

**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, tháng 12 năm 2019**

**MỤC LỤC**

[**PHẦN MỞ ĐẦU** 4](#_Toc26462648)

[**1.** **Tóm tắt ý tưởng nội dung báo cáo:** 4](#_Toc26462649)

[**2.** **Đặt vấn đề:** 4](#_Toc26462650)

[***2.1 Những nghiên cứu trong và ngoài nước liên quan đến đề tài.*** 4](#_Toc26462651)

[***2.2 Một số tài liệu liên quan đến đề tài*** 5](#_Toc26462652)

[***2.3 Tính cấp thiết cần nghiên cứu của đề tài*** 5](#_Toc26462653)

[***2.4 Mục tiêu đề tài*** 5](#_Toc26462654)

[***2.5 Đối tượng phạm vi nghiên cứu*** 5](#_Toc26462655)

[***2.6 Phương pháp nghiên cứu*** 6](#_Toc26462656)

[***2.7 Nội dung đề tài*** 6](#_Toc26462657)

[**PHẦN NỘI DUNG** 7](#_Toc26462658)

[**Chương 1: GIỚI THIỆU VI XỬ LÍ** 7](#_Toc26462659)

[***1.1. Giới thiệu dòng Vi xử lí Atmega AVR*** 7](#_Toc26462660)

[**1.1.1 Dòng vi xử lý AVR:** 7](#_Toc26462661)

[**1.1.2 Giới thiệu Atmega 8:** 10](#_Toc26462662)

[**1.1.3 Giới thiệu Atmega 328:** 16](#_Toc26462663)

[**Chương 2: CẤU TRÚC CHUNG CỦA KIT** 19](#_Toc26462664)

[***2.1*** ***Giới thiệu Kit Arduiono R3:*** 19](#_Toc26462665)

[***Một vài thông số của Arduino UNO R3:*** 19](#_Toc26462666)

[**Chương 3: NHỮNG KIẾN THỨC LIÊN QUAN** 23](#_Toc26462667)

[***3.1. Giới thiêu sơ lược module Fingerprint, LCD16x02, Keypad, Động cơ Servo*** 23](#_Toc26462668)

[**3.1.1** **Sơ lược về module Fingerprint R305** 23](#_Toc26462669)

[**3.1.2** **Giao tiếp với lcd16x02** 29](#_Toc26462670)

[**3.1.3** **Giao tiếp với keypad 4x4** 30](#_Toc26462671)

[**3.1.4** **Giao tiếp với động cơ Servo** 31](#_Toc26462672)

[**Chương 4: ỨNG DỤNG** 33](#_Toc26462673)

[**4.1 Ứng dụng trong đời sống** 33](#_Toc26462674)

[**4.2 Mô tả ứng dụng** 34](#_Toc26462675)

[**4.3 Cách nạp chương trình vào KIT** 36](#_Toc26462676)

[**4.4. Chương trình** 39](#_Toc26462677)

[**4.4.1** **Khai báo, Settup, Loop** 39](#_Toc26462678)

[**4.4.2 Chương trình con** 40](#_Toc26462679)

[**4.4.2** **Thư viện mẫu** 58](#_Toc26462680)

[**KẾT LUẬN** 72](#_Toc26462681)

[**I.** **Kết quả đạt được** 72](#_Toc26462682)

[**II.** **Ưu, nhược điểm** 72](#_Toc26462683)

[**1.** **Ưu điểm** 72](#_Toc26462684)

[**2.** **Nhược điểm** 72](#_Toc26462685)

[**III.** **Hướng phát triển** 72](#_Toc26462686)

[**PHẦN BỔ SUNG** 73](#_Toc26462687)

[**1.** **Sơ đồ nối của các module với mạch** 73](#_Toc26462688)

[**2.** **Mô hình thực tế** 76](#_Toc26462689)

[**3.** **Giải thích thư viện và hàm con** 77](#_Toc26462690)

# **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://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 Arduino.

- Hiểu được phương thức giao tiếp truyền thông UART 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 Arduino Uno

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 Atmega328.

## ***2.6 Phương pháp nghiên cứu***

* Thông qua tài liệu tìm kiếm trên Internet.
* Dựa trên tài liệu kèm theo dòng Arduino Uno.

## ***2.7 Nội dung đề tài***

* Giới thiệu về Atmega328
* Trình bày cấu trúc chung của KIT Arduino Uno 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 Atmega AVR***

### **1.1.1 Dòng vi xử lý AVR:**

Tên viết tắt của Vi điều khiển AVR là “Advanced Virtual RISC” và MCU là thuật ngữ ngắn gọn của Vi điều khiển. Vi điều khiển là một máy tính nhỏ trên một con chip và nó cũng được gọi là thiết bị điều khiển. Tương tự như máy tính, Vi điều khiển được chế tạo với nhiều thiết bị ngoại vi như các đơn vị đầu vào và đầu ra, bộ nhớ, Bộ đếm xung, truyền dữ liệu nối tiếp, lập trình được. Các ứng dụng của Vi điều khiển liên quan đến các ứng dụng nhúng & thiết bị điều khiển tự động như thiết bị y tế, thiết bị điều khiển từ xa, hệ thống điều khiển, văn phòng, công cụ điện, thiết bị điện tử, v.v. Có nhiều loại Vi điều khiển có sẵn trên thị trường như 8051, PIC và AVR vi điều khiển.

Họ vi điều khiển AVR là dòng sản phẩm được phát triển bởi hảng Atmel (1996), nó được chế tạo dựa trên cấu trúc AVR RISC (Reduced Instruction Set Computer) đồng thời AVR là một trong những họ vi điều khiển đầu tiên sử dụng bộ nhớ Flash để lưu trữ chương trình. Có thể thấy rằng trong những năm gần đây Atmel đã trở thành nhà tiên phong trên thế giới về phát triển kỹ thuật bộ nhớ Flash (không biến đổi, có thể xóa bằng điện và lập trình lại bộ nhớ, Họ AVR thường được sử dụng trong các sản phẩm như Camera số, board chủ PC…)  
  
 Vi điều khiển Atmega AVR có công suất cao, tiêu thụ năng lượng thấp, cấu trúc RISC tiến với 130 lệnh với chu kỳ thực hiện đơn xung lớn nhất, 32 thanh ghi đa mục đích 8 bít, 16 MIPS tại tần số đặt 16 MHz, bộ nhân 2 chu kỳ On-chip, Power-on Reset và Brown-out Detection có thể lập trình, bộ dao động RC bên trong có thể lập trình các mức, 5 Mode ngủ (Idle, ADC Noise Reduction, Power-save, Power-down và Standby), có khả năng Reset khi bật nguồn, khả năng dò lỗi Brown out lập trình được, có nguồn ngắt trong và ngắt ngoài.  
  
 Cốt lõi của AVR là sự kết hợp các câu lệnh phong phú với 32 thanh ghi đa mục đích. Tất cả 32 thanh ghi đều trực tiếp kết nối tới bộ xử lý logíc số học - Arithmetic Logic Unit (ALU), cho phép truy nhập 2 thanh ghi độc lập trong một câu lệnh đơn được thực hiện trong một chu kỳ xung. Kết quả của cấu trúc trở nên gọn nhẹ, hiệu quả hơn, trong khi vẫn đạt được thời gian xử lý nhanh hơn gấp 10 lần các vi điều khiển CISC thông thường khác.  
  
 8K byte Flash trên chíp có thể lập trình với các khả năng đọc trong khi ghi (Read-While-Write), 512 byte EEPROM, 1K byte SRAM, 23 đường vào ra đa mục đích, 32 thanh ghi đa mục đích, 3 Timer/Counter rất linh hoạt với các compare mode, các ngắt trong và ngắt ngoài, một bộ USART nối tiếp có thể lập trình được, ghép nối nối tiếp 2 dây định hướng byte, 6 kênh ADC (8 kênh với loại TQFP và MLF packages) trong đó 4 (hoặc 6) kênh có độ chính xác 10-bit và 2 kênh có độ chính xác 8-bit, Watchdog Timer có thể lập trình được với bộ dao động bên trong, một cổng nối tiếp SPI và 5 mode tiết kiệm năng lượng có thể lựa chọn mềm.  
  
 Idle mode dừng CPU trong khi vẫn cho phép SRAM, Timer/Counters, cổng SPI, và hệ thống ngắt tiếp tục chức năng của chúng.  
  
 Power-down mode tiết kiệm nội dung thanh ghi, nhưng hạn định bộ dao động, không cho phép tất cả các chức năng khác của chíp được hoạt động cho đến khi ngắt tiếp theo hoặc Reset phần cứng xuất hiện.  
  
 Trong Power-save mode, timer không đồng bộ tiếp tục chạy, cho phép sử dụng để duy trì thời gian nền, trong khi các phần còn lại của thiết bị được ngủ.  
  
 ADC Noise Reduction mode dừng CPU và tất các module I/O ngoại trừ timer không đồng bộ và ADC để tối thiểu hóa nhiễu mạch trong suốt quá trình ADC trong chuyển đổi.  
  
 Trong Standby mode, bộ dao động thạch anh/ resonator được phép chạy trong khi các phần còn lại của thiết bị được ngủ. Điều này cho phép start-up rất nhanh cùng với hiệu quả tiêu thụ ít năng lượng.  
  
 Thiết bị được sản suất áp dụng công nghệ tích hợp bộ nhớ non-volatile cao của Atmel. Bộ nhớ chương trình Flash này có thể lập trình thông qua ghép nối tiếp SPI bằng chương trình lập trình bộ nhớ non-volatile riêng, hoặc bằng một chương trình boot on – chip, chạy trong AVR core. Chương trình boot có thể sử dụng bất kỳ một ghép nối nào để download chương trình ứng dụng trong bộ nhớ Flash. Phần mềm trong Boot Flash sẽ tiếp tục chạy trong khi các phần sử dụng Flash vẫn được update, hỗ trợ cho hoạt động đọc trong khi ghi (Read-While-Write).  
  
 Bằng việc kết hợp với một CPU 8-bit RISC với bộ nhớ Flash tự lập trình trong hệ thống trên một chíp, Atmel ATmega8 là một vi điều khiển cực mạnh, thỏa mãn yêu cầu về một bộ vi điều khiển với độ linh hoạt cao và đem lại lợi nhuận lớn với rất nhiều các ứng dụng điều khiển tác động nhanh.  
  
 ATmega8 AVR cũng hỗ trợ đầy đủ về lập trình và phát triển các tool hệ thống, bao gồm bộ dịch C, macro assemblers, bộ mô phỏng/gỡ rối chương trình, In-Circuit Emulators, và evaluation kits.

Cơ bản họ AVR có thể chia làm 4 nhóm sau:   
• **tinyAVR** — the ATtiny series   
  
o 1–8 kB program memory  
o 6–32-pin package  
o Limited peripheral set  
  
• **megaAVR** — the ATmega series  
  
o 4–256 kB program memory  
o 28–100-pin package  
o Extended instruction set (Multiply instructions and instructions for handling larger program memories)  
o Extensive peripheral   
  
• **XMEGA** — the ATxmega series   
  
o 16–384 kB program memory  
o 44–64–100-pin package (A4, A3, A1)  
o Extended performance features, such as DMA, "Event System", and cryptography support.  
o Extensive peripheral set with DACs  
  
• **Atmel At94k FPSLIC** (Field Programmable System Level Integrated Circuit), an AVR core on-die with an FPGA. The FPSLIC uses SRAM for the AVR program code, unlike all other AVRs. Partly due to the relative speed difference between SRAM and flash, the AVR core in the FPSLIC can run at up to 50MHz.  
AVR32 AP7000 Sram 32KB 1.8 - 3.3V  
  
ATUC3A0xxxx (giá trị xxxx thể hiện dung lượng flash của VĐK)  
ATUC3A0512 Flash 512kB sram64B 3.0-3.6V.

