- Kiểm soát bảo mật của người dùng, hoặc có thể dùng trong quản trị công ty, kiểm soát ra vào, có tính bảo mật cao. Là một phương án bảo mật được tin cậy hiện nay.

## ***2.4 Mục tiêu đề tài***

- Giúp mọi người hiểu rõ hơn về vi xử lý cũng như dòng STM32F1.

- Hiểu được phương thức giao tiếp truyền thông USART với dòng vi điều khiển.- Hiểu thêm về cách sử dụng và vận dụng module fingerprint trong cuộc sống.

## ***2.5 Đối tượng phạm vi nghiên cứu***

Đối tượng: Kit STM32F103C8T6

Phạm vi: Phạm vi đề tài mang tính chất nhỏ để hỗ trợ kiến thức cho sinh viên về các dòng vi xử lý trong đó có dòng STM32.

## ***2.6 Phương pháp nghiên cứu***

* Thông qua tài liệu tìm kiếm trên Internet.
* Dựa trên tài liệu kèm theo dòng Cortex M3- STM32F1.

## ***2.7 Nội dung đề tài***

* Giới thiệu về ARM Cortex M3
* Trình bày cấu trúc chung của KIT STM32F1 và các thành phần của KIT
* Những kiến thức liên quan
* Ứng dụng
* Mã nguồn của chương trình

# **PHẦN NỘI DUNG**

# **Chương 1: GIỚI THIỆU VI XỬ LÍ**

## ***1.1. Giới thiệu dòng Vi xử lí ARM Cortex M3***

### **1.1.1 Dòng vi xử lý Cortex của ARM**

Giải pháp SOC (System-on-chip) dựa trên bộ vi xử lý nhúng ARM được ứng dụng vào rất nhiều thị trường khác nhau bao gồm các ứng dụng doanh nghiệp, các hệ thống ô tô, mạng gia đình và công nghệ mạng không dây... Dòng vi xử lý ARM Cortex dựa trên một kiến trúc chuẩn đủ để đáp ứng hầu hết các yêu cầu về hiệu năng làm việc trong tất cả các lĩnh vực trên. Dòng ARM Cortex bao gồm ba cấu hình khác nhau của kiến trúc ARMv7:

* Cortex-A cho các ứng dụng tinh vi, yêu cầu cao chạy trên các hệ điều hành mở.
* Cortex-R dành cho các hệ thống thời gian thực.
* Cortex-M được tối ưu cho các ứng dụng vi điều khiển, cần tiết kiệm chi phí.

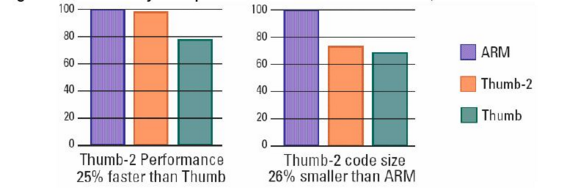
Bộ vi xử lý Cortex-M3 là bộ vi xử lý ARM đầu tiên dựa trên kiến trúc ARMv7-M và được thiết kế đặc biệt để đạt được hiệu suất cao trong các ứng dụng nhúng cần tiết kiệm năng lượng và chi phí, chẳng hạn như các vi điều khiển, hệ thống cơ ô tô, hệ thống kiểm soát công nghiệp và hệ thống mạng không dây. Thêm vào đó là việc lập trình được đơn giản hóa đáng kể giúp kiến trúc ARM trở thành một lựa chọn tốt cho ngay cả những ứng dụng đơn giản nhất.

Phiên bản tiếp theo của dòng Cortex-M cũng được dựa trên kiến trúc ARMv7-M là Cortex-M3. ARM Cortex M3 là một bộ xử lí hiệu suất cao được tích hợp DSP (Digital Signal Processor – bộ xử lý tín hiệu số), được phát triển để giải quyết vấn đề kiểm soát thị trường tín hiệu số một cách hiệu quả, dễ sử dụng. ARM Cortex M3 cung cấp những lợi ích đáng kể cho nhà phải triển bảo gồm:

* Hiệu năng xử lý vượt trội với xử lý ngắt nhanh chóng.
* Tăng cường hệ thống gỡ lỗi với khả năng mở rộng breakpoint và dấu vết.
* Bộ lỗi xử lý, hệ thống và bộ nhớ hiệu quả.
* Tiêu thụ điện năng cực thấp với chế độ ngủ tích hợp và tùy chọn chế độ ngủ sâu.

### **1.1.2 Kiến trúc ARMV7 và tập lệnh Thumb-2**

ARMv7-M là cấu hình vi điều khiển của kiến trúc ARMV7 và khác với các kiến trúc ARM trước đó ở chỗ nó chỉ hỗ trợ tập lệnh Thumb-2. Tập lệnh Thumb-2 là sự pha trộn giữa tập lệnh 16 và 32 bit, đạt được hiệu suất của các lệnh ARM 32 bit, đồng thời phù hợp với mật độ mã cũng như tương thích ngược với tập lệnh gốc Thumb 16 bit.



Hình 1: So sánh hiệu suất tương đối và độ rộng code của Thumb và Thumb-2 đối với ARM

Trong một hệ thống dựa trên bộ vi xử lý ARM7, việc chuyển đổi nhân xử lý giữa chế độ Thumb (có lợi về mật độ mã) và ARM (có lợi về mặt hiệu suất) là cần thiết cho một số ứng dụng. Còn bộ vi xử lý Cortex-M4 có các lệnh 16 bit và 32 bit tồn tại trong cùng một chế độ, cho phép mật độ mã cũng như hiệu suất đều cao hơn mà không cần phải chuyển đổi phức tạp. Vì tập lệnh Thumb-2 là tập bao hàm của tập lệnh Thumb 16 bit nên bộ vi xử lý Cortex-M4 có thể thực thi các đoạn mã trước đây viết cho Thumb 16 bit. Do được cài đặt tập lệnh Thumb-2 nên bộ vi xử lý Cortex-M4 có khả năng tương thích với các thành viên khác của dòng ARM Cortex.

Tập lệnh Thumb-2 có các lệnh đặc biệt giúp lập trình viên dễ dàng viết mã cho nhiều ứng dụng khác nhau. Các lệnh BFI và BFC là các lệnh thao tác trên bit, rất có ích trong các ứng dụng xử lý gói tin mạng. Các lệnh SBFX và UBFX giúp việc chèn vào hoặc trích xuất một số bit trong thanh ghi được nhanh chóng. Lệnh RBIT đảo bit trong một WORD, có ích trong các thuật toán DSP như DFT. Các lệnh bảng rẽ nhánh TBB và TBH tạo sự cân bằng giữa mật độ mã và hiệu suất. Tập lệnh Thumb-2 cũng giới thiệu cấu trúc If-Then mới có thể xác định điều kiện thực hiện tối đa bốn lệnh tiếp theo.

### **1.1.3 Giới thiệu ARM Cortex M3**

Lõi trung tâm Cortex-M3 dựa trên kiến trúc Harvard, được đặc trưng bằng sự tách biệt giữa vùng nhớ chứa dữ liệu và chương trình do đó có các bus riêng để truy cập (hình 3). Đặc tính này khác với dòng ARM7 dựa trên kiến trúc Von Neumann sử dụng chung vùng nhớ để chứa dữ liệu và chương trình, do đó dùng chung bus cho việc truy xuất. Vì có thể đọc cùng lúc lệnh và dữ liệu từ bộ nhớ, bộ vi xử lý Cortex-M3 có thể thực hiện nhiều hoạt động song song, tăng tốc thực thi ứng dụng.  
Lõi Cortex có cấu trúc đường ống gồm 3 tầng: Instruction Fetch, Instruction Decode và Instruction Execute. Khi gặp một lệnh nhánh, tầng decode chứa một chỉ thị nạp lệnh suy đoán có thể dẫn đến việc thực thi nhanh hơn. Bộ xử lý nạp lệnh dự định rẽ nhánh trong giai đoạn giải mã. Sau đó, trong giai đoạn thực thi, việc rẽ nhánh được giải quyết và bộ vi xử lý sẽ phân tích xem đâu là lệnh thực thi kế tiếp. Nếu việc rẽ nhánh không được chọn thì lệnh tiếp theo đã sẵn sàng. Còn nếu việc rẽ nhánh được chọn thì lệnh rẽ nhánh đó cũng đã sẵn sàng ngay lập tức, hạn chế thời gian rỗi chỉ còn một chu kỳ.  
Lõi Cortex-M3 chứa một bộ giải mã cho tập lệnh Thumb truyền thống và Thumb-2 mới, một ALU tiên tiến hỗ trợ nhân chia phần cứng, điều khiển logic, và các giao tiếp với các thành phần khác của bộ xử lý.  
 Bộ vi xử lý Cortex-M3 là một bộ vi xử lý 32-bit, với độ rộng của đường dẫn dữ liệu 32 bit, các dải thanh ghi và giao tiếp bộ nhớ. Có 13 thanh ghi đa dụng, hai con trỏ ngăn xếp, một thanh ghi liên kết, một bộ đếm chương trình và một số thanh ghi đặc biệt trong đó có một thanh ghi trạng thái chương trình.  
 Bộ vi xử lý Cortex-M3 hỗ trợ hai chế độ hoạt động (Thread và Handler) và hai mức truy cập tài nguyên của lõi xử lí (đặc quyền và không đặc quyền), tạo điều kiện cho việc cài đặt các hệ thống mở và phức tạp nhưng vẫn bảo mật. Những dòng mã không đặc quyền bị giới hạn hoặc không cho phép truy cập vào một số tài nguyên quan trọng (một số lệnh đặc biệt và các vùng nhớ nhất định). Chế độ Thread là chế độ hoạt động tiêu biểu hỗ trợ cả mã đặc quyền và không đặc quyền. Bộ vi xử lý sẽ vào chế độ Handler khi một ngoại lệ (exception) xảy ra và tất cả các mã là đặc quyền trong chế độ này. Ngoài ra, tất cả các hoạt động trong bộ vi xử lý đều thuộc một trong hai trạng thái hoạt động: Thumb cho chế độ thực thi bình thường và Debug cho việc gỡ lỗi.  
  
 Bộ vi xử lý Cortex-M3 là một hệ thống ánh xạ bộ nhớ đơn giản, quản lí vùng nhớ cố định lên tới 4 gigabyte với các địa chỉ định nghĩa sẵn, dành riêng cho mã lệnh (vùng mã lệnh), SRAM (vùng nhớ), bộ nhớ/thiết bị bên ngoài, thiết bị ngoại vi bên trong và bên ngoài. Ngoài ra còn có một vùng nhớ đặc biệt dành riêng cho nhà cung cấp.  
 Bộ vi xử lý Cortex-M3 cho phép truy cập trực tiếp đến từng bit dữ liệu trong các hệ thống đơn giản bằng cách thực thi một kỹ thuật được gọi là bit-banding (hình 5). Bộ nhớ bao gồm hai vùng bit-band (mỗi vùng 1MB) trong SRAM và vùng bí danh 32MB của vùng không gian ngoại vi (Mỗi byte trong vùng bí danh sẽ tương ứng với một bit trong vùng bit-band). Mỗi hoạt động nạp/lưu tại một địa chỉ trong khu vực bí danh (alias region) sẽ trực tiếp tương ứng với hoạt động trên bit được đại diện bởi bí danh đó. Cụ thể, khi ghi giá trị 0x01 vào một địa chỉ trên vùng bí danh thì có nghĩa là xác định bit tương ứng sẽ có giá trị là 1, tương tự giá trị 0x00 sẽ xác định bit tương ứng có giá trị 0. Còn đọc giá trị tại một địa chỉ vùng bí danh có nghĩa là đọc được giá trị của bit tương ứng. Một vấn đề cần chú ý nữa là hoạt động này mang tính nguyên tử (không chia nhỏ được nữa), không thể bị gián đoạn bởi các hoạt động khác trên bus.  
 Các hệ thống cũ dựa trên ARM7 chỉ hỗ trợ truy xuất dữ liệu thẳng hàng, chỉ cho phép lưu trữ và truy xuất dữ liệu của một khối bộ nhớ mà mỗi phần tử có đơn vị là một word. Bộ vi xử lý Cortex-M3 hỗ trợ truy xuất dữ liệu không thẳng hàng, cho phép chuyển dữ liệu không thẳng hàng trong một truy xuất đơn. Thực tế, việc chuyển dữ liệu không thẳng hàng được biến thành việc chuyển nhiều lần dữ liệu thẳng hàng và có tính trong suốt đối với lập trình viên (nghĩa là lập trình viên hoàn toàn không cần quan tâm đến điều này). Ngoài ra bộ vi xử lý Cortex-M3 cũng hỗ trợ phép nhân 32-bit hoạt động trong một chu trình đơn và các phép chia có dấu, không dấu với các lệnh SDIV và UDIV, mất từ 2 đến 12 chu kỳ phụ thuộc vào kích thước của toán hạng. Phép chia được thực thi nhanh hơn nếu số chia và số bị chia có kích thước tương tự nhau. Những cải tiến trong khả năng toán học giúp Cortex-M3 trở thành bộ vi xử lý lý tưởng cho các ứng dụng thiên về tính toán như đọc cảm biến hoặc các hệ thống mô phỏng.