### **1.1.2 Giới thiệu Atmega 8:**

Vào năm 1996, Vi điều khiển AVR được sản xuất bởi Tập đoàn “Atmel Corporation”. Vi điều khiển bao gồm kiến ​​trúc Harvard hoạt động nhanh chóng với RISC. Các tính năng của Vi điều khiển này bao gồm các tính năng khác nhau so với các chế độ nghỉ khác – 6, ADC sẵn có (bộ chuyển đổi tương tự sang số) , bộ tạo dao động nội bộ và giao tiếp dữ liệu nối tiếp, thực hiện các hướng dẫn trong một chu kỳ thực hiện. Các vi điều khiển này rất nhanh và chúng sử dụng năng lượng thấp để hoạt động ở các chế độ tiết kiệm năng lượng khác nhau. Có các cấu hình khác nhau của bộ vi điều khiển AVR có sẵn để thực hiện các hoạt động khác nhau như 8 bit, 16 bit và 32 bit. Vui lòng tham khảo các liên kết dưới đây cho;

Các loại vi điều khiển AVR

[](https://advancecad.edu.vn/wp-content/uploads/2018/12/Atmega8-Microcontroller.jpg)

*Vi điều khiển Atmega8*

Bộ vi điều khiển AVR có ba phiên bản khác nhau như TinyAVR, MegaAVR và XmegaAVR

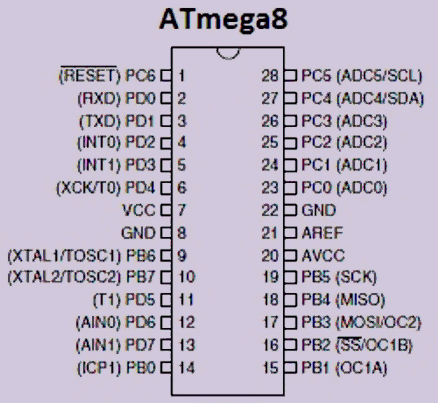
Bộ vi điều khiển Tiny AVR có kích thước rất nhỏ và được sử dụng trong nhiều ứng dụng đơn giản

Bộ vi điều khiển Mega AVR rất nổi tiếng nhờ có số lượng lớn các thành phần tích hợp, bộ nhớ tốt và được sử dụng cho nhiều ứng dụng

Bộ vi điều khiển Xmega AVR được áp dụng trong các ứng dụng khó, đòi hỏi tốc độ cao và bộ nhớ chương trình khổng lồ.

**Mô tả chân vi điều khiển Atmega8**

Các tính năng chính của Vi điều khiển là Atmega8, tất cả các chân của Vi điều khiển có thể truyền hai tín hiệu trừ 5 chân. Bộ vi điều khiển Atmega8 bao gồm 28 chân trong đó các chân 9,10,14,15,16,17,18,19 được sử dụng cho chân B, Chân 23,24,25,26,27,28 và 1 được sử dụng cho chân C và chân 2,3,4,5,6,11,12 được sử dụng cho chân D.

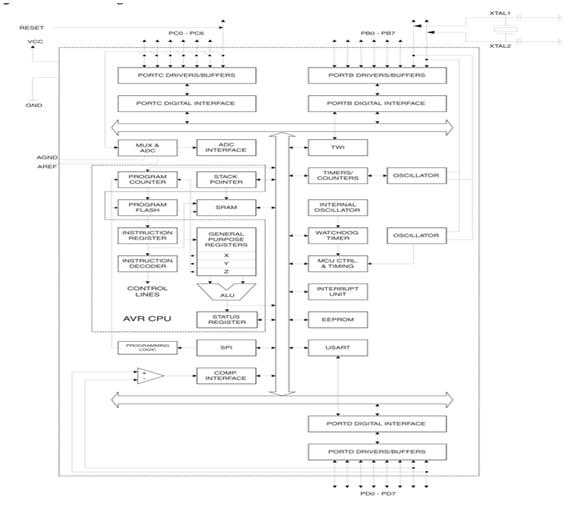
*[](https://advancecad.edu.vn/wp-content/uploads/2018/12/Atmega8-Microcontroller-Pin-Configuration.png)*

*Cấu hình chân vi điều khiển Atmega8*

* Chân -1 là chân RST (Reset) và áp dụng tín hiệu mức thấp trong thời gian dài hơn độ dài xung tối thiểu sẽ tạo ra RESET.
* Chân-2 và chân-3 được sử dụng trong USART để nối nối tiếp
* Chân-4 và chân-5 được sử dụng làm ngắt ngoài. Một trong số chúng sẽ kích hoạt khi một bit cờ ngắt của thanh ghi trạng thái được đặt và cái còn lại sẽ kích hoạt miễn là điều kiện kích hoạt thành công.
* Chân-9 & chân-10 được sử dụng làm bộ dao động xung cũng như bộ tạo dao động ngoài trong đó tinh thể được liên kết trực tiếp với hai chân. Chân-10 được sử dụng cho bộ tạo dao động tần số thấp hoặc bộ tạo dao động tinh thể. Nếu bộ tạo dao động RC được điều chỉnh bên trong được sử dụng làm nguồn CLK và bộ định thời không đồng bộ được cho phép, các chân này có thể được sử dụng làm chân dao động xung.
* Chân-19 được sử dụng làm Master CLK o / p, CLK i / p phụ cho kênh SPI.
* Chân-18 được sử dụng làm Master CLK i / p, phụ cho CLK o / p.
* Chân-17 được sử dụng làm dữ liệu chủ o / p, dữ liệu nô lệ i / p cho kênh SPI. Nó được sử dụng như một i / p khi được trao quyền bởi một nô lệ & là hai chiều khi được chủ nhân cho phép. Ghim này cũng có thể được sử dụng làm o / p so với o / p khớp, giúp làm o / p bên ngoài cho bộ đếm thời gian / bộ đếm.
* Chân-16 được sử dụng làm lựa chọn phụ cho i / p. Nó cũng có thể được sử dụng như một bộ đếm xung hoặc bộ đếm tương đối bằng cách sắp xếp chân PB2 làm o / p.
* Chân-15 có thể được sử dụng làm o / p bên ngoài của bộ đếm xung hoặc bộ đếm so sánh khớp A.
* Chân-23 đến Chân 28 đã được sử dụng cho các kênh ADC (giá trị số của đầu vào tương tự). Chân-27 cũng có thể được sử dụng làm giao diện nối tiếp CLK & chân-28 có thể được sử dụng làm dữ liệu giao diện nối tiếp
* Chân-12 và chân-13 được sử dụng làm i / ps so sánh tương tự.
* Chân-6 và chân-11 được sử dụng làm nguồn xung / bộ đếm.

**Kiến trúc vi điều khiển Atmega8 AVR**

Kiến trúc Vi điều khiển Atmega AVR bao gồm các khối sau.

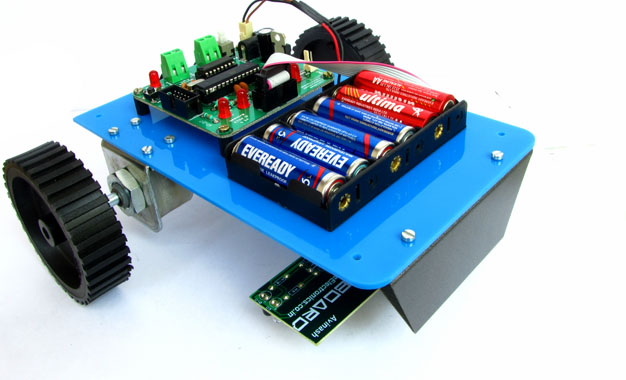
[](https://advancecad.edu.vn/wp-content/uploads/2018/12/Architecture-of-Atmega8-Microcontroller.jpg)

*Kiến trúc của vi điều khiển Atmega8*

* Bộ nhớ: Nó có 1Kbyte Internal SRAM, 8 Kb bộ nhớ chương trình Flash và 512 Byte EEPROM.
* Chân I / O: Nó có ba chân, đó là chân B, chân-C và chân-D và 23 I / O có thể đạt được từ các chân này.
* Ngắt: Hai nguồn Ngắt bên ngoài được đặt tại chân D. Mười chín vectơ ngắt khác nhau hỗ trợ mười chín sự kiện được tạo ra bởi các thiết bị ngoại vi bên trong.
* Xung / Bộ đếm: Có 3 Bộ đếm xung nội bộ có thể truy cập, 8 bit-2, 16 bit-1, trình bày nhiều chế độ hoạt động và hỗ trợ xung bên trong / bên ngoài.
* Giao diện ngoại vi nối tiếp (SPI): Vi điều khiển ATmega8 chứa ba thiết bị liên lạc tích hợp. Một trong số đó là SPI, 4 chân được phân bổ cho Vi điều khiển để thực hiện hệ thống truyền thông này.
* USART: USART là một trong những giải pháp truyền thông mạnh mẽ nhất. Vi điều khiển ATmega8 hỗ trợ cả hai sơ đồ truyền dữ liệu đồng bộ & không đồng bộ. Nó có ba chân được phân bổ cho điều đó. Trong nhiều dự án truyền thông, mô-đun USART được sử dụng rộng rãi để liên lạc với PC-Vi điều khiển.
* Giao diện hai dây (TWI): TWI là một thiết bị giao tiếp khác có trong vi điều khiển ATmega8. Nó cho phép các nhà thiết kế thiết lập một giao tiếp b / n hai thiết bị sử dụng hai dây cùng với kết nối GND, vì o / p của TWI được tạo bằng phương tiện của bộ thu mở o / ps, do đó, điện trở bên ngoài là bắt buộc để làm mạch.
* Bộ so sánh tương tự: Mô-đun này được kết hợp trong mạch tích hợp cung cấp cơ sở tương phản giữa hai điện áp được liên kết với hai đầu vào của bộ so sánh thông qua các chân ngoài được liên kết với Vi điều khiển.
* ADC: ADC sẵn có (bộ chuyển đổi tương tự sang số) có thể thay đổi tín hiệu i / p tương tự thành dữ liệu số có độ phân giải 10 bit. Đối với ứng dụng cấp thấp, độ phân giải này là đủ.

**Ứng dụng vi điều khiển Atmega8**

Bộ vi điều khiển Atmega8 được sử dụng để xây dựng các dự án điện và điện tử khác nhau. Một số dự án Vi điều khiển AVR atmega8 được liệt kê dưới đây.

[](https://advancecad.edu.vn/wp-content/uploads/2018/12/Atmega8-Project.jpg)

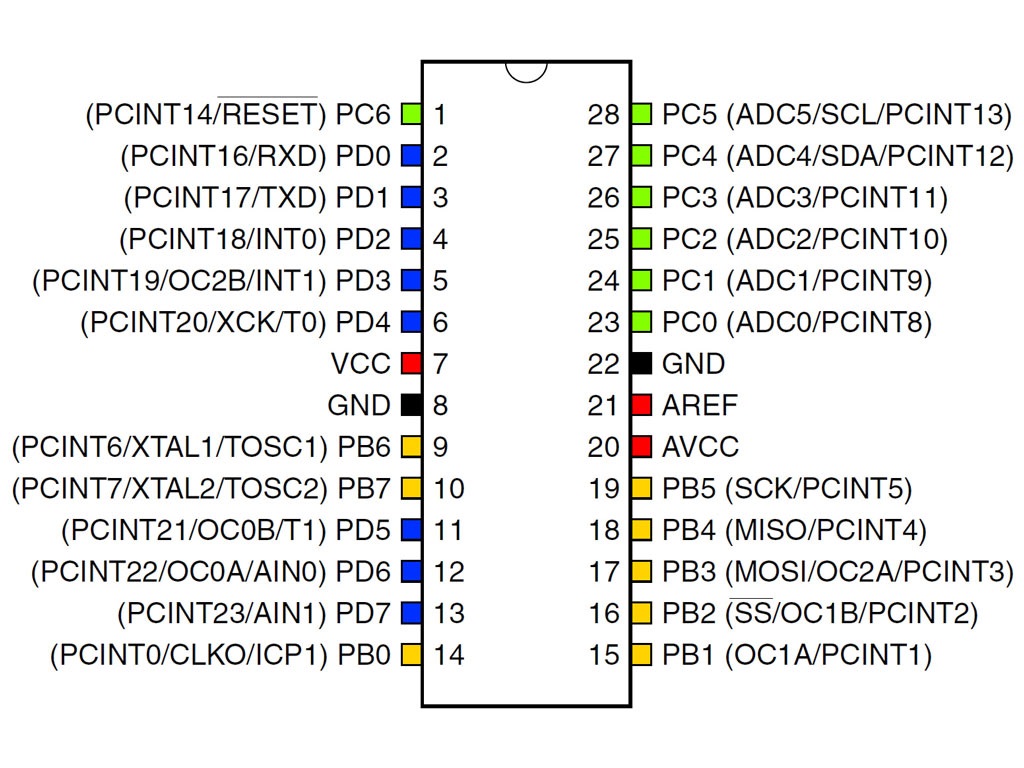
*Dự án dựa trên Atmega8*

* Giao diện ma trận LED dựa trên vi điều khiển AVR
* Giao tiếp UART giữa ArduinoUno và ATmega8
* Giao diện của bộ ghép nối với vi điều khiển ATmega8
* Hệ thống báo cháy dựa trên vi điều khiển AVR
* Đo cường độ ánh sáng bằng vi điều khiển AVR và LDR
* Vi điều khiển AVR với Ampe kế 100mA
* Hệ thống báo động chống trộm dựa trên vi điều khiển ATmega8
* Bộ điều khiển dựa trên vi điều khiển AVR
* Giao diện điều khiển dựa trên vi điều khiển AVR
* Điều khiển động cơ bước bằng vi điều khiển AVR

### **1.1.3 Giới thiệu Atmega 328:**

Atmega328 là một chíp vi điều khiển được sản xuất bời hãng [Atmel](http://www.atmel.com/) thuộc họ MegaAVR có sức mạnh hơn hẳn [Atmega8](http://thegioichip.com.vn/san-pham/MEGA8A-PU-80.html). Atmega 328 là một bộ vi điều khiển 8 bít dựa trên kiến trúc RISC bộ nhớ chương trình 32KB ISP flash có thể ghi xóa hàng nghìn lần, 1KB EEPROM, một bộ nhớ RAM vô cùng lớn trong thế giới vi xử lý 8 bít (2KB SRAM)

Với 23 chân có thể sử dụng cho các kết nối vào hoặc ra i/O, 32 thanh ghi, 3 bộ timer/counter có thể lập trình, có các gắt nội và ngoại (2 lệnh trên một vector ngắt), giao thức truyền thông nối tiếp USART, SPI, I2C. Ngoài ra có thể sử dụng bộ biến đổi số tương tự 10 bít (ADC/DAC) mở rộng tới 8 kênh, khả năng lập trình được watchdog timer, hoạt động với 5 chế độ nguồn, có thể sử dụng tới 6 kênh điều chế độ rộng xung (PWM), hỗ trợ bootloader.

****

Atemega328 có khả năng hoạt động trong một dải điện áp rộng (1.8V – 5.5V), tốc độ thực thi (thông lượng) 1MIPS trên 1MHz

Ngày nay vi điều khiển Atmega328 thực sử được sử dụng phổ biến từ các dự án nhỏ của sinh viên, học sinh với giá thành rẻ, xử lý mạnh mẽ, tiêu tốn ít năng lượng (chế độ hoạt động : 0.2 mA, chế độ ngủ: 0.1 μA, chế độ tích kiệm: 0.75 μA) và sự hỗ trợ nhiệt tình của cộng đồng người dùng AVR. Và không thể không nhắc tới sự thành công của Vi điều khiển Atmega328 trong dự án mã nguồn mở [Arduino](https://www.arduino.cc/) với các modul [Adruino Uno (R3)](http://thegioichip.com.vn/san-pham/ARDUINO-UNO-R3-DIP-18.html) những sản phẩm dẫn dắt chúng ta vào thế giới mã nguồn mở để hoàn thành một chương trình trong “nháy mắt”.

**Thông số chính Atmega328P-PU:**

+ Kiến trúc: AVR 8bit  
+ Xung nhịp lớn nhất: 20Mhz  
+ Bộ nhớ chương trình (FLASH): 32KB  
+ Bộ nhớ EEPROM: 1KB  
+ Bộ nhớ RAM: 2KB  
+ Điện áp hoạt động rộng: 1.8V – 5.5V  
+ Số timer: 3 timer gồm 2 timer 8-bit và 1 timer 16-bit  
+ Số kênh xung PWM: 6 kênh (1timer 2 kênh)

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

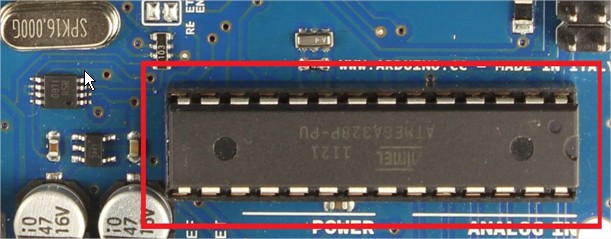
## ***Giới thiệu Kit Arduiono R3:***

Arduino UNO R3 là kit Arduino UNO thế hệ thứ 3, với khả năng lập trình cho các ứng dụng điều khiển phức tạp do được trang bị cấu hình mạnh cho các loại bộ nhớ ROM, RAM và Flash, các ngõ vào ra digital I/O trong đó có nhiều ngõ có khả năng xuất tín hiệu PWM, các ngõ đọc tín hiệu analog và các chuẩn giao tiếp đa dạng như UART, SPI, TWI (I2C).

## ***Một vài thông số của Arduino UNO R3:***

|  |  |
| --- | --- |
| Vi điều khiển | ATmega328 họ 8bit |
| Điện áp hoạt động | 5V DC (chỉ được cấp qua cổng USB) |
| Tần số hoạt động | 16 MHz |
| Dòng tiêu thụ | khoảng 30mA |
| Điện áp vào khuyên dùng | 7-12V DC |
| Điện áp vào giới hạn | 6-20V DC |
| Số chân Digital I/O | 14 (6 chân hardware PWM) |
| Số chân Analog | 6 (độ phân giải 10bit) |
| Dòng tối đa trên mỗi chân I/O | 30 mA |
| Dòng ra tối đa (5V) | 500 mA |
| Dòng ra tối đa (3.3V) | 50 mA |
| Bộ nhớ flash | 32 KB (ATmega328) với 0.5KB dùng bởi bootloader |
| SRAM | 2 KB (ATmega328) |
| EEPROM | 1 KB (ATmega328) |

**Vi điều khiển:**



Arduino UNO có thể sử dụng 3 vi điều khiển họ 8bit AVR là ATmega8, ATmega168, ATmega328. Bộ não này có thể xử lí những tác vụ đơn giản như điều khiển đèn LED nhấp nháy, xử lí tín hiệu cho xe điều khiển từ xa, làm một trạm đo nhiệt độ - độ ẩm và hiển thị lên màn hình LCD,…

**Năng lượng:**

Arduino UNO có thể được cấp nguồn 5V thông qua cổng USB hoặc cấp nguồn ngoài với điện áp khuyên dùng là 7-12V DC và giới hạn là 6-20V. Thường thì cấp nguồn bằng pin vuông 9V là hợp lí nhất nếu bạn không có sẵn nguồn từ cổng USB. Nếu cấp nguồn vượt quá ngưỡng giới hạn trên, bạn sẽ làm hỏng Arduino UNO.

**Các chân năng lượng:**

* **GND (Ground)**: cực âm của nguồn điện cấp cho Arduino UNO. Khi bạn dùng các thiết bị sử dụng những nguồn điện riêng biệt thì những chân này phải được nối với nhau.
* **5V**: cấp điện áp 5V đầu ra. Dòng tối đa cho phép ở chân này là 500mA.
* **3.3V**: cấp điện áp 3.3V đầu ra. Dòng tối đa cho phép ở chân này là 50mA.
* **Vin (Voltage Input)**: để cấp nguồn ngoài cho Arduino UNO, bạn nối cực dương của nguồn với chân này và cực âm của nguồn với chân GND.
* **IOREF**: điện áp hoạt động của vi điều khiển trên Arduino UNO có thể được đo ở chân này. Và dĩ nhiên nó luôn là 5V. Mặc dù vậy bạn không được lấy nguồn 5V từ chân này để sử dụng bởi chức năng của nó không phải là cấp nguồn.
* **RESET**: việc nhấn nút Reset trên board để reset vi điều khiển tương đương với việc chân RESET được nối với GND qua 1 điện trở 10KΩ.

**Bộ nhớ:**

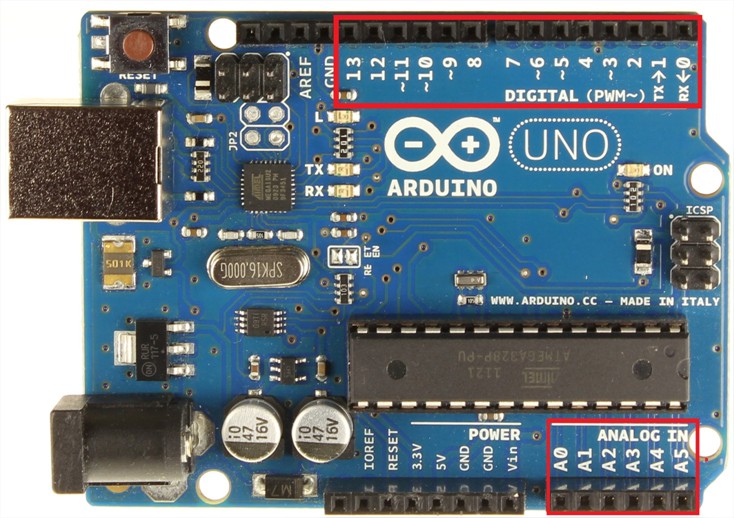
Vi điều khiển Atmega328 tiêu chuẩn cung cấp cho người dùng:

**32KB bộ nhớ Flash**: những đoạn lệnh bạn lập trình sẽ được lưu trữ trong bộ nhớ Flash của vi điều khiển. Thường thì sẽ có khoảng vài KB trong số này sẽ được dùng cho bootloader nhưng đừng lo, bạn hiếm khi nào cần quá 20KB bộ nhớ này đâu.

**2KB cho SRAM** (**S**tatic **R**andom **A**ccess **M**emory): giá trị các biến bạn khai báo khi lập trình sẽ lưu ở đây. Bạn khai báo càng nhiều biến thì càng cần nhiều bộ nhớ RAM. Tuy vậy, thực sự thì cũng hiếm khi nào bộ nhớ RAM lại trở thành thứ mà bạn phải bận tâm. Khi mất điện, dữ liệu trên SRAM sẽ bị mất.