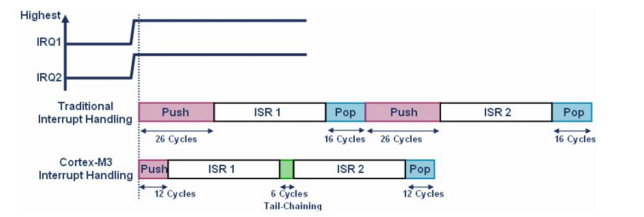
### **1.1.4 Bộ điều khiển vecto ngắt lồng nhau (NVIC)**

NVIC (Nested Vectored Interrupt Controller) là thành phần tích hợp của bộ vi xử lý Cortex-M4 có khả năng xử lý ngắt rất linh hoạt và nhanh chóng. Trong cài đặt chuẩn, nó cung cấp một NMI (Non-Maskable Interrupt) và 32 ngắt vật lý đa dụng với 8 mức ưu tiên pre-emption. Nó có thể được cấu hình từ 1 đến 240 ngắt vật lý với tối đa 256 mức độ ưu tiên.

Bộ vi xử lý Cortex-M4 sử dụng một bảng vector có thể tái định vị được, dùng để chứa địa chỉ của hàm xử lý ngắt. Khi nhận một ngắt, bộ xử lý sẽ lấy địa chỉ từ bảng vector thông qua bus chương trình. Bảng vector ngắt được đặt ở địa chỉ 0 khi reset, nhưng có thể được di chyển đến vị trí khác bằng cách lập trình một thanh ghi điều khiển.

Để giảm bớt số cổng và tăng tính linh hoạt hệ thống, bộ vi xử lý Cortex-M4 đã chuyển từ mô hình ngoại lệ thanh ghi theo dõi của bộ vi xử lý ARM7 sang mô hình ngoại lệ dựa trên stack. Khi có một ngoại lệ xuất hiện thì bộ đếm chương trình (Program Counter), thanh ghi trạng thái chương trình (Program Status Register), thanh ghi liên kết (Link Register) và các thanh ghi đa dụng từ R0-R3, R12 bị đẩy vào ngăn xếp. Trong khi bus dữ liệu đẩy các thanh ghi lên vùng ngăn xếp thì bus chương trình xác định các vector ngoại lệ từ bảng vector và nạp lệnh đầu tiên của mã chương trình xử lí ngoại lệ. Sau khi hoàn tất việc lưu trữ dữ liệu trên ngăn xếp và nạp lệnh, chương trình phục vụ ngắt và xử lý lỗi được thực thi, tiếp theo đó các thanh ghi sẽ được phục hồi tự động để chương trình bị ngắt tiếp tục thực hiện bình thường. Vì thực hiện các hoạt động ngăn xếp bằng phần cứng nên ta không cần viết các đoạn hợp ngữ để thực hiện các thao tác trên ngăn xếp cho các hàm xử lý ngắt truyền thống dựa trên ngôn ngữ C, giúp việc phát triển ứng dụng dễ dàng hơn rất nhiều.

NVIC hỗ trợ ngắt lồng nhau, cho phép một ngắt được xử lý trước một ngắt khác dựa trên mức độ ưu tiên. Nó cũng hỗ trợ cấu hình mức ưu tiên động cho các ngắt. Độ ưu tiên có thể được thay đổi bằng phần mềm trong thời gian chạy (run time). Các ngắt đang được xử lý đều bị khóa cho đến khi hàm xử lý ngắt hoàn thành, do đó, độ ưu tiên của ngắt có thể thay đổi mà không cần lo đến chuyện trùng lặp.

Trong trường hợp các ngắt nối đuôi nhau, các hệ thống cũ sẽ lặp lại hai lần việc lưu trạng thái hoàn thành và khôi phục, dẫn đến độ trễ cao. Bộ vi xử lý Cortex-M4 đơn giản hóa việc chuyển đổi giữa các ngắt đang hoạt động và đang chờ bằng cách cài đặt công nghệ tail-chaining trong phần cứng NVIC. Tail-chaining đạt độ trễ thấp hơn nhiều bằng cách thay thế chuỗi các thao tác pop và push vốn mất hơn 30 chu kỳ xung nhịp bằng một thao tác nạp lệnh đơn giản chỉ mất 6 chu kỳ. Trạng thái bộ vi xử lý được tự động lưu khi ngắt bắt đầu được xử lý và phục hồi ngay khi kết thúc, ít chu kỳ hơn so với việc thực thi bằng phần mềm, nâng cao hiệu suất đáng kể ở hệ thống hoạt động dưới 100MHz.

Hình 2: Tail chaining trong NVIC

NVIC cũng cài đặt cách thức quản lý năng lượng của bộ vi xử lý Cortex-M3 tích hợp chế độ ngủ. Chế độ Sleep Now được gọi bằng một trong hai lệnh WFI (Wait For Interrupt) hoặc WFE (Wait For Event) sẽ ngay lập tức đặt nhân bộ vi xử lý vào trạng thái năng lượng thấp và chờ một ngoại lệ (exception). Chế độ Sleep On Exit đặt hệ thống vào chế độ năng lượng thấp ngay khi nó thoát khỏi hàm xử lý ngắt có độ ưu tiên thấp nhất. Nhân bộ vi xử lý vẫn ở trạng thái ngủ cho đến khi gặp một ngoại lệ. Vì chỉ có thể thoát khỏi chế độ này bằng ngắt nên trạng thái hệ thống không được phục hồi. Bit SLEEPDEEP của thanh ghi điều khiển hệ thống nếu được thiết lập có thể được sử dụng để khoá cổng (clock gate) lõi bộ vi xử lý và các thành phần hệ thống khác để tiết kiệm điện năng.

NVIC cũng tích hợp một bộ đếm SysTick 24-bit đếm ngược (count-down timer) có thể được sử dụng để định thời tạo ra ngắt, cung cấp nhịp đập để một hệ điều hành thời gian thực hoạt động hoặc các tác vụ được lập lịch.

### **1.1.5 Đơn vị bảo vệ bộ nhớ (MPU)**

MPU là một thành phần tùy chọn của bộ vi xử lý Cortex-M4, có thể nâng cao độ tin cậy của hệ thống nhúng bằng cách bảo vệ các dữ liệu quan trọng được hệ điều hành sử dụng khỏi các ứng dụng khác, tách biệt độc lập các tác vụ đang thực thi bằng cách không cho phép truy cập vào dữ liệu của nhau, vô hiệu hoá quyền truy cập vào một số vùng nhớ, cho phép các vùng nhớ được định nghĩa là chỉ đọc (read only) và phát hiện các truy cập bộ nhớ có thể phá vỡ hệ thống.

MPU cho phép một ứng dụng được chia nhỏ thành các tiến trình. Mỗi tiến trình sẽ có bộ nhớ (code, dữ liệu, ngăn xếp, heap) và thiết bị riêng, cũng như có quyền truy cập vào bộ nhớ và các thiết bị được chia sẻ. MPU cũng có các quy tắc (rule) truy cập của người dùng và đặc quyền bao gồm việc thực thi mã tại mức đặc quyền thích hợp cũng như quyền sở hữu bộ nhớ và các thiết bị của mã đặc quyền và mã người dùng.

MPU chia bộ nhớ thành các vùng riêng biệt và thực hiện việc bảo vệ bằng cách ngăn các truy cập trái phép. MPU có thể chia bộ nhớ thành tối đa 8 vùng trong đó mỗi vùng có thể được chia thành 8 vùng con. Kích thước vùng có thể bắt đầu từ 32 byte và tăng gấp đôi dần cho đến tối đa 4 gigabyte. Các vùng được đánh số thứ tự bắt đầu từ 0. Có thể xác định một bản đồ bộ nhớ (memory map) nền mặc định để truy cập đặc quyền. Việc truy cập đến các địa chỉ bộ nhớ không được xác định trong vùng MPU hoặc không được phép sẽ tạo ra ngoại lệ lỗi về quản lí bộ nhớ (Memory Management Fault Exception).

Quy tắc bảo vệ vùng nhớ được dựa trên vào loại tác vụ (đọc, viết hoặc thực thi) và đặc quyền của mã thực hiện việc truy cập. Mỗi vùng bao gồm một bộ bit quy định loại truy cập được phép và hành động nào được phép trên bus. MPU cũng hỗ trợ các vùng chồng lên nhau (overlapping regions), tức là có sự giao nhau cùng một vùng địa chỉ. Vì kích thước mỗi vùng là bội số của 2 nên nếu 2 vùng chồng lên nhau thì sẽ có thể có một vùng nằm hoàn toàn trong vùng kia. Do đó, hoàn toàn có khả năng xảy ra trường hợp nhiều vùng nằm trọn trong một vùng hoặc trường hợp chồng lồng nhau. Trong trường hợp địa chỉ tra cứu nằm trong vùng chồng nhau thì kết quả trả về sẽ là vùng có số thứ tự cao nhất.

## ***1.2. Vi xử lí STM32***

### **1.2.1 Giới thiệu**

STM32 là dòng điều khiển của hãng ST dựa vào nền tảng lõi xử lý Cortex M4 của ARM. ST đưa ra thị trường 4 dòng dự trên ARM7 và ARM9 nhưng STM32 là một bước tiến trên đường cong chi phí.

STM32 gồm 14 biến thể được phân thành 2 nhóm:

* Dòng Perormance có tần số hoạt dộng của CPU lên tới 72 MHX
* Dòng Access có tần số hoạt động lên tới 36Mhz. Tuy nhiên có ít các ngoại vi dòng Performaince.

Hiện nay ST đưa ra thêm 2 dòng nữa là USB Access và Connectivity. Các biến thể STM32 tương thích hoàn toàn về sơ dồ chân rất tiện cho thiết kế mạch in.

Trong các nhóm lại được phân theo số lượng các thiết bị ngoại vi hỗ trợ, kích thước bộ nhớ flash mà chia thành các thiết bị với mật độ tích hợp khác nhau như: low density divices, medium density divices, high density divices, xl-line density divices, connectivity line divices.

**1.2.2 Các đặc điểm nổi bật của STM32**

* **Sự tinh vi:**

Thoạt nhìn thì các ngoại vi của STM32 cũng giống như những vi điều khiển khác, như hai bộ chuyển đổi ADC, timer, I2C, SPI, CAN, USB và RTC. Tuy nhiên mỗi ngoại vi trên đều có rất nhiều đặc điểm thú vị. Ví dụ như bộ ADC 12-bit có tích hợp một cảm biến nhiệt độ để tự động hiệu chỉnh khi nhiệt độ thay đổi và hỗ trợ nhiều chế độ chuyển đổi. Mỗi bộ định thời có 4 khối capture compare (dùng để bắt sự kiện với tính năng input capture và tạo dạng sóng ở ngõ ra với output compare), mỗi khối định thời có thể liên kết với các khối định thời khác để tạo ra một mảng các định thời tinh vi hơn. Một bộ định thời cao cấp chuyên hỗ trợ điều khiển động cơ, với 6 đầu ra PWM với dead time (khoảng thời gian được chèn vào giữa hai đầu tín hiệu xuất PWM bù nhau trong điều khiển mạch cầu H) lập trình được và một đường break input (khi phát hiện điều kiện dừng khẩn cấp) sẽ buộc tín hiệu PWM sang một trạng thái an toàn đã được cài sẵn. Ngoại vi nối tiếp SPI có một khối kiểm tổng (CRC) bằng phần cứng cho 8 và 16 word hỗ trợ tích cực cho giao tiếp thẻ nhớ SD hoặc MMC.