**1KB cho EEPROM** (**E**lectrically **E**raseble **P**rogrammable **R**ead **O**nly **M**emory): đây giống như một chiếc ổ cứng mini – nơi bạn có thể đọc và ghi dữ liệu của mình vào đây mà không phải lo bị mất khi cúp điện giống như dữ liệu trên SRAM.

**Các cổng vào/ra:**

[](http://k3.arduino.vn/img/2014/05/25/0/467_8121-1401018414-0--input.jpg)

Arduino UNO có 14 chân digital dùng để đọc hoặc xuất tín hiệu. Chúng chỉ có 2 mức điện áp là 0V và 5V với dòng vào/ra tối đa trên mỗi chân là 40mA. Ở mỗi chân đều có các điện trở pull-up từ được cài đặt ngay trong vi điều khiển ATmega328 (mặc định thì các điện trở này không được kết nối).

***Một số chân digital có các chức năng đặc biệt như sau:***

* **2 chân Serial**: 0 (RX) và 1 (TX): dùng để gửi (transmit – TX) và nhận (receive – RX) dữ liệu TTL Serial. Arduino Uno có thể giao tiếp với thiết bị khác thông qua 2 chân này. Kết nối bluetooth thường thấy nói nôm na chính là kết nối Serial không dây. Nếu không cần giao tiếp Serial, bạn không nên sử dụng 2 chân này nếu không cần thiết
* **Chân PWM (~): 3, 5, 6, 9, 10, và 11**: cho phép bạn xuất ra xung PWM với độ phân giải 8bit (giá trị từ 0 → 28-1 tương ứng với 0V → 5V) bằng hàm analogWrite(). Nói một cách đơn giản, bạn có thể điều chỉnh được điện áp ra ở chân này từ mức 0V đến 5V thay vì chỉ cố định ở mức 0V và 5V như những chân khác.
* **Chân giao tiếp SPI:** 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).  Ngoài các chức năng thông thường, 4 chân này còn dùng để truyền phát dữ liệu bằng giao thức SPI với các thiết bị khác.
* **LED 13**: trên Arduino UNO có 1 đèn led màu cam (kí hiệu chữ L). Khi bấm nút Reset, bạn sẽ thấy đèn này nhấp nháy để báo hiệu. Nó được nối với chân số 13. Khi chân này được người dùng sử dụng, LED sẽ sáng.

Arduino UNO có 6 chân analog (A0 → A5) cung cấp độ phân giải tín hiệu 10bit (0 → 210-1) để đọc giá trị điện áp trong khoảng 0V → 5V. Với chân **AREF** trên board, bạn có thể để đưa vào điện áp tham chiếu khi sử dụng các chân analog. Tức là nếu bạn cấp điện áp 2.5V vào chân này thì bạn có thể dùng các chân analog để đo điện áp trong khoảng từ 0V  → 2.5V với độ phân giải vẫn là 10bit.

Đặc biệt, Arduino UNO có 2 chân A4 (SDA) và A5 (SCL) hỗ trợ giao tiếp I2C/TWI với các thiết bị khác.

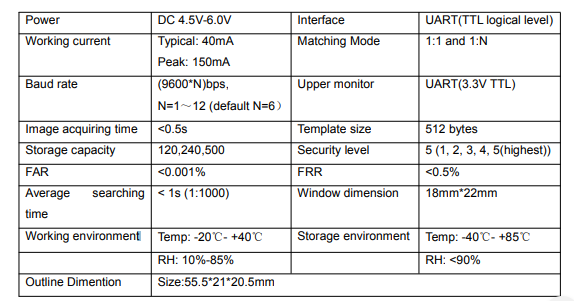
# **Chương 3: NHỮNG KIẾN THỨC LIÊN QUAN**

## ***3.1. Giới thiêu sơ lược module Fingerprint, LCD16x02, Keypad, Động cơ Servo***

* + 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.



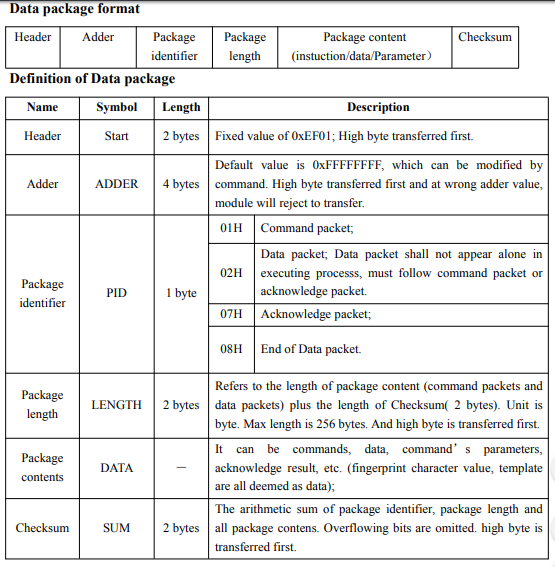
**Serial communication protocol**

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:



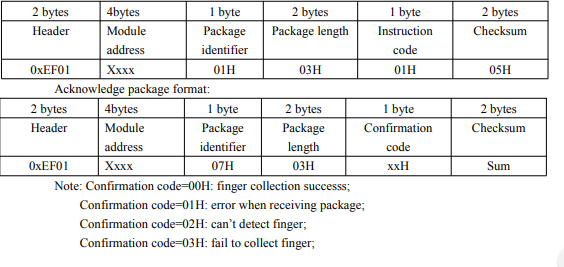
**Các tập lệnh giao tiếp**

**1. Lấy hình ảnh vân tay GetImage**

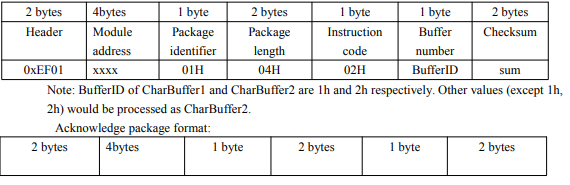
Mô tả : phát hiện vân tay và lưu trữ hình ảnh vân tay được phát hiện trong ImageBuffer khi trả lại mã xác nhận thành công; Nếu không có ngón tay, mã xác nhận được trả về sẽ có thể phát hiện ra vân tay.

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;

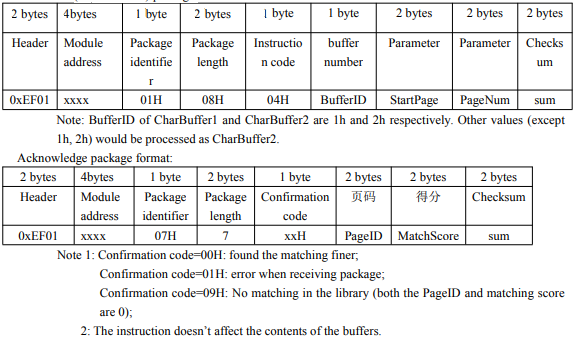
Confirmation code=06H: fail to generate character file due to the over-disorderly fingerprint image;

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

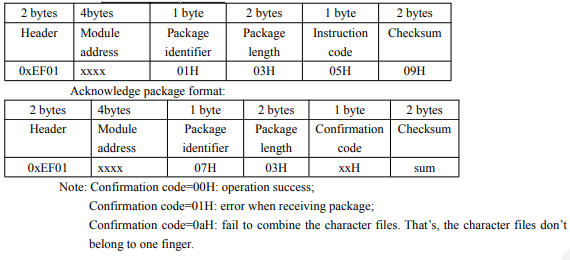
 Tham số trả về: Mã xác nhận (1 byte), PageID (vị trí mẫu phù hợp code Mã lệnh: 04H Lệnh (hoặc lệnh) định dạng gói:

**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):



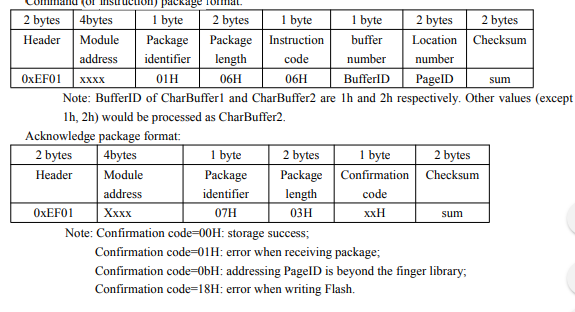
**5. Để lưu mẫu Store**

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):

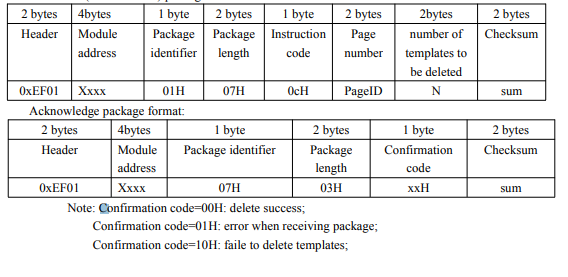


**6. Để xóa mẫu DeletChar**

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)

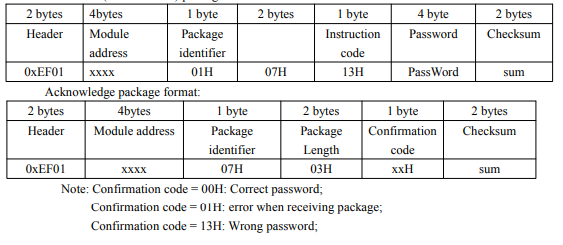
Trả về tham số: Mã xác nhận (1 byte) Mã định dạng: 0cH Định dạng gói lệnh (hoặc lệnh):



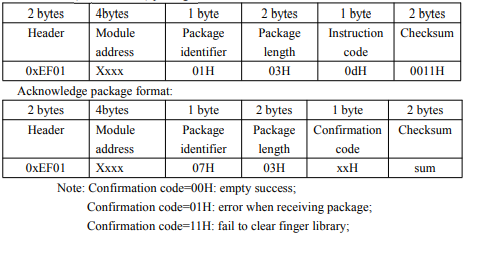
**7. Xác minh mật khẩu VfyPwd**

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)

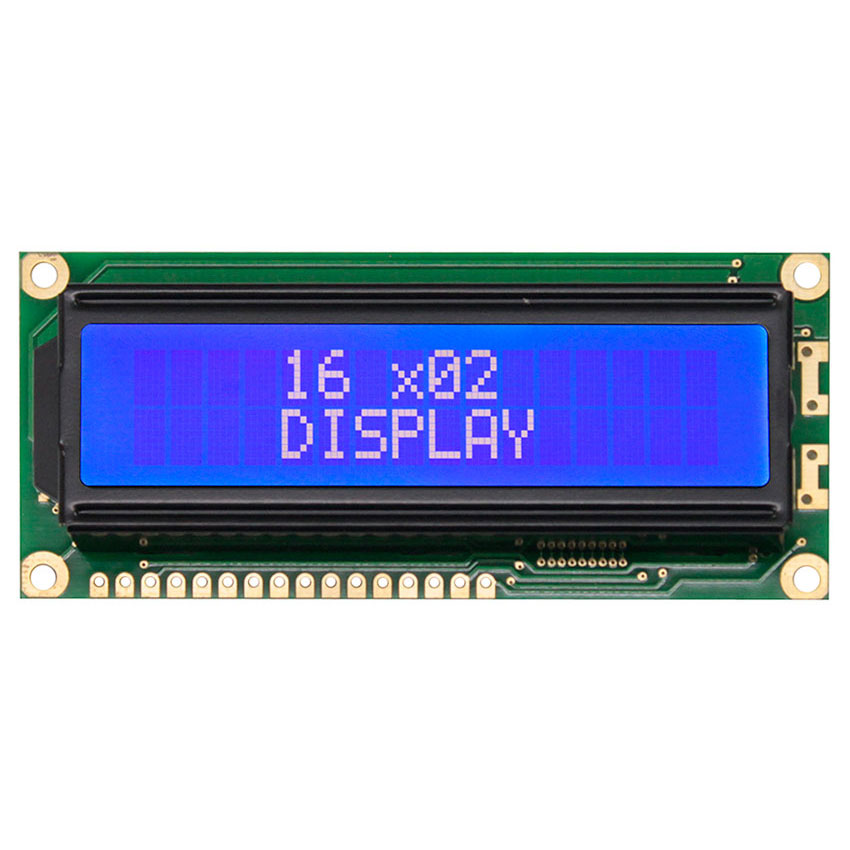
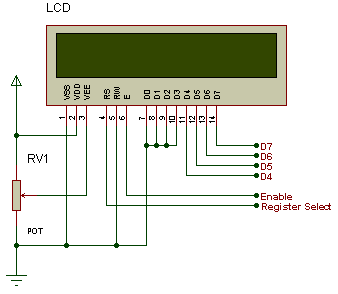
Tham số trả về: Mã xác nhận (1 byte) Mã lệnh: Định dạng gói 13H Lệnh (hoặc lệnh):