STM32 có hỗ trợ thêm tối đa 12 kênh DMA (Direct Memory Access). Mỗi kênh có thể được dùng để truyền dữ liệu đến các thanh ghi ngoại vi hoặc từ các thanh ghi ngoại vi đi với kích thước từ (word) dữ liệu truyền đi có thể là 8/16 hoặc 32-bit. Mỗi ngoại vi có thể có một bộ điều khiển DMA (DMA controller) đi kèm dùng để gửi hoặc đòi hỏi dữ liệu như yêu cầu. Một bộ phân xử bus nội (bus arbiter) và ma trận bus (bus matrix) tối thiểu hoá sự tranh chấp bus giữa truy cập dữ liệu thông qua CPU (CPU data access) và các kênh DMA. Điều đó cho phép các đơn vị DMA hoạt động linh hoạt, dễ dùng và tự động điều khiển các luồng dữ liệu bên trong vi điều khiển.

STM32 là một vi điều khiển tiêu thụ năng lượng thấp và đạt hiệu suất cao. Nó có thể hoạt động ở điện áp 2V, chạy ở tần số 72MHz và dòng tiêu thụ chỉ có 36mA với tất cả các khối bên trong vi điều khiển đều được hoạt động. Kết hợp với các chế độ tiết kiệm năng lượng của Cortex, STM32 chỉ tiêu thụ 2μA khi ở chế độ Standby. Một bộ dao động nội RC 8MHz cho phép chip nhanh chóng thoát khỏi chế độ tiết kiệm năng lượng trong khi bộ dao động ngoài đang khởi động. Khả năng nhanh đi vào và thoát khỏi các chế độ tiết kiệm năng lượng làm giảm nhiều sự tiêu thụ năng lượng tổng thể.

* **Sự an toàn:**

Ngày nay các ứng dụng hiện đại thường phải hoạt động trong môi trường khắc khe, đòi hỏi tính an toàn cao, cũng như đòi hỏi sức mạnh xử lý và càng nhiều thiết bị ngoại vi tinh vi. Để đáp ứng các yêu cầu khắc khe đó, STM32 cung cấp một số tính năng phần cứng hỗ trợ các ứng dụng một cách tốt nhất. Chúng bao gồm một bộ phát hiện điện áp thấp, một hệ thống bảo vệ xung Clock và hai bộ Watchdogs. Bộ đầu tiên là một Watchdog cửa sổ (windowed watchdog). Watchdog này phải được làm tươi trong một khung thời gian xác định. Nếu nhấn nó quá sớm, hoặc quá muộn, thì Watchdog sẽ kích hoạt. Bộ thứ hai là một Watchdog độc lập (independent watchdog), có bộ dao động bên ngoài tách biệt với xung nhịp hệ thống chính. Hệ thống bảo vệ xung nhịp có thể phát hiện lỗi của bộ dao động chính bên ngoài (thường là thạch anh) và tự động chuyển sang dùng bộ dao động nội RC 8MHz.

* **Tính bảo mật:**

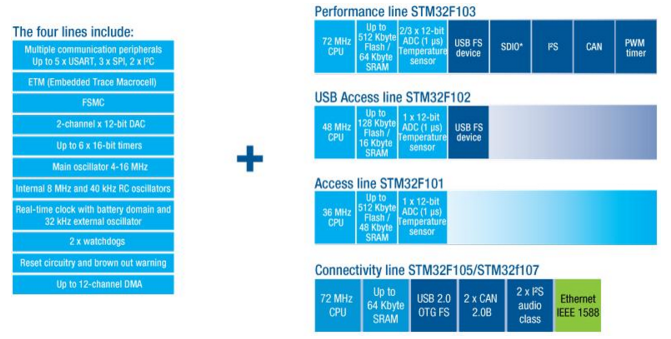
Một trong những yêu cầu khắc khe khác của thiết kế hiện đại là nhu cầu bảo mật mã chương trình để ngăn chặn sao chép trái phép phần mềm. Bộ nhớ Flash của STM32 có thể được khóa để chống truy cập đọc Flash thông qua cổng Debug. Khi tính năng bảo vệ đọc được kích hoạt, bộ nhớ Flash cũng được bảo vệ chống ghi để ngăn chặn mã không tin cậy được chèn vào bảng vector ngắt. Hơn nữa bảo vệ ghi có thể được cho phép trong phần còn lại của bộ nhớ Flash. STM32 cũng có một đồng hồ thời gian thực và một khu vực nhỏ dữ liệu trên SRAM được nuôi nhờ nguồn pin. Khu vực này có một đầu vào chống giả mạo (anti-tamper input), có thể kích hoạt một sự kiện ngắt khi có sự thay đổi trạng thái ở đầu vào này. Ngoài ra một sự kiện chống giả mạo sẽ tự động xóa dữ liệu được lưu trữ trên SRAM được nuôi bằng nguồn pin.

* **Phát triển phần mềm:**

Nếu bạn đã sử dụng một vi điều khiển dựa trên lõi ARM, thì các công cụ phát triển cho ARM hiện có đã được hỗ trợ tập lệnh Thumb-2 và dòng Cortex. Ngoài ra ST cũng cung cấp một thư viện điều khiển thiết bị ngoại vi, một bộ thư viện phát triển USB như là một thư viện ANSI C và mã nguồn đó là tương thích với các thư viện trước đó được công bố cho vi điều khiển STR7 và STR9. Có rất nhiều RTOS mã nguồn mở và thương mại và middleware (TCP/IP, hệ thống tập tin, v.v.) hỗ trợ cho họ Cortex. Dòng Cortex-M3 cũng đi kèm với một hệ thống gỡ lỗi hoàn toàn mới gọi là CoreSight. Truy cập vào hệ thống CoreSight thông qua cổng truy cập Debug (Debug Access Port), cổng này hỗ trợ kết nối chuẩn JTAG hoặc giao diện 2 dây (serial wire-2 Pin), cũng như cung cấp trình điều khiển chạy gỡ lỗi, hệ thống CoreSight trên STM32 cung cấp hệ thống điểm truy cập(data watchpoint) và một công cụ theo dõi (instrumentation trace). Công cụ này có thể gửi thông tin về ứng dụng được lựa chọn đến công cụ gỡ lỗi. Điều này có thể cung cấp thêm các thông tin gỡ lỗi và cũng có thể được sử dụng trong quá trình thử nghiệm phần mềm.

* **Dòng perfomance và Access của STM32**

Họ STM32 có hai nhánh đầu tiên riêng biệt: dòng Performance và dòng Access. Dòng Performance tập hợp đầy đủ các thiết bị ngoại vi và chạy với xung nhịp tối đa 72MHz. Dòng Access có các thiết bị ngoại vi ít hơn và chạy tối đa 36MHz. Quan trọng hơn là cách bố trí chân (pins layout) và các kiểu đóng gói chip (package type) là như nhau giữa dòng Access và dòng Performance. Điều này cho phép các phiên bản khác nhau của STM32 được hoán vị mà không cần phải sửa đổi sắp sếp lại footprint (mô hình chân của chip trong công cụ layout bo mạch) trên PCB (Printed Circuit Board).

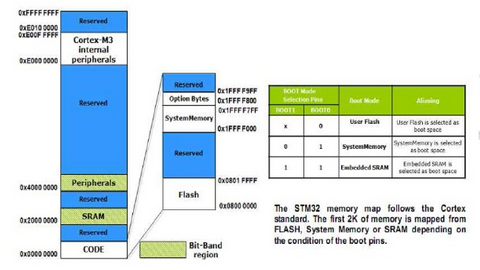
Ngoài hai dòng Performance và Access đầu tiên, hiện nay ST đã đưa ra thị trường thêm hai dòng USB Access và Connectivity như hình bên dưới.

Hình 3: Đặc điểm của 4 nhánh trong họ STM32

### **1.2.3 Bộ nhớ trong STM32**

STM32 tuân theo tiêu chuẩn phân bố bộ nhớ của Cortex. Vùng nhớ code chia làm 3 vùng nhỏ:

* Vùng User Flash dung chứa code người dung.
* Vùng System memory có dộ lớn 4kb được nhà sản xuất cài bootloader. Bootloader dung để tài chương trình thông qua Uart 1 và chứa trong User Flash.
* Vùng Option byte chứa thông tin cấu hình STM32.



Hình 4: Cấu trúc bộ nhớ của STM32

Phần chuyển từ nạp dữ liệu sang chương trình thực thi sẽ được giới thiệu tiếp trong phần mạch nạp.

Xung nhịp STM32 ngoài hỗ trợ 2 bộ tạo xung nhịp ngoài hỗ trợ 2 bộ tạo xung nhịp ngoài nó còn cung cấp thêm 2 bộ tạo dao động nội:

* High speed internal oscillator hoạt động ở mức 8Mhz
* Low speed internal oscillator hoạt động ở mức 32758Khz được dùng cho đồng hồ thời gian thực.

Dù xung nhịp được lấy từ bộ tạo dao động nội ngoại thì xung cung cấp cho nhân Cortex đều được lấy từ dầu ra bộ PLL.

# **Chương 2: CẤU TRÚC CHUNG CỦA KIT**

## ***Giới thiệu Kit STM32F103C8T6***

STM32 là một trong những dòng chip phổ biến của ST với nhiều họ thông dụng như F0,F1,F2,F3,F4….. Stm32f103 thuộc họ F1 với lõi là ARM COTEX M3. STM32F103 là vi điều khiển 32 bit, tốc độ tối đa là 72Mhz. Giá thành cũng khá rẻ so với các loại vi điều khiển có chức năng tương tự. Mạch nạp cũng như công cụ lập trình khá đa dạng và dễ sử dụng.

Một số ứng dụng chính: dùng cho driver để điều khiển ứng dụng, điều khiển ứng dụng thông thường, thiết bị cầm tay và thuốc, máy tính và thiết bị ngoại vi chơi game, GPS cơ bản, các ứng dụng trong công nghiệp, thiết bị lập trình PLC, biến tần, máy in, máy quét, hệ thống cảnh báo, thiết bị liên lạc nội bộ…

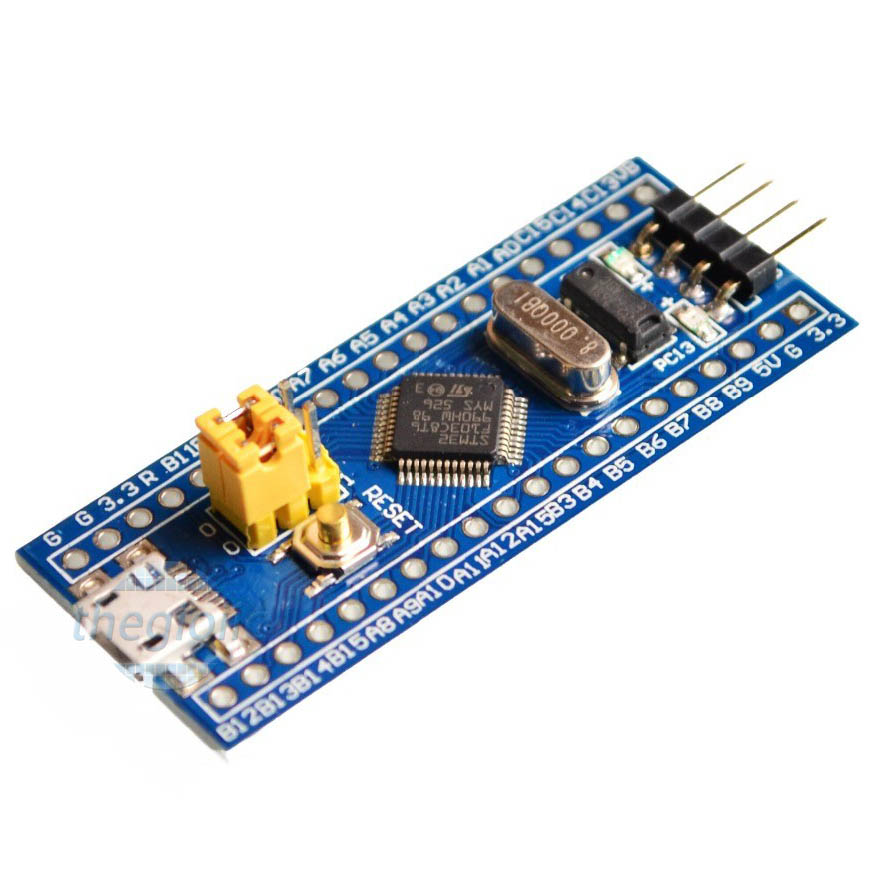
Phần mềm lập trình: có khá nhiều trình biên dịch cho STM32 như IAR Embedded Workbench, Keil C… Ở đây mình sử dụng Keil C nên các bài viết sau mình chỉ đề cập đến Keil C.

Thư viện lập trình: có nhiều loại thư viện lập trình cho STM32 như: STM32snippets, STM32Cube LL, STM32Cube HAL, Standard Peripheral Libraries, Mbed core. Mỗi thư viện đều có ưu và khuyết điểm riêng, ở đây mình xin phép sử dụng Standard Peripheral Libraries vì nó ra đời khá lâu và khá thông dụng, hỗ trợ nhiều ngoại vi và cũng dễ hiểu rõ bản chất của lập trình.

Mạch nạp: có khá nhiều loại mạch nạp như : ULINK, J-LINK , CMSIS-DAP, STLINK… ở đây mình sử dụng Stlink vì giá thành khá rả và debug lỗi cũng tốt.

Board để lập trình: các bạn có thể mua sẵn 1 số kit ra chân đã có sẵn trên thị trường hoặc thiết kế 1 cái board dành riêng cho bản thân mình. Ở đây mình đã thiết kế 1 board đã tích hợp sẵn mạch nạp, mình cũng đã test và chạy khá ổn, debug bằng Keil C khá giống với phiên bản STlink V2 trên thị trường. Nói chung là sử dụng đồ tự làm khá thú vị. Đây là board của mình:

Sơ lược về Board trên:



* + 1 cổng Mini USB dùng để cấp nguồn, nạp cũng như debug.
  + 2 MCU bao gồm 1 MCU nạp và 1 MCU dùng để lập trình.
  + Có chân Output riêng cho các chân mạch nạp trên MCU1.
  + Có chân Output đầy đủ cho các chân MCU2.
  + Chân cấp nguồn ngoài riêng cho MCU2 nếu không sử dụng nguồn từ USB.
  + Thạch anh 32,768khz dùng cho RTC và Backup.
  + Chân nạp dùng cho chế độ nạp boot loader.
  + Nút Reset ngoài và 1 led hiển thị trên chân PB9, 1 led báo nguồn cho MCU2.

- Cấu hình chi tiết của STM32F103C8T6:

* + ARM 32-bit Cortex M3 với clock max là 72Mhz.
  + Bộ nhớ:
    - 64 kbytes bộ nhớ Flash(bộ nhớ lập trình).
    - 20kbytes SRAM.
  + Clock, reset và quản lý nguồn.
    - Điện áp hoạt động 2.0V -> 3.6V.
    - Power on reset(POR), Power down reset(PDR) và programmable voltage detector (PVD).
    - Sử dụng thạch anh ngoài từ 4Mhz -> 20Mhz.
    - Thạch anh nội dùng dao động RC ở mode 8Mhz hoặc 40khz.
    - Sử dụng thạch anh ngoài 32.768khz được sử dụng cho RTC.
  + Trong trường hợp điện áp thấp:
    - Có các mode :ngủ, ngừng hoạt động hoặc hoạt động ở chế độ chờ.
    - Cấp nguồn ở chân Vbat bằng pin để hoạt động bộ RTC và sử dụng lưu trữ data khi mất nguồn cấp chính.
  + 2 bộ ADC 12 bit với 9 kênh cho mỗi bộ.
    - Khoảng giá trị chuyển đổi từ 0 – 3.6V.
    - Lấy mẫu nhiều kênh hoặc 1 kênh.
    - Có cảm biến nhiệt độ nội.
  + DMA: bộ chuyển đổi này giúp tăng tốc độ xử lý do không có sự can thiệp quá sâu của CPU.
    - 7 kênh DMA.
    - Hỗ trợ DMA cho ADC, I2C, SPI, UART.
  + 7 timer.
    - 3 timer 16 bit hỗ trợ các mode IC/OC/PWM.
    - 1 timer 16 bit hỗ trợ để điều khiển động cơ với các mode bảo vệ như ngắt input, dead-time..
    - 2 watdog timer dùng để bảo vệ và kiểm tra lỗi.
    - 1 sysTick timer 24 bit đếm xuống dùng cho các ứng dụng như hàm Delay….
  + Hỗ trợ 9 kênh giao tiếp bao gồm:
    - 2 bộ I2C(SMBus/PMBus).
    - 3 bộ USART(ISO 7816 interface, LIN, IrDA capability, modem control).
    - 2 SPIs (18 Mbit/s).
    - 1 bộ CAN interface (2.0B Active)
    - USB 2.0 full-speed interface
  + Kiểm tra lỗi CRC và 96-bit ID.

# **Chương 3: NHỮNG KIẾN THỨC LIÊN QUAN**

## ***3.1. Giới thiêu sơ lược về protocol USART, module Fingerprint và LCD16x02***

**3.1.1 Sơ lược về lý thuyết USART**

UART - Universal synchronous asynchronous receiver transmitter là một ngoại vi cơ bản và thường dùng trong các quá trình giao tiếp với các module như : Xbee, Wifi, Blutooth…. Khi giao tiếp UART kết hợp với các IC giao tiếp như MAX232CP, SP485EEN…. thì sẽ tạo thành các chuẩn giao tiếp RS232, RS485. Đây là các chuẩn giao tiếp thông dụng và phổ biến trong công nghiệp từ trước đến nay.

Khi ta sử dụng chân UART\_CLK thì giao tiếp UART sẽ trở thành giao tiếp đồng bộ và không dùng sẽ là chuẩn giao tiếp không đồng bộ. Các bạn để ý là với bất cứ 1 chuẩn truyền thông nào, khi có sử dụng 1 chân tín hiệu làm chân CLK thì chuẩn giao tiếp đó sẽ là chuẩn giao tiếp đồng bộ và ngược lại. Ở đây mình chỉ đề cập đến giao tiếp UART không đồng bộ.

Ưu điểm của giao tiếp UART không đồng bộ: tiết kiệm chân vi điều khiển (2 chân), là ngoại vi mà bất kì 1 VĐK nào cũng có, có khá nhiều module, cảm biến dùng UART để truyền nhận data với VĐK. Nhược điểm của loại ngoại vi này là tốc độ khá chậm, tốc độ tối đa tùy thuộc vào từng dòng; quá trình truyền nhận dễ xảy ra lỗi nên trong quá trình truyền nhận cần có các phương pháp để kiểm tra (thông thường là truyền thêm bit hoặc byte kiểm tra lỗi). UART không phải là 1 chuẩn truyền thông, Khi muốn nó là 1 chuẩn truyền thông hoặc truyền data đi xa, chúng ta cần phải sử dụng các IC thông dụng để tạo thành các chuẩn giao tiếp đáng tin cậy như RS485 hay RS232....

Thông thường chúng ta sẽ dùng ngắt nhận UART để nhận dữ liệu vì sử dụng ngắt sẽ tiện lợi, không tốn thời gian chờ cũng như mất dữ liệu.Các tốc độ thường dùng để giao tiếp với máy tính: 600,1200,2400,4800,9600,14400,19200,38400,56000,57600,115200.

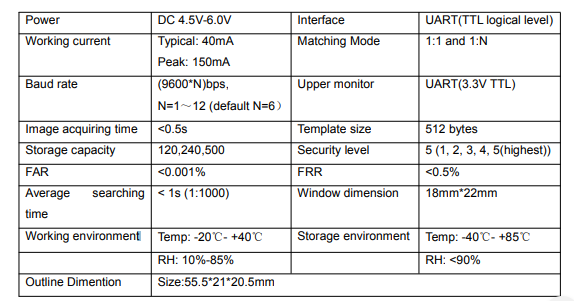
Một số phần mềm giao tiếp với máy tính: hercules\_3-2-5,teraterm, Serial-Oscilloscope-v1.5...Một số modulde dùng để giao tiếp với máy tính: CP2102 USB 2.0, USB ra UART dùng PL2303, USB to UART dùng TTL FT232RL, USB ra UART dùng CH340G…

STM32F103C8 có 3 bộ UART với nhiều mode hoạt động, với nhiều bộ UART ta có thể sử dụng được nhiều ứng dụng với 1 chip điều khiển so với STM8S. Một số tính năng nổi bật như sau:

* + Đầy đủ các tính năng của bộ giao tiếp không đồng bộ.
  + Điều chỉnh baud rate bằng lập trình và tốc độ tối đa lên đến 4.5Mb/s.
  + Độ dài được lập trình là 8 hoặc 9 bit.
  + Cấu hình bit stop hỗ trợ là 1 hoặc 2.
  + Có chân clock nếu muốn chuyển giao tiếp thành đồng bộ.
  + Cấu hình sử dụng 1 dây hoặc 2 dây.a
  + Có bộ DMA nếu muốn đẩy cao thời gian truyền nhận.
  + Bit cho phép truyền nhận riêng biệt.
    1. **Sơ lược về module Fingerprint R305**

R305 là một module đọc dấu vân tay riêng biệt, đặc biệt tốc độ cao làm bộ phận cốt lõi, tương thích cảm biến vân tay khác nhau. Đây là một mô-đun thông minh có thể tự do lấy dấu vân tay, xử lý hình ảnh, vân tay được xác minh, tìm kiếm và lưu trữ và nó có thể hoạt động bình thường mà không cần quản lý có sự tham gia của màn hình trên. Xử lý vân tay bao gồm hai phần: đăng ký vân tay và khớp dấu vân tay (khớp có thể là 1: 1 hoặc 1: N)

Đăng ký vân tay, người dùng cần nhập ngón tay 2-4 lần cho mỗi một ngón tay, xử lý hình ảnh ngón tay với nhiều lần, lưu trữ tạo mẫu trên mô-đun. Khi khớp dấu vân tay, đăng ký và xử lý hình ảnh dấu vân tay đã xác minh Và sau đó khớp với mô-đun (nếu khớp với các mẫu chỉ định trên mô-đun, xác minh dấu vân tay, Phương thức đối sánh 1: 1; hệ thống được đặt tên 1: N) sẽ trả về kết quả khớp, thành công hay thất bại.



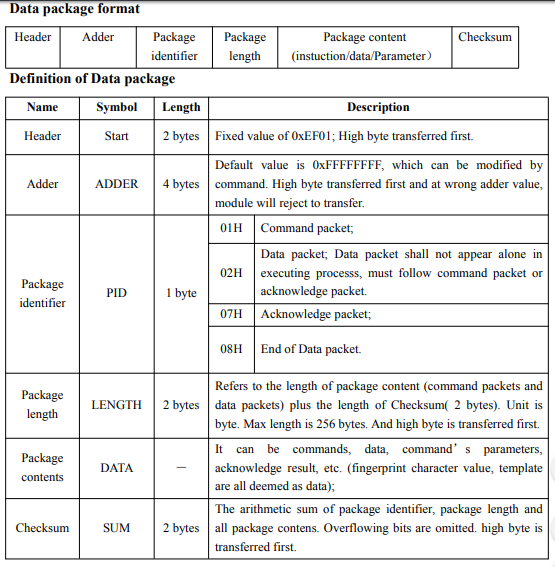
**Serial communication protocol**

Chế độ là giao tiếp nối tiếp không đồng bộ semiduplex. Và tốc độ truyền mặc định là 57600bps. Truyền định dạng khung là 10 bit: bit bắt đầu mức thấp, dữ liệu 8 bit với LSB đầu tiên và bit kết thúc. Không có bit kiểm tra.

Communication of protocol

1. Data package format

Khi giao tiếp, việc truyền và nhận lệnh/dữ liệu/kết quả được bọc tất cả trong gói dữ liệu sau:



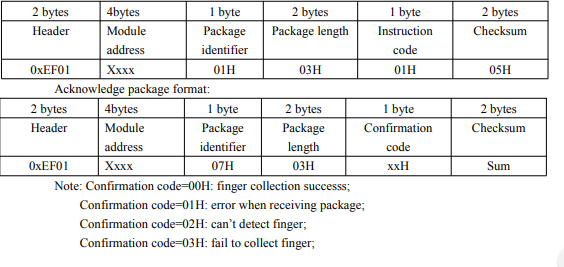
**Các tập lệnh giao tiếp**

**1. Lấy hình ảnh vân tay GetImage**

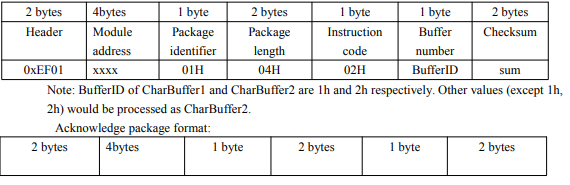
Mô tả : phát hiện vân tay và lưu trữ hình ảnh vân tay được phát hiện trong ImageBuffer khi trả lại mã xác nhận thành công; Nếu không có ngón tay, mã xác nhận được trả về sẽ có thể phát hiện ra vân tay.