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**

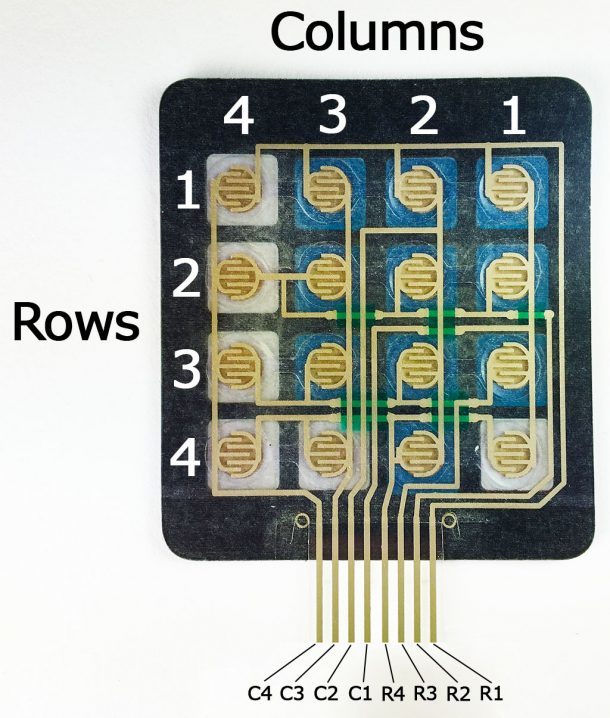
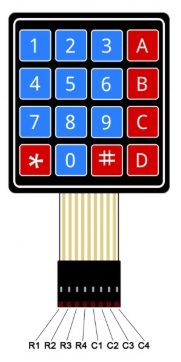


Chân kết nối LCD được khởi tạo trong hàm lcd\_txt.h ở thư viện lcd.c

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

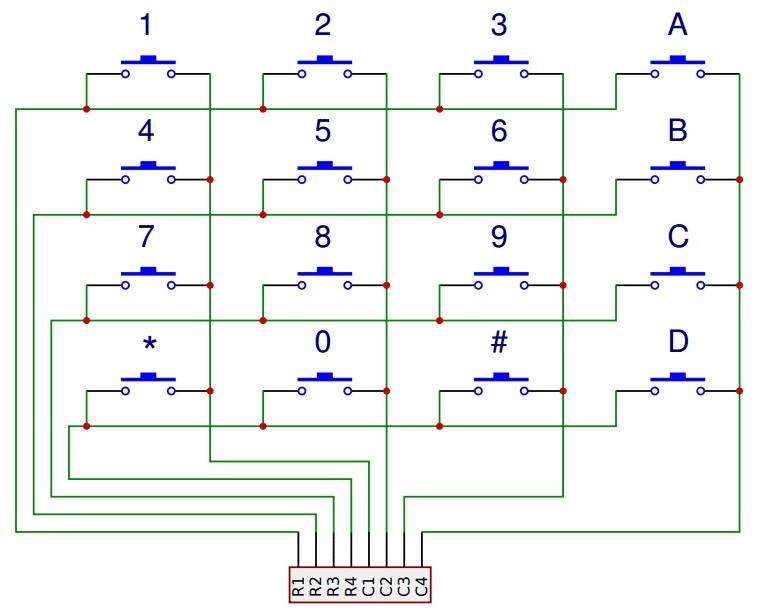
1. VSS: tương đương với GND - cực âm
2. VDD: tương đương với VCC - cực dương (5V)
3. Constrast Voltage (Vo): điều khiển độ sáng màn hình
4. Register Select (RS): điều khiển địa chỉ nào sẽ được ghi dữ liệu
5. Read/Write (RW): đọc (read mode) hay ghi (write mode) dữ liệu
6. Enable pin: Cho phép ghi vào 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.
   * 1. **Giao tiếp với keypad 4x4**

Các nút nhấn trên bàn phím được sắp xếp theo hàng và cột. Bàn phím 4X4 có 4 hàng và 4 cột:



Bên dưới mỗi phím là một màng công tắc. Mỗi công tắc trong một hàng được kết nối với các công tắc khác trong hàng bằng một đường dẫn điện bên dưới miếng đệm. Mỗi công tắc trong một cột được kết nối theo cùng một cách – một bên của công tắc được kết nối với tất cả các công tắc khác trong cột đó bằng một đường dẫn điện. Mỗi hàng và cột được đưa ra một chân duy nhất, có tất cả 8 chân đối với bàn phím 4X4.

Khi một nút nhấn được nhấn sẽ đóng công tắc giữa một cột và đường dẫn điện một hàng, cho phép dòng điện chạy giữa một chân cột và một chân hàng.

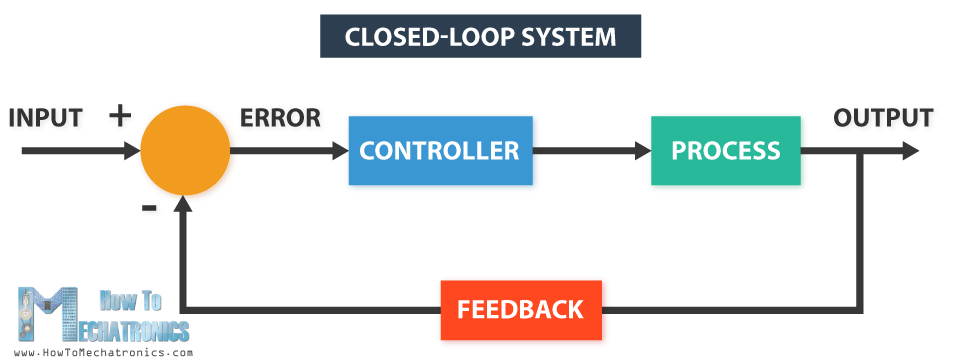
Sơ đồ bàn phím 4X4 cho thấy cách các hàng và cột được kết nối:

Arduino phát hiện nút nhấn nào được nhấn bằng cách phát hiện chân hàng và cột mà được kết nối với nút nhấn.

* + 1. **Giao tiếp với động cơ Servo**

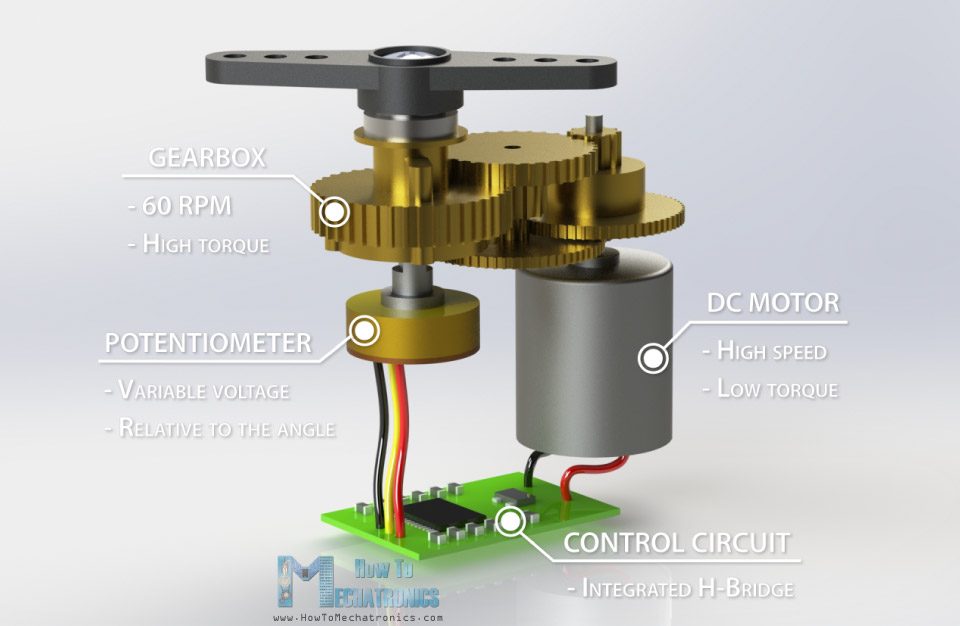
**Giới thiệu tổng quan về động cơ servo**

Có nhiều loại động cơ servo và tính năng chính của chúng là khả năng điều khiển chính xác vị trí của trục. Động cơ servo là một hệ thống vòng kín sử dụng phản hồi vị trí để điều khiển chuyển động và vị trí cuối cùng của nó.



Trong công nghiệp động cơ servo là loại cảm biến phản hồi vị trí, thường là một bộ code hóa có độ chính xác cao, trong khi trong các động cơ RC hoặc nhỏ hơn, cảm biến vị trí thường là một chiết áp đơn giản. Vị trí thực tế được tìm thấy bởi các thiết bị này được đưa trở lại bộ phát hiện lỗi nơi nó được so sánh với vị trí đích. Sau đó, theo lỗi, bộ điều khiển sửa vị trí thực tế của động cơ để khớp với vị trí đích.

**Cách thức hoạt động của Servo RC / cổ điển**

Bên trong một servo cổ diển có bốn thành phần chính, động cơ DC, hộp số, biến trở và mạch điều khiển. Động cơ DC có tốc độ cao và mô-men xoắn thấp nhưng hộp số giảm tốc độ xuống khoảng 60 vòng / phút, đồng thời tăng mô-men xoắn.

Cách thức hoạt động của Servo bên trong các thành phần Mạch điều khiển chiết áp động cơ DC

Chiết áp được gắn trên bánh răng cuối cùng hoặc trục đầu ra, do đó động cơ cũng quay chiết áp, tạo ra một điện áp liên quan đến góc tuyệt đối của trục đầu ra. Trong mạch điều khiển, điện áp chiết áp này được so sánh với điện áp đến từ đường tín hiệu. Nếu cần, bộ điều khiển kích hoạt mạch cầu H tích hợp cho phép động cơ quay theo hai hướng cho đến khi hai tín hiệu đạt mức chênh lệch bằng không.

Một động cơ servo được điều khiển bằng cách gửi một loạt các xung qua đường tín hiệu. Tần số của tín hiệu điều khiển phải là 50Hz hoặc một chu kỳ xung là 20ms. Độ rộng của xung xác định vị trí góc của servo và các loại servo này thường có thể xoay 180 độ (chúng có giới hạn vật lý khi di chuyển).