Tham số đầu vào : none

Trả về tham số : Mã xác nhận (1 byte) Mã khởi động : 01H Định dạng gói (hoặc lệnh)

**2**. **Để tạo tệp ký tự từ hình ảnh Genchar**

Mô tả: để tạo tệp ký tự từ hình ảnh ngón tay ban đầu trong ImageBuffer và lưu trữ tệp trong CharBuffer1 hoặc CharBuffer2. Tham số đầu vào: BufferID (số bộ đệm tệp ký tự) Trả về tham số: Mã xác nhận (1 byte) Mã khởi động: 02H

Định dạng gói (hoặc lệnh):

Note:

Confirmation code=00H: generate character file complete;

Confirmation code=01H: error when receiving package;

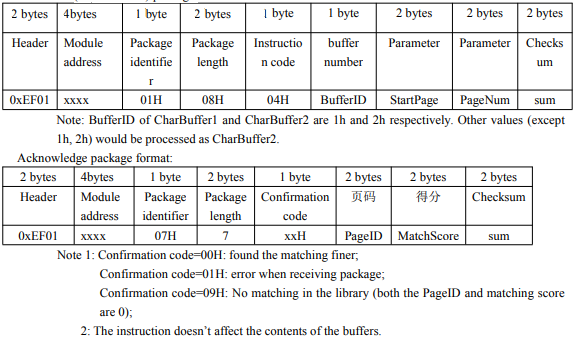
Confirmation code=06H: fail to generate character file due to the over-disorderly fingerprint image;

Confirmation code=07H: fail to generate character file due to lackness of character point or over-smallness of fingerprint image;

Confirmation code=15H: fail to generate the image for the lackness of valid primary image

**3. Để tìm kiếm thư viện ngón tay Search**

Mô tả: để tìm kiếm toàn bộ thư viện ngón tay cho mẫu phù hợp với mẫu trong CharBuffer1 hoặc CharBuffer2. Khi tìm thấy, PageID sẽ được trả lại. Tham số đầu vào: BufferID, StartPage (địa chỉ bắt đầu tìm kiếm), PageNum numbers số tìm kiếm

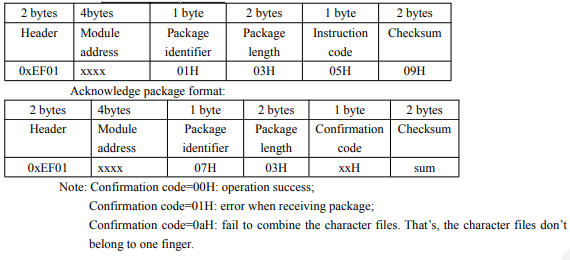
 Tham số trả về: Mã xác nhận (1 byte), PageID (vị trí mẫu phù hợp code Mã lệnh: 04H Lệnh (hoặc lệnh) định dạng gói:

**4. Để tạo mẫu RegModel**

Mô tả: Để kết hợp thông tin của các tệp ký tự từ CharBuffer1 và CharBuffer2 và tạo một mẫu được vuốt lại trong cả CharBuffer1 và CharBuffer2.

Nhập tham số vào: không có

Trả về tham số mã xác nhận (1 byte) Mã xác định: 05H Định dạng gói (hoặc lệnh):



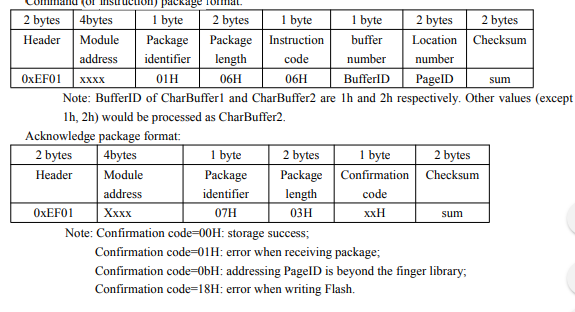
**5. Để lưu mẫu Store**

Mô tả: để lưu mẫu của bộ đệm được chỉ định (Buffer1 / Buffer2) tại vị trí được

chỉ định của thư viện Flash.

Tham số đầu vào: BufferID (số bộ đệm), PageID location Vị trí flash của mẫu, hai byte có byte cao phía trước và byte thấp phía sau

Trả về tham số: Mã xác nhận (1 byte) Mã định dạng: 06H Định dạng gói lệnh (hoặc lệnh):

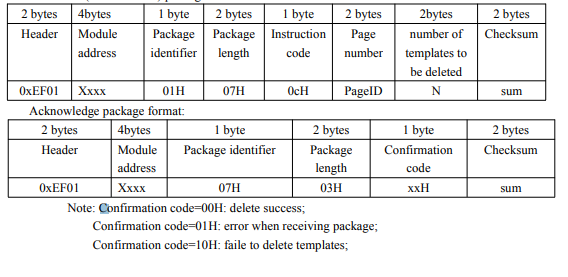


**6. Để xóa mẫu DeletChar**

Mô tả: để xóa một phân đoạn (N) mẫu của thư viện Flash bắt đầu từ vị trí đã chỉ định (hoặc PageID);

Tham số đầu vào: PageID (số mẫu trong Flash), N (số mẫu cần xóa)

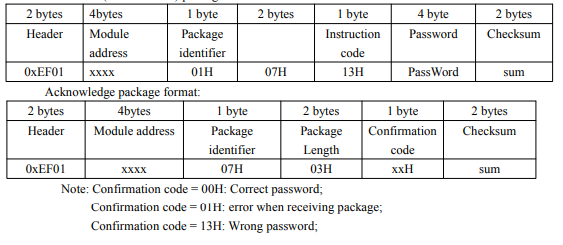
Trả về tham số: Mã xác nhận (1 byte) Mã định dạng: 0cH Định dạng gói lệnh (hoặc lệnh):



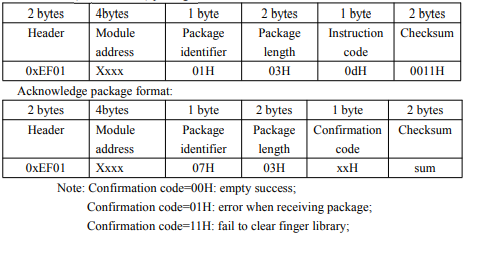
**7. Xác minh mật khẩu VfyPwd**

Mô tả: Xác minh mật khẩu bắt tay Mô-đun. (Tham khảo 4.6 để biết chi tiết) Tham số đầu vào: PassWord (4 byte)

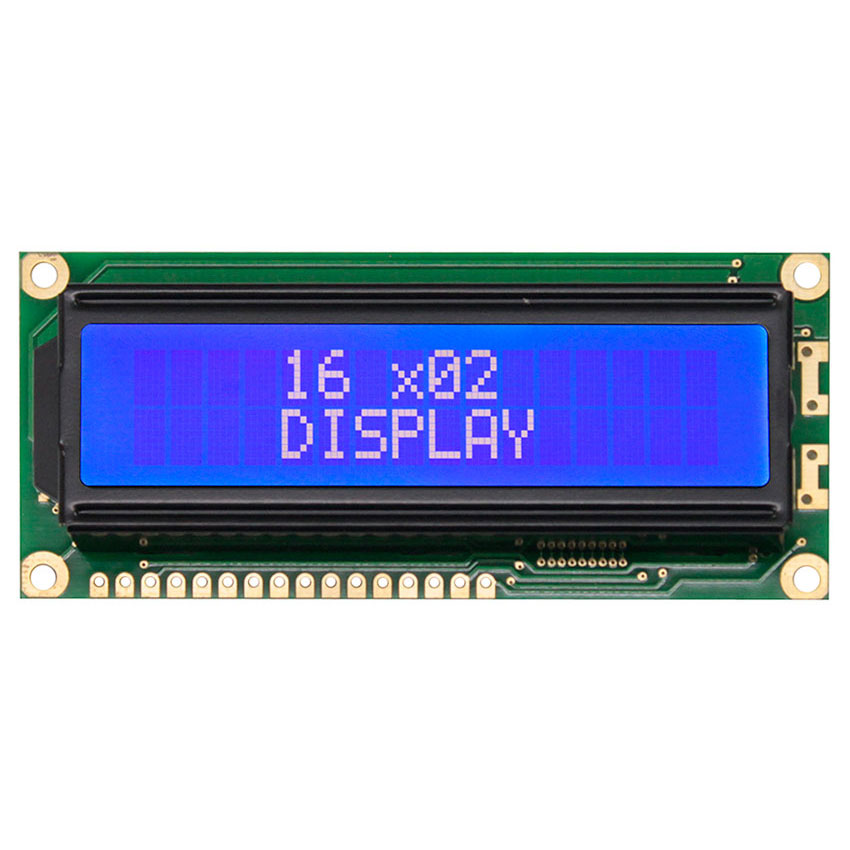
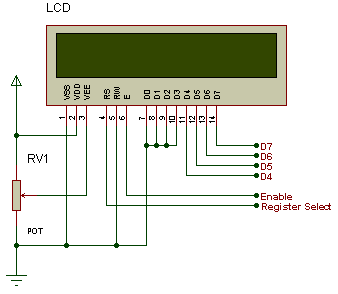
Tham số trả về: Mã xác nhận (1 byte) Mã lệnh: Định dạng gói 13H Lệnh (hoặc lệnh):



Đa số các SD card hiện nay đều sử dụng SPI (Serial Peripheral Interface). SPI à môt giao thức đươc sử dung rộng rãi và nó có sẵn trong hầu hết các vi điều khiển hiên nay, đươc sử dung trong hầu hầu các hê thống nhúng chi phí thấp. Phạm vi điên áp àm viêc của họ là 2.7V đến 3.6V, đươc nêu trong thanh ghi OCR.



* + 1. **Giao tiếp với lcd16x02**



Chân kết nối LCD được khởi tạo trong hàm lcd\_txt.h ở thư viện lcd.c

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LCD | VSS | VDD | RS | E | D4 | D5 | D6 | D7 |
| Kit STM32F1 | GND | 5V | A4 | A5 | B6 | B7 | B0 | B1 |

1. VSS: tương đương với GND - cực âm
2. VDD: tương đương với VCC - cực dương (5V)
3. Constrast Voltage (Vo): điều khiển độ sáng màn hình
4. Register Select (RS): điều khiển địa chỉ nào sẽ được ghi dữ liệu
5. Read/Write (RW): đọc (read mode) hay ghi (write mode) dữ liệu
6. Enable pin: Cho phép ghi vào LCD
7. D0 - D7: 8 chân dư liệu, mỗi chân sẽ có giá trị HIGH hoặc LOW nếu đang ở chế độ đọc (read mode) và lcd sẽ nhận giá trị HIGH hoặc LOW nếu đang ở chế độ ghi (write mode)
8. Backlight (Backlight Anode (+) và Backlight Cathode (-)): Tắt bật đèn màn hình LCD.

# **Chương 4: ỨNG DỤNG**

## **4.1 Ứng dụng trong đời sống**

Việc ứng dụng công nghệ nhận dạng vân tay đã có từ rất lâu. Tuy nhiên, trong khoảng một thời gian dài con người chỉ thực hiện việc đối sánh giữa hai dấu vân tay bằng kỹ thuật truyền thống mang nặng tính thủ công, các kết quả của lĩnh vực này gần như không được ứng dụng trong các lĩnh vực dân sự thông thường của đời sống mà chủ yếu được sử dụng trong lĩnh vực hình sự. Với sự phát triển ngày càng nhanh chóng của ngành công nghệ điện toán thì việc sử dụng dấu vân tay để nhận dạng được áp dụng rộng rãi trong đời sống bằng Hệ thống nhận dạng vân tay tự động (AFIS). Cùng với sự phát triển mạnh các sản phẩm phần mềm nhúng và một thị trường thiết bị nhúng vô cùng to lớn. Theo cách đối sánh vân tay truyền thống, để kiểm chứng hai dấu vân tay có giống nhau hay không thì phải dùng kính lúp để đối chiếu từng đường vân. Trang nhận dạng vân tay lên các thiết bị nhúng đã mang lại nhiều lợi ích và hiệu quả không những cho nhà phát triển công nghệ này lên thiết bị nhúng mà còn cho xã hội. Kể từ đây, công nghệ Nhận dạng vân tay đề cập trong luận văn này chính là Hệ thống nhận dạng vân tay tự động trên hệ thống nhúng. Công nghệ này không những được ứng dụng trong lĩnh vực hình sự mà còn được ứng dụng đa dạng trong lĩnh vực dân sự, thương mại, … cụ thể là: việc xác nhận nhân thân của cá nhân khi truy cập mạng, hồ sơ cá nhân, khóa phòng trộm, thẻ ngân hàng, hệ thống chấm công, hệ thống bảo mật, …

## **4.2 Mô tả ứng dụng**

Ứng dụng sẽ có 2 nút nhấn để ra lệnh : Kiển tra vân tay và Thêm mới vân tay:

- Nút nhấn 1 được kết nối với chân PA1 dùng để thực thi nhiệm vụ dò vân tay có trong bộ nhớ của module và thông báo trên lcd16x02.

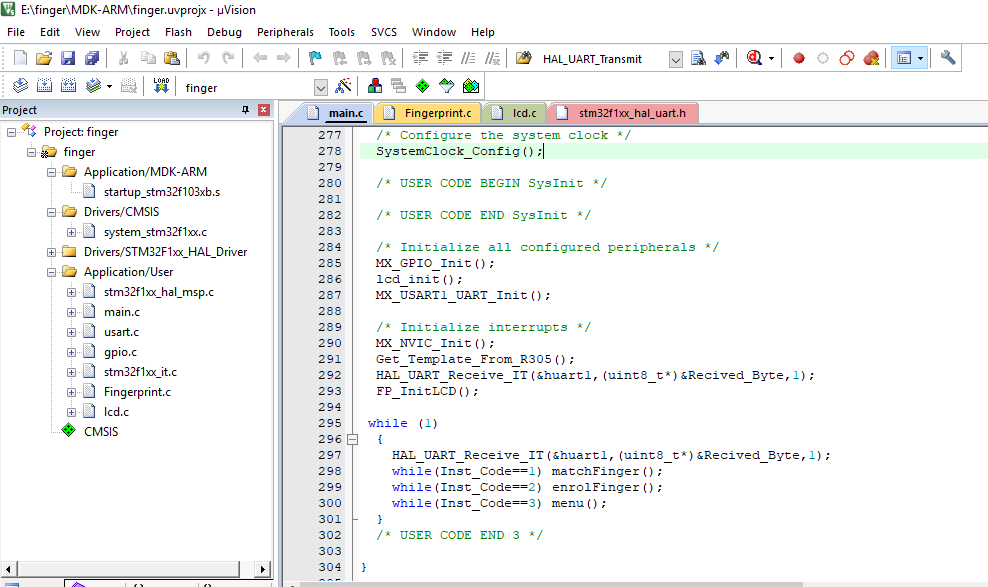
- Nút nhấn 2 được kết nối với chân PA2 dùng để thực thi nhiệm vụ thêm mới vân tay vào trong bộ nhớ của module với id được chỉ định trong source code

## **4.3 Cách nạp chương trình vào KIT**

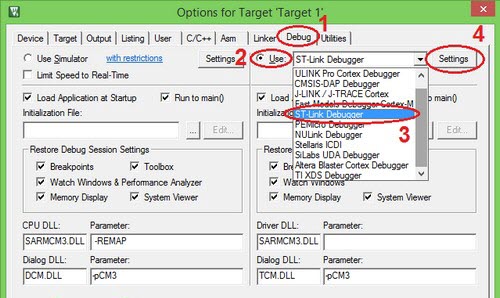
Sử dụng mạch nạp ST-LinK V2 đê nạp chương trình vào KIT

Bước 1: Gắn ST-LinK V2 vào cổng USB của máy tính.

Bước 2: Chạy phần mềm KeilC và bật chương trình finger.uvprojx

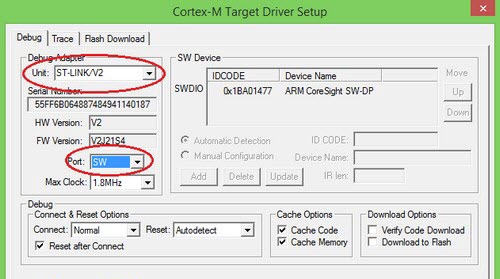


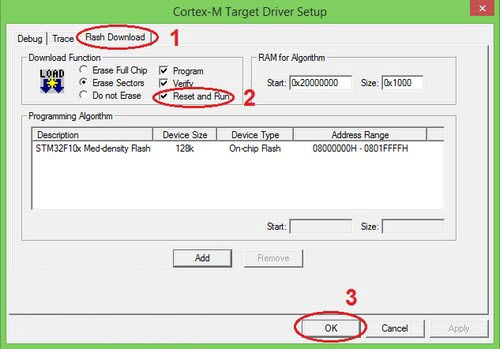
Bước 3: Chọn Tab Debug -> Use: ST-Link Debugger -> Settings



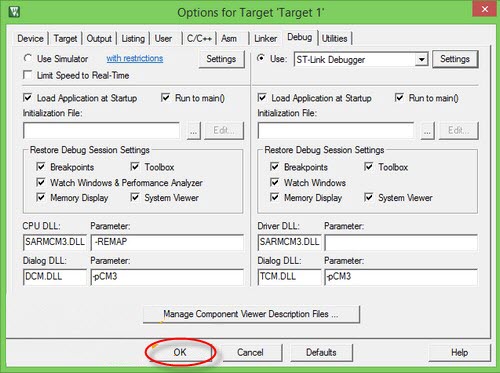
Bước 4: Kiểm tra như hình.

Nếu xuất hiện đầy đủ nội dung cả 2 bên “Debug Adapter” và “SW Device” thì các bước bạn thiết lập mới đúng, nếu không xin vui lòng kiểm tra lại Driver và kết nối phần cứng.

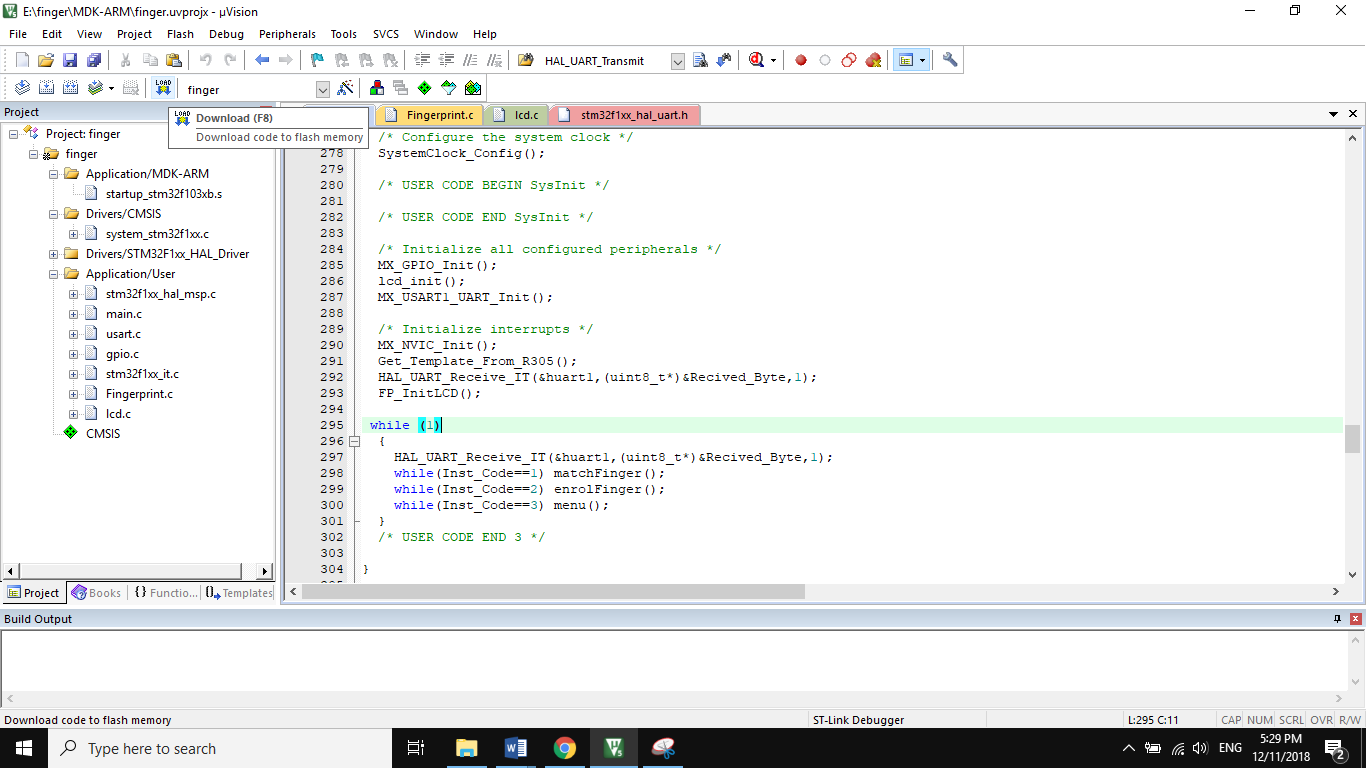


Bước 5: Chuyển Tab Flash Download, tick vào ô “Reset and Run”, chọn OK

Bước 6: Chọn OK lần nữa để xác nhận các thiết lập



Bước 7: Vậy là đã thiết lập xong mạch nạp, chọn như hình (hoặc bấm F8) để tiến hành nạp code xuống mạch.





Vậy hoàn tất việc nạp chương trình cho KIT

## **4.4. Chương trình**

* + 1. **Chương trình chính**

|  |
| --- |
| int main(void)  {  /\* USER CODE BEGIN 1 \*/  /\* USER CODE END 1 \*/  /\* MCU Configuration----------------------------------------------------------\*/  /\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/  HAL\_Init();  /\* USER CODE BEGIN Init \*/  /\* USER CODE END Init \*/  /\* Configure the system clock \*/  SystemClock\_Config();  /\* USER CODE BEGIN SysInit \*/  /\* USER CODE END SysInit \*/  /\* Initialize all configured peripherals \*/  MX\_GPIO\_Init();  lcd\_init();  MX\_USART1\_UART\_Init();    /\* Initialize interrupts \*/  MX\_NVIC\_Init();  Get\_Template\_From\_R305();  HAL\_UART\_Receive\_IT(&huart1,(uint8\_t\*)&Recived\_Byte,1);  FP\_InitLCD();    while (1)  {  HAL\_UART\_Receive\_IT(&huart1,(uint8\_t\*)&Recived\_Byte,1);  while(Inst\_Code==1) matchFinger();  while(Inst\_Code==2) enrolFinger();  while(Inst\_Code==3) menu();  }  /\* USER CODE END 3 \*/  } |