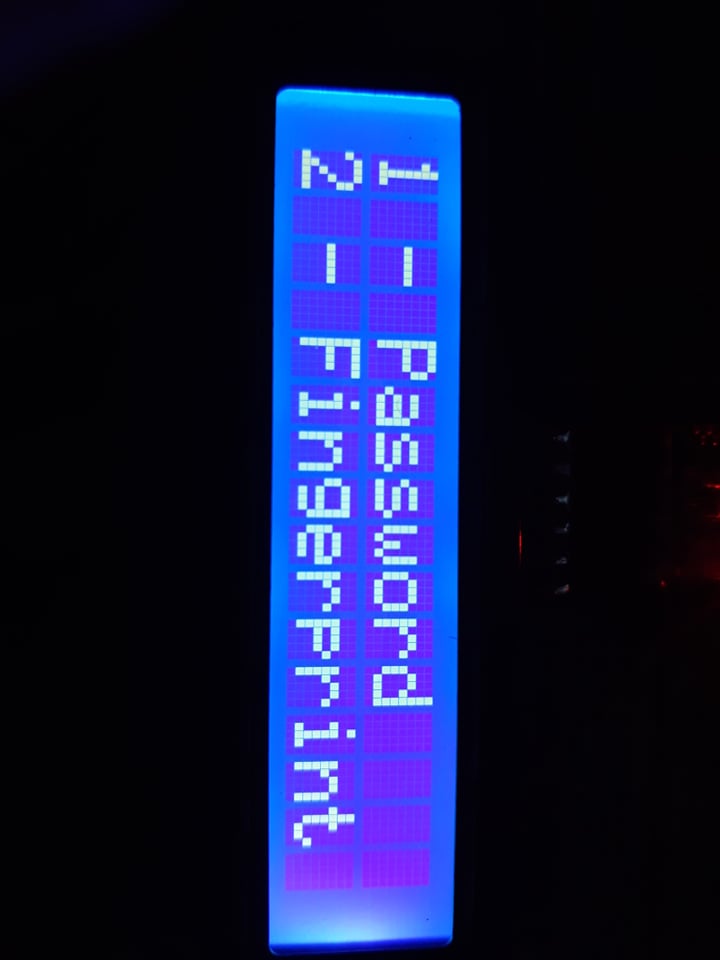
# **Chương 4: ỨNG DỤNG**

## **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**

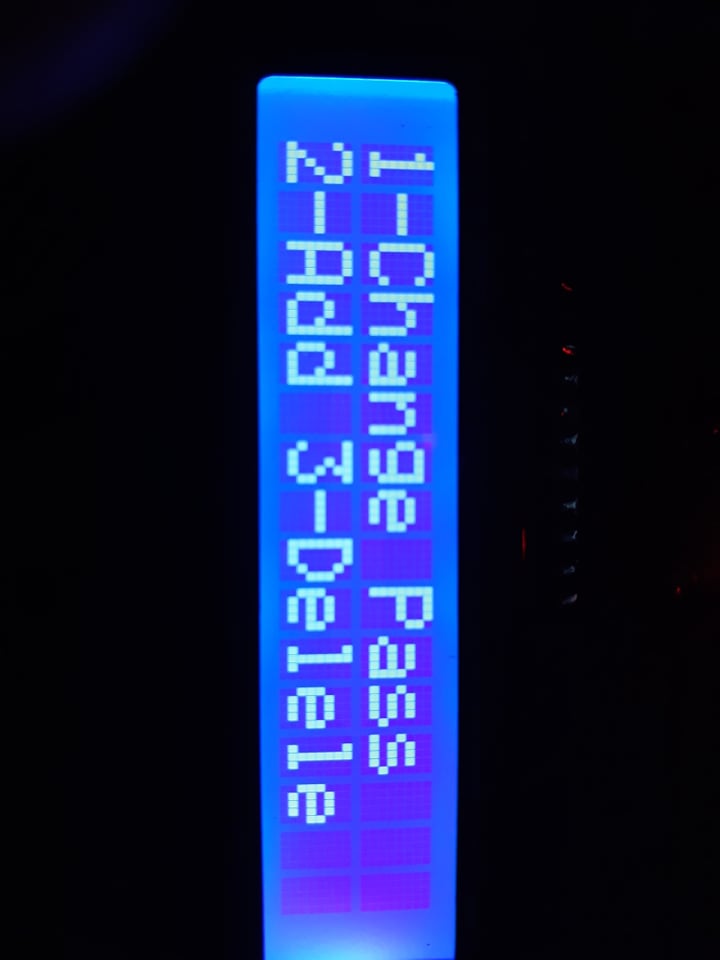
Ứng dụng sẽ có một bộ bàn phím để tương tác, ban đầu khi khởi động thiết bị ta có màn hình sau:



- Nhấn phím ‘A’ trên keypad để thực hiện các chức năng bảo mật của hệ thống mở khóa cửa với 2 sự lựa chọn là dùng password hoặc dấu vân tay.

Khi chọn lựa password và nhập pass, và nhập đúng password thì màn hình sẽ có thông báo và động cơ Servo sẽ hoạt động và xoay gọc 90o tương đương việc mở cửa.

Khi lựa chọn là dấu vân tay, thì cảm biến vân tay sẽ hoạt động, nhận dạng dấu vân tay để lấy ID gửi về cổng 12, 13 của Arduino, tương tự nếu đúng sẽ có thông báo và động cơ Servo hoạt động xoay để mở khóa cửa.

- Nhấn phím ‘B’ thì cảm biến vân tay sẽ hoạt động để xác nhận có phải là admin hay không, nêu chính xác thì ta có màn hình sau:

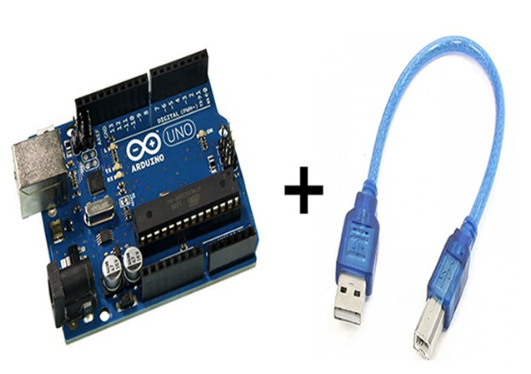
Ấn phím ‘1’ đê thực hiện chức năng đổi password.

Ấn phím ‘2’ để thực hiện thêm mới vân tay.

Ấn phím ‘3’ để thực hiện xóa vân tay được lưu trữ trong cảm biến.

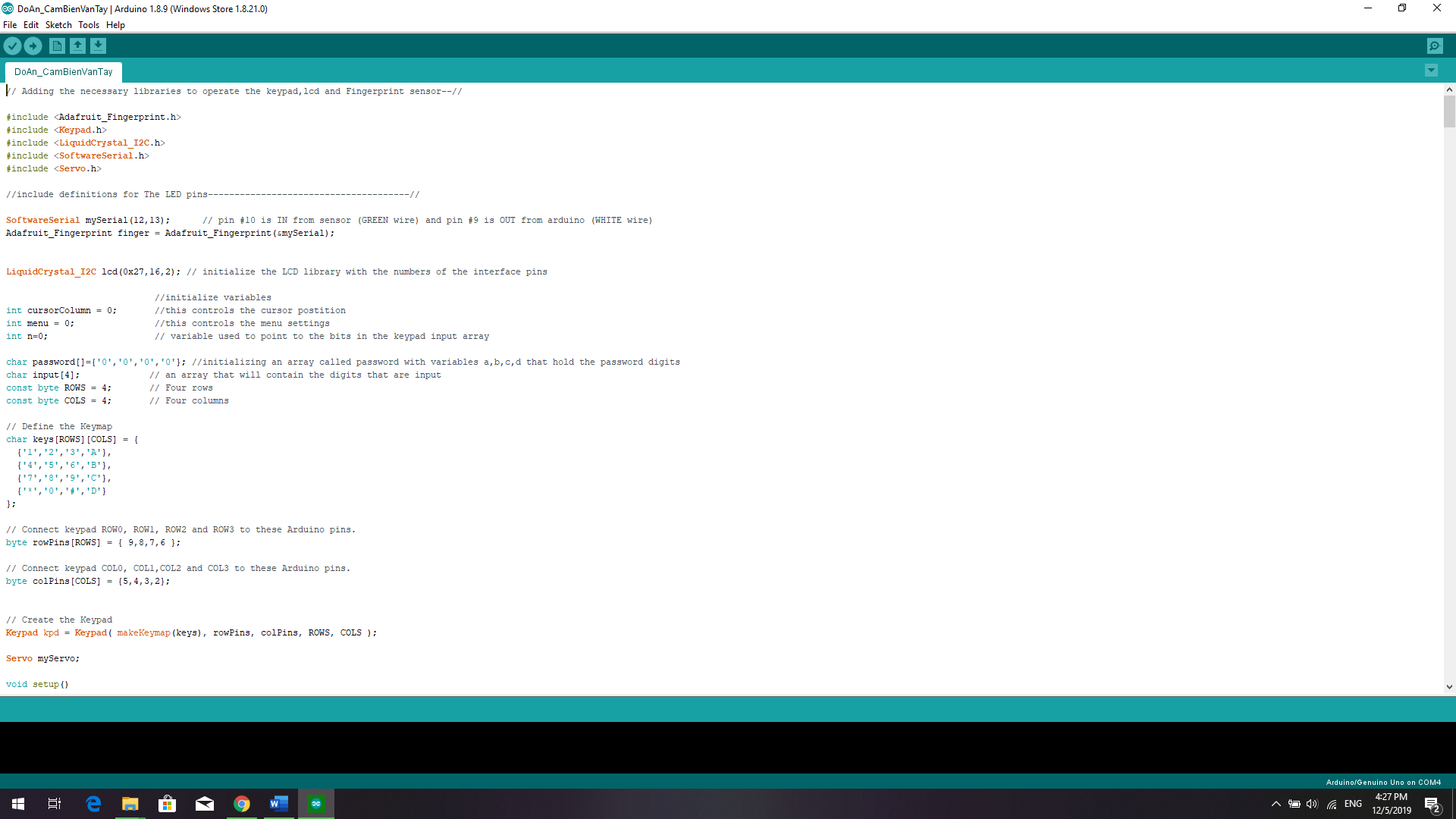
## **4.3 Cách nạp chương trình vào KIT**