* + 1. **Chương trình con**

|  |
| --- |
| void enrolFinger()  {  lcd\_clear();  lcd\_puts(1,0,(int8\_t \*)"Add Finger ");  HAL\_Delay(500);    lcd\_clear();  lcd\_puts(1,0,(int8\_t \*)"Add Finger ");  HAL\_Delay(500);    lcd\_clear();  lcd\_puts(1,0,(int8\_t \*)"Add Finger ");  HAL\_Delay(500);  lcd\_clear();  lcd\_puts(0,0,(int8\_t \*)"Put your finger ");  HAL\_Delay(3000);  if(sendcmd2fp(DETECT)) {  if(sendcmd2fp(IMG2CHAR)){  lcd\_puts(0,0,(int8\_t \*)"Finger Detected ");  HAL\_Delay(2000);  lcd\_puts(1,0,(int8\_t \*)"Put finger again");  HAL\_Delay(2000);  if(sendcmd2fp(DETECT)){  if(sendcmd2fp(IMG2CHAR2)){  lcd\_puts(0,0,(int8\_t \*)"Finger Detected ");  HAL\_Delay(2000);    if(sendcmd2fp(CREATE\_MODEL)){  Store\_template[11]= (id>>8) ;  Store\_template[12]= id & 0xff;  Store\_template[14]= 14+id;    if(sendcmd2fp(STORE\_MODEL)){  if(buf[8] == 0x3 && buf[11] == 0x0A){  sprintf(buflcd,"Finger ID: %d ",id);  lcd\_puts(0,0,(int8\_t \*)"Finger Saved "); lcd\_puts(1,0,(int8\_t \*)&buflcd);    HAL\_Delay(1000);  id++;  }  else {  lcd\_puts(0,0,(int8\_t \*)"Finger Stored ");  lcd\_puts(1,0,(int8\_t \*)"Failed ");  }  }  }  }  }  }  }  HAL\_Delay(5000);  indexx=0;  Inst\_Code=3;  }  void matchFinger()  {  lcd\_clear();  lcd\_puts(0,0,(int8\_t \*)"Search Finger ");  HAL\_Delay(500);    lcd\_clear();  lcd\_puts(0,0,(int8\_t \*)"Search Finger ");  HAL\_Delay(500);    lcd\_clear();  lcd\_puts(0,0,(int8\_t \*)"Search Finger ");  HAL\_Delay(500);  lcd\_puts(0,0,(int8\_t \*)"Put your finger ");  HAL\_Delay(3000);  HAL\_UART\_Receive\_IT(&huart1,(uint8\_t\*)&Recived\_Byte,1);  if(sendcmd2fp(DETECT )) {  if(sendcmd2fp(IMG2CHAR)) {  if(sendcmd2fp(SEACRCH)) {  uint16\_t id= buf[10];  id<<=8;  id+=buf[11];  uint16\_t score=buf[12];  score<<=8;  score+=buf[13];  buf1[0]=(uint32\_t)id;  buf1[1]=(uint32\_t)score;  if(score >50) {  sprintf(buflcd,"ID: %d Score: %d",buf1[0],buf1[1]);  lcd\_puts(0,0,(int8\_t \*)"Finger Found ");  lcd\_puts(1,0,(int8\_t \*)&buflcd);  }  else {  lcd\_clear();  lcd\_puts(0,0,(int8\_t \*)"Finger Not Found");  }  }  }  }  HAL\_Delay(5000);  indexx=0;  Inst\_Code=3;  }  int sendcmd2fp(int pack)  {  uint8\_t res=ERROR;  for(int i=0;i<sizeof(buf);i++)  {  buf[i]=0;  }  indexx=0;    switch(pack)  {  case PASSPACK: PassPack();  break;  case DETECT : Get\_Image\_From\_R305();  break;  case IMG2CHAR : Convert\_To\_Char();  break;  case IMG2CHAR2 : Convert\_To\_Char2();  break;  case CREATE\_MODEL : Generate\_Template();  break;  case STORE\_MODEL : Store\_Finger();  break;  case SEACRCH : Search\_Finger();  break;  default: break;  }  HAL\_UART\_Receive\_IT(&huart1,(uint8\_t\*)&Recived\_Byte,1);  HAL\_Delay(1000);  if(flag == 1)  {  HAL\_UART\_Receive\_IT(&huart1,(uint8\_t\*)&Recived\_Byte,1);  if(buf[0] == 0xEF && buf[1] == 0x01) {  res=SUCCESS;  }  else {  res=ERROR;  }  indexx=0;  flag=0;  }  return res;  } |

* + 1. **Thư viện mẫu**

Base on user manual of R305 module

|  |
| --- |
| #include "Fingerprint.h"  const uint8\_t passPack[]={0xEF, 0x1, 0xFF, 0xFF, 0xFF, 0xFF, 0x1, 0x0, 0x7, 0x13, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0};  const uint8\_t Get\_Image\_R305[] = {0xEF,0x01, 0xFF, 0xFF, 0xFF, 0xFF, 0x01, 0x00, 0x03, 0x01, 0x00, 0x05};//12  const uint8\_t Get\_Template\_Number[] = {0xEF, 1, 0xFF, 0xFF, 0xFF, 0xFF, 1, 0, 0x03, 0x1D, 0, 0x21};//12  const uint8\_t ConvertImageToChar[] = {0xEF, 1, 0xFF, 0xFF, 0xFF, 0xFF, 1, 0, 4, 2, 1, 0, 0x08};//13  const uint8\_t ConvertImageToChar2[] = {0xEF, 1, 0xFF, 0xFF, 0xFF, 0xFF, 1, 0, 4, 2, 2, 0, 0x09};//13  const uint8\_t Gen\_Template[] = {0xEF, 1, 0xFF, 0xFF, 0xFF, 0xFF, 1, 0, 3, 5, 0, 9};//12  uint8\_t Store\_template[] ={0xEF,0x1,0xFF,0xFF,0xFF,0xFF,0x1,0x0,0x6,0x6,0x1,0x0,0x1,0x0,0xE};//15  uint8\_t Delete\_Template[] = {0xEF, 1, 0xFF, 0xFF, 0xFF, 0xFF, 1, 0, 7, 0x0C, 0, 0, 0, 1, 0, 0x15};//16  const uint8\_t Delete\_all\_Templates[] = {0xEF, 1, 0xFF, 0xFF, 0xFF, 0xFF, 1, 0, 3, 0x0D, 0, 0x11};//12  const uint8\_t Read\_template\_from\_Lib[] = {0xEF, 1, 0xFF, 0xFF, 0xFF, 0xFF, 1, 0, 6, 7, 0, 0, 0, 0, 0x0E};//15  const uint8\_t Verify\_Template[] = {0xEF, 1, 0xFF, 0xFF, 0xFF, 0xFF, 1, 0, 3, 3, 0, 0x07};//12  const uint8\_t Gen\_Rand\_Numb[] = {0xEF,1,0xFF,0xFF,0xFF,0xFF,1,0,3,0x14,0,0x18};//12  //const uint8\_t Search\_library[] = {0xEF, 0x1, 0xFF, 0xFF, 0xFF, 0xFF, 0x1, 0x0, 0x8, 0x4, 0x0, 0x0, 0x0, 0x0, 0xF4, 0x0, 0x1};//17  const uint8\_t Search\_library[] = {0xEF, 0x1, 0xFF, 0xFF, 0xFF, 0xFF, 0x1, 0x0, 0x8, 0x1B, 0x1, 0x0, 0x0, 0x0, 0xA3, 0x0, 0xC8};  unsigned int random\_seed = 6;  unsigned char Length;    unsigned char i;  unsigned char Recived\_Byte;  void PassPack(void)  {  for(i=0;i<16;i++)  {  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&passPack[i],1,HAL\_MAX\_DELAY);//(UART1\_BASE,Get\_Image\_R305[i]);    }  }  void Get\_Image\_From\_R305(void)  {  for(i=0;i<12;i++)  {  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&Get\_Image\_R305[i],1,HAL\_MAX\_DELAY);//(UART1\_BASE,Get\_Image\_R305[i]);    }  }  void Convert\_To\_Char(void)  {  for(i=0;i<13;i++)  {  // ConvertImageToChar[10]  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&ConvertImageToChar[i],1,HAL\_MAX\_DELAY); //(UART1\_BASE,ConvertImageToChar[i]);    }  }  void Convert\_To\_Char2(void)  {  for(i=0;i<13;i++)  {  // ConvertImageToChar[10]  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&ConvertImageToChar2[i],1,HAL\_MAX\_DELAY); //(UART1\_BASE,ConvertImageToChar2[i]);  }  }  void Get\_Template\_From\_R305(void)  {  for(i=0;i<15;i++)  {  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&Read\_template\_from\_Lib[i],1,HAL\_MAX\_DELAY); //(UART1\_BASE,Read\_template\_from\_Lib[i]);    }  }  void Match\_Buffers()  {  for(i=0;i<12;i++)  {  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&Verify\_Template[i],1,HAL\_MAX\_DELAY); //(UART1\_BASE,Verify\_Template[i]);  }  }  void R305\_Get\_Random\_Number()  {  for(i=0;i<12;i++)  {  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&Gen\_Rand\_Numb[i],1,HAL\_MAX\_DELAY); //(UART1\_BASE,Gen\_Rand\_Numb[i]);  }  }  int rand()  {    random\_seed = random\_seed \* 1103515245 +12345;  //FlashProgram(random\_seed & 1FF, 0x28800, 4);  return (unsigned int)(random\_seed / 65536) % 32768;  }  void Search\_Finger()  {  for(i=0;i<17;i++)  {  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&Search\_library[i],1,1); //(UART1\_BASE,Search\_library[i]);    }  }  void Generate\_Template()  {  for(i=0;i<12;i++)  {  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&Gen\_Template[i],1,1); //(UART1\_BASE,Gen\_Template[i]);  }  }  void Store\_Finger()  {      for(i=0;i<15;i++)  {  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&Store\_template[i],1,1);//(UART1\_BASE,Store\_template[i]);    }  }  void Store\_Finger\_wID(unsigned char ID)  {  Store\_template[14] = 0x0E;  Store\_template[12] = ID;  Store\_template[14] = 0x0E + ID;  for(i=0;i<15;i++)  {  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&Store\_template[i],1,1);//(UART1\_BASE,Store\_template[i]);    }  }  void Get\_Current\_Template\_Number()  {  for(i=0;i<12;i++)  {  HAL\_UART\_Transmit(&huart1,(uint8\_t\*)&Get\_Template\_Number[i],1,HAL\_MAX\_DELAY);//(UART1\_BASE,Get\_Template\_Number[i]);    }  }  void Delete\_All\_Fingerprints()  {  for(i=0;i<12;i++)  {  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&Delete\_all\_Templates[i],1,1); //(UART1\_BASE,Delete\_all\_Templates[i]);    }  }  void Delete\_Template\_Func(unsigned char ID)  {  Delete\_Template[11] = ID;  Delete\_Template[15] = 0x15+ID;  for(i=0;i<16;i++)  {  HAL\_UART\_Transmit(&huart1, (uint8\_t\*)&Delete\_Template[i],1,1); //(UART1\_BASE,Delete\_Template[i]);  }  } |

* + 1. **Hàm thực thi ngắt và hàm con của lcd**

|  |
| --- |
| void EXTI1\_IRQHandler(void)  {  /\* USER CODE BEGIN EXTI1\_IRQn 0 \*/  /\* USER CODE END EXTI1\_IRQn 0 \*/  Inst\_Code=1;  HAL\_GPIO\_EXTI\_IRQHandler(GPIO\_PIN\_1);  /\* USER CODE BEGIN EXTI1\_IRQn 1 \*/  /\* USER CODE END EXTI1\_IRQn 1 \*/  }  /\*\*  \* @brief This function handles EXTI line2 interrupt.  \*/  void EXTI2\_IRQHandler(void)  {  /\* USER CODE BEGIN EXTI2\_IRQn 0 \*/  Inst\_Code=2;  /\* USER CODE END EXTI2\_IRQn 0 \*/  HAL\_GPIO\_EXTI\_IRQHandler(GPIO\_PIN\_2);  /\* USER CODE BEGIN EXTI2\_IRQn 1 \*/  /\* USER CODE END EXTI2\_IRQn 1 \*/  }  void lcd\_init(void)  {  HAL\_Delay(30);  PIN\_LOW(D4\_PORT,D4\_PIN);  PIN\_HIGH(D5\_PORT,D5\_PIN);  PIN\_LOW(D6\_PORT,D6\_PIN);  PIN\_LOW(D7\_PORT,D7\_PIN);  PIN\_LOW(RS\_PORT,RS\_PIN);  PIN\_HIGH(EN\_PORT,EN\_PIN);  PIN\_LOW(EN\_PORT,EN\_PIN);  lcd\_write(0,0x28);  lcd\_write(0,0x0c);  lcd\_write(0,0x06);  lcd\_write(0,0x01);  }  /\*--------------- Write To LCD ---------------\*/  void lcd\_write(uint8\_t type,uint8\_t data)  {  HAL\_Delay(2);  if(type)  {  PIN\_HIGH(RS\_PORT,RS\_PIN);  }else  {  PIN\_LOW(RS\_PORT,RS\_PIN);  }  //Send High Nibble  if(data&0x80)  {  PIN\_HIGH(D7\_PORT,D7\_PIN);  }else  {  PIN\_LOW(D7\_PORT,D7\_PIN);  }  if(data&0x40)  {  PIN\_HIGH(D6\_PORT,D6\_PIN);  }else  {  PIN\_LOW(D6\_PORT,D6\_PIN);  }  if(data&0x20)  {  PIN\_HIGH(D5\_PORT,D5\_PIN);  }else  {  PIN\_LOW(D5\_PORT,D5\_PIN);  }  if(data&0x10)  {  PIN\_HIGH(D4\_PORT,D4\_PIN);  }else  {  PIN\_LOW(D4\_PORT,D4\_PIN);  }  PIN\_HIGH(EN\_PORT,EN\_PIN);  PIN\_LOW(EN\_PORT,EN\_PIN);  //Send Low Nibble  if(data&0x08)  {  PIN\_HIGH(D7\_PORT,D7\_PIN);  }else  {  PIN\_LOW(D7\_PORT,D7\_PIN);  }  if(data&0x04)  {  PIN\_HIGH(D6\_PORT,D6\_PIN);  }else  {  PIN\_LOW(D6\_PORT,D6\_PIN);  }  if(data&0x02)  {  PIN\_HIGH(D5\_PORT,D5\_PIN);  }else  {  PIN\_LOW(D5\_PORT,D5\_PIN);  }  if(data&0x01)  {  PIN\_HIGH(D4\_PORT,D4\_PIN);  }else  {  PIN\_LOW(D4\_PORT,D4\_PIN);  }  PIN\_HIGH(EN\_PORT,EN\_PIN);  PIN\_LOW(EN\_PORT,EN\_PIN);  }  void lcd\_puts(uint8\_t x, uint8\_t y, int8\_t \*string)  {  //Set Cursor Position  #ifdef LCD16xN //For LCD16x2 or LCD16x4  switch(x)  {  case 0: //Row 0  lcd\_write(0,0x80+0x00+y);  break;  case 1: //Row 1  lcd\_write(0,0x80+0x40+y);  break;  case 2: //Row 2  lcd\_write(0,0x80+0x10+y);  break;  case 3: //Row 3  lcd\_write(0,0x80+0x50+y);  break;  }  #endif  #ifdef LCD20xN //For LCD20x4  switch(x)  {  case 0: //Row 0  lcd\_write(0,0x80+0x00+y);  break;  case 1: //Row 1  lcd\_write(0,0x80+0x40+y);  break;  case 2: //Row 2  lcd\_write(0,0x80+0x14+y);  break;  case 3: //Row 3  lcd\_write(0,0x80+0x54+y);  break;  }  #endif  while(\*string)  {  lcd\_write(1,\*string);  string++;  }  }  void lcd\_clear(void)  {  lcd\_write(0,0x01);  } |

# **KẾT LUẬN**

1. **Kết quả đạt được**

* Hiểu được cấu trúc, chức năng cơ bản của bộ vi xử lý ARM Cortex M3
* Hiểu được một số thành phần của KIT ARM STM32F103
* Hiểu được cấu trúc bộ cảm biến vân tay R305 và cách giao tiếp giữa cảm biến vân tay R305 và KIT
* Biết cách nhúng Code vào KIT

1. **Ưu, nhược điểm**
2. **Ưu điểm**

* Tài liệu về STM32 dễ tìm kiếm, dễ nghiên cứu
* Các hướng dẫn về STM32 nhiều, đa dạng
* Tốc độ xử lý nhanh, ổn định
* Tài nguyên trên KIT lớn, phù hợp với các ứng dụng khác nhau

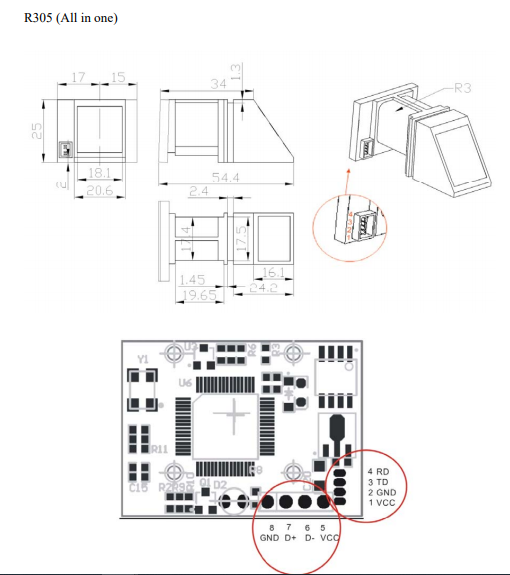
1. **Nhược điểm**

* Nhiều thanh ghi, cấu trúc mỗi thanh ghi là khác nhau nên gây khó cho người lập trình
* Vì là ngôn ngữ mới nên còn nhiều hàm không biết

1. **Hướng phát triển**

* Nâng cấp ứng dụng trở thành 1 ứng dụng bảo mật khóa cửa
* Bảo mật cho laptop, điện thoại,…
* Sử dụng cho máy chấm công, kiểm soát ra vào của công nhân viên

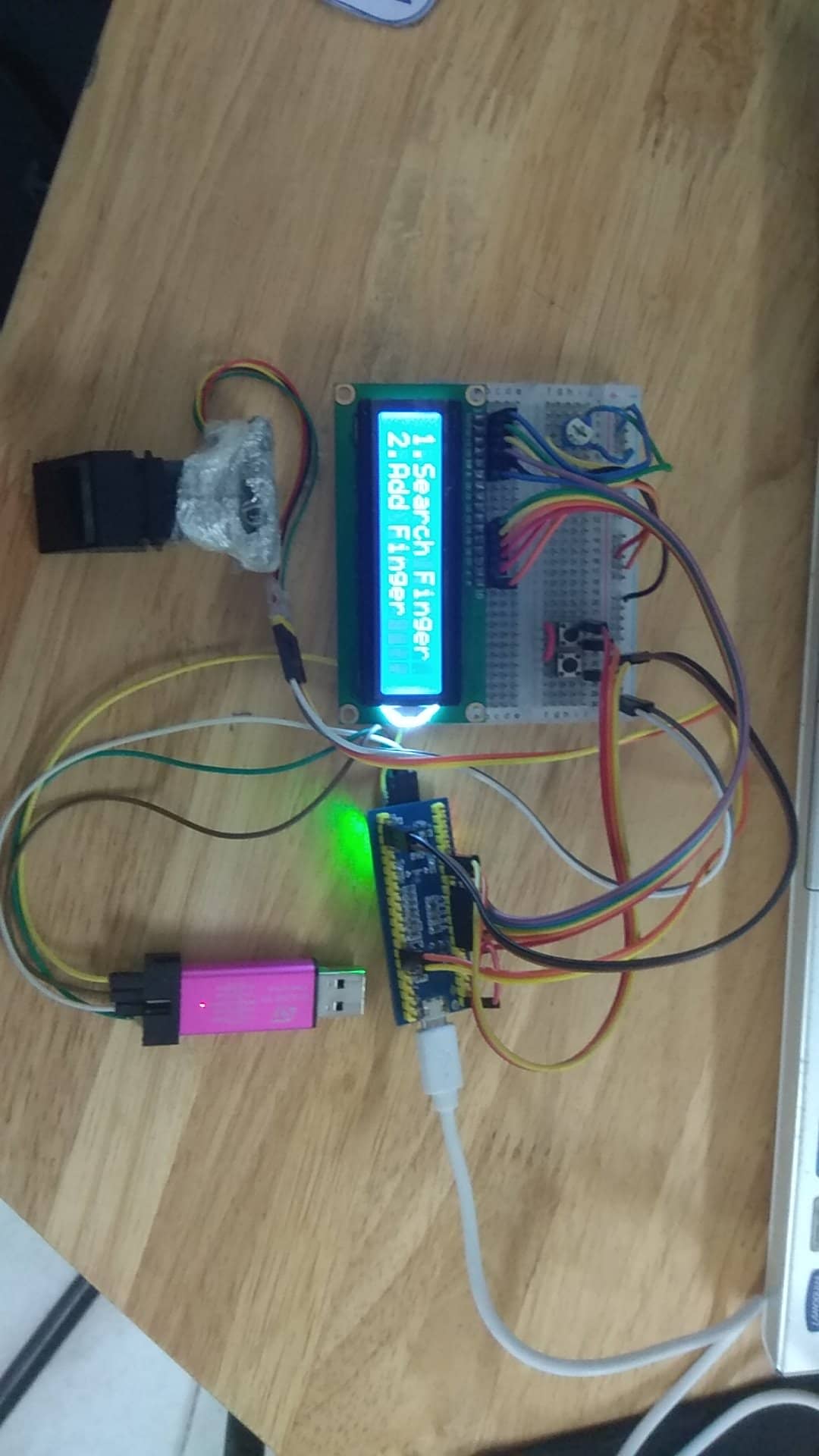
# **PHẦN BỔ SUNG**

* 1. **Sơ đồ nối module đọc vân tay với STM32**

Sơ đồ nối dây với module R305

|  |  |
| --- | --- |
| STM32F103C8T6 Kit | Module Fingerprint |
| PA9 – USART1-RX | TXD |
| PA10–USART1- TX | RXD |
| 5V | VCC |
| G (GND) | GND |

# **Mô hình thực tế**



# **Giải thích thư viện và hàm con**

|  |  |  |
| --- | --- | --- |
| **Thư viện** | **Hàm** | **Tham số, chức năng** |
| stm32f4xx\_hal.h |  | Thư viện của hãng cung cấp để khởi tạo các khối chức năng của chip. Chứa tất cả định nghĩa thanh ghi ngoại vi, bit; định nghĩa và lập bản đồ bộ nhớ cho thiết bị STM32F4xx |
| Lcd.c | void lcd\_write(uint8\_t type,uint8\_t data); | Tham số truyền vào :  - type unsigned int 8 bit  - data unsigned int 8 bit  Chức năng :Truyền data tới lcd16x02 |
|  | void lcd\_puts(uint8\_t x, uint8\_t y, int8\_t \*string); | Tham số truyền vào :  - x unsigned int 8 bit : vị trí hàng của lcd  0<x<2  - y unsigned int 8 bit : vị trí cột của lcd 0<y<16  - \*string : chuôi kí tự muốn hiển thị  Chức năng : truyền chuỗi string để hiển thị lên lcd |
| Main.c | static void MX\_NVIC\_Init(void) | Khởi tao các vectơ ngắt NVIC cho 2 nút nhấn, PA1 ( EXT1),PA2 (EXT2) |
|  | void EnrollFinger(void) | Chương trình con thực hiện việc thêm vân tay |
|  | void MatchFinger(void) | Chương trình con thưc hiện việc kiểm tra vân tay |
|  | int sendcmd2fp(int pack) | Tham số truyền vào :  - pack : 0,1,2,3…8 gồm 8 command để finger thực thi  #define PASSPACK 1  #define DETECT 2  #define IMG2CHAR 3  #define IMG2CHAR2 4  #define CREATE\_MODEL 5  #define STORE\_MODEL 6  #define SEACRCH 7  #define DELETE 8  Hàm gửi command tới module finger |
|  | Static void MX\_USART1\_UART\_Init (void) | Khởi tao bộ USART  - Baudrate 57600  - Wordlength 8bit |
|  | void SystemClock\_Config(void) | Thiết lập clock hệ thống : 72MHz |
|  | lcd\_init (); | Khởi động lcd16x02 |
|  | FP\_InitLCD | Kiểm tra module finger kết nối thành công chưa và thông báo trên lcd |
|  | HAL\_Init() | Khởi tạo thư viện HAL |
|  | HAL\_UART\_Receive\_IT(&huart1,(uint8\_t\*)&Recived\_Byte,1); | Khởi tạo ngắt của bộ USART khi có dữ liệu nhận về chân RX của kit PA10, và truyền vào biến Recived\_Byte |
| Stm32f1xx\_it.c | Void EXTI1\_IRQHandler(void) | Chương trình ngắt của nút nhấn PA1:  - Khi nhấn nút 1 biến Inst\_Code= 1 và thực hiện chương trình matchFinger(), sau khi thực hiện xong biến Inst\_Code sẽ trả về giá trị 0 và đợi lệnh tiếp theo. |
|  | Void EXTI2\_IRQHandler(void) | Chương trình ngắt của nút nhấn PA2:  - Khi nhấn nút 1 biến Inst\_Code= 2 và thực hiện chương trình enrollFinger(), sau khi thực hiện xong biến Inst\_Code sẽ trả về giá trị 0 và đợi lệnh tiếp theo. |