Sử dụng Dây cáp USB 2 đầu Type A-B đê nạp chương trình vào KIT

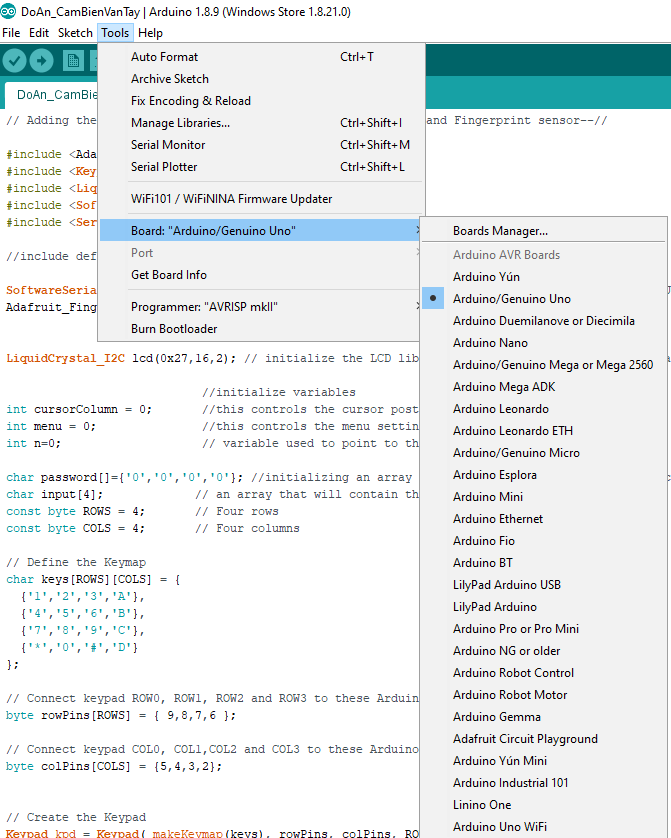


Bước 1: **Bước 1: Kết nối Arduino UNO R3 vào máy tính**

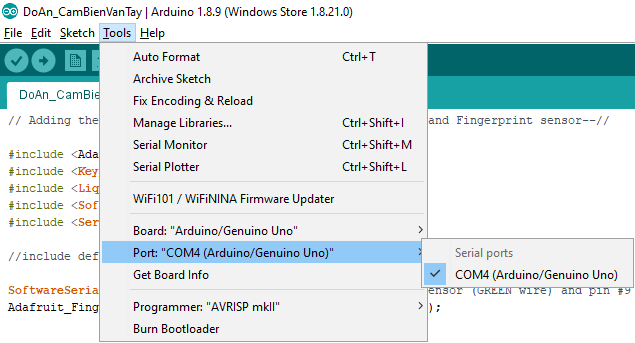
Bước 2: Chạy phần mềm Arduino IDE

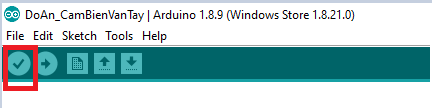


Bước 3: Chọn Tool -> Board -> chọn KIT phụ hợp (ở đây là Arduino Uno

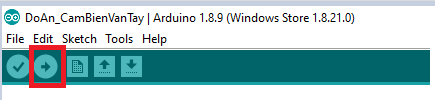


Bước 4: Kiểm Port: Chọn Tool -> Port -> chọn cổng nếu cần.

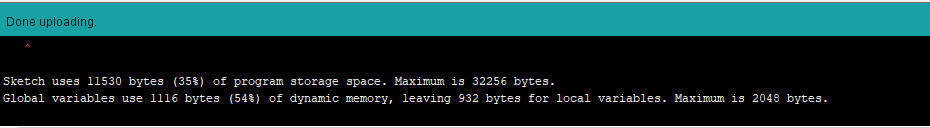


Bước 5: Để đảm bảo, ta bấm duyệt lại để xem code có lỗi hay không.

Bước 6: Nhấn Upload để nhúng code vào KIT.



Bước 7: Màn hình bên dưới như này thì hoàn tất việc nhúng code vào KIT.



## **4.4. Chương trình**

* + 1. **Khai báo, Settup, Loop**

|  |
| --- |
| // Adding the necessary libraries to operate the keypad, lcd and Fingerprint sensor--//  #include <Adafruit\_Fingerprint.h>  #include <Keypad.h>  #include <LiquidCrystal\_I2C.h>  #include <SoftwareSerial.h>  #include <Servo.h>    //include definitions for The LED pins--------------------------------------//  SoftwareSerial mySerial(12,13); // pin #10 is IN from sensor (GREEN wire) and pin #9 is OUT from arduino (WHITE wire)  Adafruit\_Fingerprint finger = Adafruit\_Fingerprint(&mySerial);    LiquidCrystal\_I2C lcd(0x27,16,2); // initialize the LCD library with the numbers of the interface pins  //initialize variables  int cursorColumn = 0; //this controls the cursor postition  int menu = 0; //this controls the menu settings  int n=0; // variable used to point to the bits in the keypad input array  char password[]={'0','0','0','0'}; //initializing an array called password with variables a,b,c,d that hold the password digits  char input[4]; // an array that will contain the digits that are input  const byte ROWS = 4; // Four rows  const byte COLS = 4; // Four columns  // Define the Keymap  char keys[ROWS][COLS] = {  {'1','2','3','A'},  {'4','5','6','B'},  {'7','8','9','C'},  {'\*','0','#','D'}  };  // Connect keypad ROW0, ROW1, ROW2 and ROW3 to these Arduino pins.  byte rowPins[ROWS] = { 9,8,7,6 };  // Connect keypad COL0, COL1,COL2 and COL3 to these Arduino pins.  byte colPins[COLS] = {5,4,3,2};  // Create the Keypad  Keypad kpd = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );  Servo myServo;  void setup()  {  // set the data rate for the sensor //Serial port  finger.begin(57600);  myServo.attach(10);  lcd.begin(); // set up the LCD's number of columns and rows:  // Prints a message to the LCD.  startScreen();  myServo.write(0);  }  void loop()  {  // myServo.write(0);  readKeypad(); // Handles the Keypad object and switch case to read the inputs and decides the output state and leds based on the input  } |

**4.4.2 Chương trình con**

|  |
| --- |
| void readKeypad()  {  char key = kpd.getKey();  lcd.setCursor(cursorColumn,1); // set the cursor to column "cursorColumn", line 1  if(key) // Check for a valid key.  {  switch (key)  {  case '0':  {  // Each case is a button that is pressed    switch(menu)  { // the value of "menu" determines the setting parameter and what each button does in that setting  case 0:  {  break;  }  default:  {  lcd.print('0');  cursorColumn=cursorColumn+1;  input[n]='0';  n=n+1;  break;  }  }  break;  }  case '1':  {  switch(menu)  { // the value of "menu" determines the setting parameter and what each button does in that setting  case 0:  {  break;  }  case 1:  {  menu = 2;  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Pin: ");  break;  }  case 5:  {  menu = 6;  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Enter old code");  lcd.setCursor(0,1);  break;  }  default:  {  lcd.print('1');  cursorColumn = cursorColumn + 1;  input[n]='1';  n=n+1;  break;  }  }  break;  }  case '2':  {  switch(menu)  { // the value of "menu" determines the setting parameter and what each button does in that setting  case 0:  {  break;  }  case 1:  {  menu = 3;  lcd.clear();  fingerCheck();  break;  }  case 5:  {  menu = 7;  lcd.clear();  lcd.setCursor(0,0);  finger.getTemplateCount();  int id = finger.templateCount+1;  lcd.print("Add ID ");  lcd.print(id);  lcd.setCursor(0,1);  lcd.print("Waiting ");  break;  }  default:  {  lcd.print('2');  cursorColumn=cursorColumn+1;  input[n]='2';  n=n+1;  break;  }  }  break;  }  case '3':  {  switch(menu)  { // the value of "menu" determines the setting parameter and what each button does in that setting  case 0:  {  break;  }  case 5:  {  menu = 8;  deleteEvent();  break;  }  default:  {  lcd.print('3');  cursorColumn=cursorColumn+1;  input[n]='3';  n=n+1;  break;  }  }  break;  }  case '4':  {  switch(menu)  { // the value of "menu" determines the setting parameter and what each button does in that setting  case 0:  {  break;  }  default:  {  lcd.print('4');  cursorColumn=cursorColumn+1;  input[n]='4';  n=n+1;  break;  }  }  break;  }    case '5':  {  switch(menu)  { // the value of "menu" determines the setting parameter and what each button does in that setting  case 0:  {  break;  }  default:  {  lcd.print('5');  cursorColumn=cursorColumn+1;  input[n]='5';  n=n+1;  break;  }  }  break;  }    case '6':  {  switch(menu)  { // the value of "menu" determines the setting parameter and what each button does in that setting  case 0:  {  break;  }  default:  {  lcd.print('6');  cursorColumn=cursorColumn+1;  input[n]='6';  n=n+1;  break;  }  }  break;  }  case '7':  {  switch(menu)  { // the value of "menu" determines the setting parameter and what each button does in that setting  case 0:  {  break;  }  default:  {  lcd.print('7');  cursorColumn=cursorColumn+1;  input[n]='7';  n=n+1;  break;  }  }  break;  }  case '8':  {  switch(menu)  { // the value of "menu" determines the setting parameter and what each button does in that setting  case 0:  {  break;  }  default:  {  lcd.print('8');  cursorColumn=cursorColumn+1;  input[n]='8';  n=n+1;  break;  }  }  break;  }  case '9':  {  switch(menu)  { // the value of "menu" determines the setting parameter and what each button does in that setting  case 0:  {  break;  }  default:  {  lcd.print('9');  cursorColumn=cursorColumn+1;  input[n]='9';  n=n+1;  break;  }  }  break;  }  case 'A':  {  if(menu == 0)  {  lcd.clear();  lcd.setCursor(0,0);  askForSelectionInAuthenicationMenu();  lcd.setCursor(0,1);  menu= 1;  }  break;  }  case 'B':  {  if(menu == 0)  {  lcd.clear();  lcd.print("Scan for Admin");  delay(2000);  menu= 4;  }  }  case 'C':  {  break;  }  case 'D':  {  reset();  break;  }  case '\*':  {  switch(menu)  {  case 0:  break;  case 8:  {  lcd.clear();  int id = convertCharAndInt(input);  lcd.clear();  lcd.print("Deleting ");  lcd.print(id);  delay(2000);  int flag = deleteFingerprint(id);  if( flag == 1)  {  lcd.clear();  lcd.print("Delete ");  lcd.print(id);  lcd.setCursor(0,1);  lcd.print("Successfully");  reset();  }  else  {  lcd.clear();  lcd.print("Some Error");  lcd.setCursor(0,1);  lcd.print("Cant delete ");  delay(500);  lcd.clear();  lcd.print("Go back to");  lcd.setCursor(0,1);  lcd.print("Admin Method ");  menu = 5;  }  break;  }  }  break;  }  default:  {  break;  }  }  }    switch(menu)  {  case 2:  {  if(n>3)  {  doorlockCheck(); //calls the function to check whether the code that was input matches the code that is stored  }  break;  }  case 3:  {  if(getFingerprintIDez() != -1)  {  reset();  }  break;  }  case 4:  {  authenicateAdministrator();  break;  }  case 6:  {  if(n > 3)  {  if(oldCodeCheck() == true)  {  menu=9;  }  }  break;  }  case 7:  {  getFingerprintEnroll();  }  case 9:  {  if( n>3)  {  changeToNewCode(password,input);  delay(250);  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Pass Changed");  delay(1000);  reset();  }  }  }  }  int convertCharAndInt(char A[4])  {  int tempNum = 0;  int tempCount = 0;  for (int i = n - 1; i >= 0; i--)  {  int temp = 1;  for (int j = 0; j < tempCount; j++)  {  temp = temp \* 10;  }  tempNum += (int)(A[i] - 48) \* temp;  tempCount++;  }  return tempNum;  }  void startScreen()  {  lcd.clear();  lcd.setCursor(0,0);  lcd.print("A Authenicating");  lcd.setCursor(0,1);  lcd.print("B Admin Methods");  }  void deleteEvent()  {  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Input ID");  lcd.setCursor(0,1);  lcd.print("You want to delete");  delay(1500);  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Enter by \* ");  lcd.setCursor(0,1);  n=0;  for(int i=0;i<4;i++)  {  input[i]='h';  }  }  void askForSelectionInAdminMethod()  {  lcd.clear();  lcd.setCursor(0,0);  lcd.print("1-Change Pass");  lcd.setCursor(0,1);  lcd.print("2-Add 3-Delele");  }  void authenicateAdministrator()  {  int idadmin = getFingerprintIDez();  if(idadmin != 1 && idadmin != 2 )  {  lcd.clear();  lcd.setCursor(0,0);  lcd.print(" Not Exactly ! ");  lcd.setCursor(0,1);  lcd.print("Try Again");  }  else  {  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Hello Admin! ");  delay(1000);  menu = 5;  askForSelectionInAdminMethod();  }  }  void askForSelectionInAuthenicationMenu()  {  lcd.clear();  lcd.setCursor(0,0);  lcd.print("1 - Password"); //prints when the user wants to enter the code  lcd.setCursor(0,1);  lcd.print("2 - Fingerprint");  }  boolean checkCode(char \*a,char \*b)  { //The function to check whether the contents of the first parameter,an array, match the //match the contents of the second parameter, also an array.  for(int p=0; p<4; p++)  {  if(a[p]!=b[p]) return false;  }  return true;  }  int changeToNewCode(char \*a, char \*b)  {  for(int p=0; p<4; p++)  {  a[p]=b[p];  }  n=0;  }  int doorlockCheck()  {  if(n > 3)  {  if(checkCode(password,input) == true)  {  delay(250);  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Correct!");  myServo.write(90);  delay(7500);  myServo.write(0);  reset();  }  else  {  delay(250);  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Invalid pass !");    delay(2000);  reset();  }  cursorColumn=0;  n=0;  }  }  bool oldCodeCheck()  {  if(checkCode(password,input) == true)  {  delay(250);  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Correct!");  delay(300);  n=0;  for(int i=0;i<4;i++)  {  input[i]='r';  }  lcd.clear();  lcd.print("Enter new pass: ");  lcd.setCursor(0,1);  return true;  }  else  {  delay(250);  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Invalid pass!");  delay(2000);  return false;  }  }  void reset()  {  int i;  cursorColumn=0;  startScreen();  menu=0;  n=0;  for(int i=0;i<4;i++)  {  input[i]='h';  }  }  void fingerCheck()  {  if (finger.verifyPassword())  {  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Found sensor!");  }  else  {  lcd.clear();  lcd.setCursor(0,0);  lcd.print("Fingersensor not found :(");  while (1);  }  lcd.setCursor(0,1);  lcd.print("Waiting...");  return ;  }  // returns -1 if failed, otherwise returns ID #  int getFingerprintIDez()  {  uint8\_t p = finger.getImage();  if (p != FINGERPRINT\_OK) return -1;  p = finger.image2Tz();  if (p != FINGERPRINT\_OK) return -1;  p = finger.fingerFastSearch();  if (p != FINGERPRINT\_OK) return -1;    // found a match!  if(menu!=4)  {  lcd.clear();  lcd.print("Found ID #"); lcd.print(finger.fingerID);  lcd.setCursor(0,1);  lcd.print("Confidence "); lcd.print(finger.confidence);  myServo.write(90);  delay(7500);  myServo.write(0);  reset();  }  return finger.fingerID;  }  uint8\_t deleteFingerprint(uint8\_t id)  {  uint8\_t p = -1;    p = finger.deleteModel(id);  if (p == FINGERPRINT\_OK)  {  //Serial.println("Deleted!");  return 1;  }  else if (p == FINGERPRINT\_PACKETRECIEVEERR)  {  //Serial.println("Communication error");  return p;  }  else if (p == FINGERPRINT\_BADLOCATION)  {  //Serial.println("Could not delete in that location");  return p;  }  else if (p == FINGERPRINT\_FLASHERR)  {  //Serial.println("Error writing to flash");  return p;  }  else  {  //Serial.print("Unknown error: 0x"); //Serial.println(p, HEX);  return p;  }  }  uint8\_t getFingerprintEnroll()  {  int p = -1;  finger.getTemplateCount();  int id = finger.templateCount+1;  //Serial.print("Waiting for valid finger to enroll as #");  //Serial.println(id);  while (p != FINGERPRINT\_OK)  {  p = finger.getImage();  switch (p)  {  case FINGERPRINT\_OK:  //Serial.println("Image taken");  break;  case FINGERPRINT\_NOFINGER:  //Serial.println(".");  break;  case FINGERPRINT\_PACKETRECIEVEERR:  //Serial.println("Communication error");  break;  case FINGERPRINT\_IMAGEFAIL:  //Serial.println("Imaging error");  break;  default:  //Serial.println("Unknown error");  break;  }  }  // OK success!  p = finger.image2Tz(1);  switch (p)  {  case FINGERPRINT\_OK:  //Serial.println("Image converted");  break;  case FINGERPRINT\_IMAGEMESS:  //Serial.println("Image too messy");  return p;  case FINGERPRINT\_PACKETRECIEVEERR:  //Serial.println("Communication error");  return p;  case FINGERPRINT\_FEATUREFAIL:  //Serial.println("Could not find fingerprint features");  return p;  case FINGERPRINT\_INVALIDIMAGE:  //Serial.println("Could not find fingerprint features");  return p;  default:  //Serial.println("Unknown error");  return p;  }  lcd.clear();  lcd.print("Remove finger");  //Serial.println("Remove finger");  delay(2000);  p = 0;  while (p != FINGERPRINT\_NOFINGER)  {  p = finger.getImage();  }  //Serial.print("ID "); //Serial.println(id);  p = -1;  lcd.clear();  lcd.print("Place same finger again");  //Serial.println("Place same finger again");  while (p != FINGERPRINT\_OK)  {  p = finger.getImage();  switch (p)  {  case FINGERPRINT\_OK:  //Serial.println("Image taken");  break;  case FINGERPRINT\_NOFINGER:  //Serial.print(".");  break;  case FINGERPRINT\_PACKETRECIEVEERR:  //Serial.println("Communication error");  break;  case FINGERPRINT\_IMAGEFAIL:  //Serial.println("Imaging error");  break;  default:  //Serial.println("Unknown error");  break;  }  }  // OK success!  p = finger.image2Tz(2);  switch (p)  {  case FINGERPRINT\_OK:  //Serial.println("Image converted");  break;  case FINGERPRINT\_IMAGEMESS:  //Serial.println("Image too messy");  return p;  case FINGERPRINT\_PACKETRECIEVEERR:  //Serial.println("Communication error");  return p;  case FINGERPRINT\_FEATUREFAIL:  //Serial.println("Could not find fingerprint features");  return p;  case FINGERPRINT\_INVALIDIMAGE:  //Serial.println("Could not find fingerprint features");  return p;  default:  //Serial.println("Unknown error");  return p;  }    // OK converted!  //Serial.print("Creating model for #"); //Serial.println(id);    p = finger.createModel();  if (p == FINGERPRINT\_OK)  {  //Serial.println("Prints matched!");  }  else if (p == FINGERPRINT\_PACKETRECIEVEERR)  {  ////Serial.println("Communication error");  return p;  }  else if (p == FINGERPRINT\_ENROLLMISMATCH)  {  //Serial.println("Fingerprints did not match");  return p;  }  else  {  //Serial.println("Unknown error");  return p;  }    //Serial.print("ID ");  //Serial.println(id);  p = finger.storeModel(id);  if (p == FINGERPRINT\_OK)  {  //Serial.println("Stored!");  }  else if (p == FINGERPRINT\_PACKETRECIEVEERR)  {  //Serial.println("Communication error");  return p;  }  else if (p == FINGERPRINT\_BADLOCATION)  {  //Serial.println("Could not store in that location");  return p;  }  else if (p == FINGERPRINT\_FLASHERR)  {  //Serial.println("Error writing to flash");  return p;  }  else  {  //Serial.println("Unknown error");  return p;  }  lcd.clear();  lcd.print("Add Successfully");  lcd.setCursor(0,1);  lcd.print("Identity no: ");  lcd.print(id);  delay(6000);  reset();  } |

* + 1. **Thư viện mẫu**

**ADAFRUIT\_FINGERPRINT\_H**

|  |
| --- |
| #ifndef ADAFRUIT\_FINGERPRINT\_H  #define ADAFRUIT\_FINGERPRINT\_H  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  This is a library for our optical Fingerprint sensor  Designed specifically to work with the Adafruit Fingerprint sensor  ----> http://www.adafruit.com/products/751  These displays use TTL Serial to communicate, 2 pins are required to  interface  Adafruit invests time and resources providing this open source code,  please support Adafruit and open-source hardware by purchasing  products from Adafruit!  Written by Limor Fried/Ladyada for Adafruit Industries.  BSD license, all text above must be included in any redistribution  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "Arduino.h"  #if defined(\_\_AVR\_\_) || defined(ESP8266)  #include <SoftwareSerial.h>  #elif defined(FREEDOM\_E300\_HIFIVE1)  #include <SoftwareSerial32.h>  #define SoftwareSerial SoftwareSerial32  #endif  #define FINGERPRINT\_OK 0x00  #define FINGERPRINT\_PACKETRECIEVEERR 0x01  #define FINGERPRINT\_NOFINGER 0x02  #define FINGERPRINT\_IMAGEFAIL 0x03  #define FINGERPRINT\_IMAGEMESS 0x06  #define FINGERPRINT\_FEATUREFAIL 0x07  #define FINGERPRINT\_NOMATCH 0x08  #define FINGERPRINT\_NOTFOUND 0x09  #define FINGERPRINT\_ENROLLMISMATCH 0x0A  #define FINGERPRINT\_BADLOCATION 0x0B  #define FINGERPRINT\_DBRANGEFAIL 0x0C  #define FINGERPRINT\_UPLOADFEATUREFAIL 0x0D  #define FINGERPRINT\_PACKETRESPONSEFAIL 0x0E  #define FINGERPRINT\_UPLOADFAIL 0x0F  #define FINGERPRINT\_DELETEFAIL 0x10  #define FINGERPRINT\_DBCLEARFAIL 0x11  #define FINGERPRINT\_PASSFAIL 0x13  #define FINGERPRINT\_INVALIDIMAGE 0x15  #define FINGERPRINT\_FLASHERR 0x18  #define FINGERPRINT\_INVALIDREG 0x1A  #define FINGERPRINT\_ADDRCODE 0x20  #define FINGERPRINT\_PASSVERIFY 0x21  #define FINGERPRINT\_STARTCODE 0xEF01  #define FINGERPRINT\_COMMANDPACKET 0x1  #define FINGERPRINT\_DATAPACKET 0x2  #define FINGERPRINT\_ACKPACKET 0x7  #define FINGERPRINT\_ENDDATAPACKET 0x8  #define FINGERPRINT\_TIMEOUT 0xFF  #define FINGERPRINT\_BADPACKET 0xFE  #define FINGERPRINT\_GETIMAGE 0x01  #define FINGERPRINT\_IMAGE2TZ 0x02  #define FINGERPRINT\_REGMODEL 0x05  #define FINGERPRINT\_STORE 0x06  #define FINGERPRINT\_LOAD 0x07  #define FINGERPRINT\_UPLOAD 0x08  #define FINGERPRINT\_DELETE 0x0C  #define FINGERPRINT\_EMPTY 0x0D  #define FINGERPRINT\_SETPASSWORD 0x12  #define FINGERPRINT\_VERIFYPASSWORD 0x13  #define FINGERPRINT\_HISPEEDSEARCH 0x1B  #define FINGERPRINT\_TEMPLATECOUNT 0x1D  //#define FINGERPRINT\_DEBUG  #define DEFAULTTIMEOUT 1000 ///< UART reading timeout in milliseconds  ///! Helper class to craft UART packets  struct Adafruit\_Fingerprint\_Packet {  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*!  @brief Create a new UART-borne packet  @param type Command, data, ack type packet  @param length Size of payload  @param data Pointer to bytes of size length we will memcopy into the internal buffer  \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  Adafruit\_Fingerprint\_Packet(uint8\_t type, uint16\_t length, uint8\_t \* data) {  this->start\_code = FINGERPRINT\_STARTCODE;  this->type = type;  this->length = length;  address[0] = 0xFF; address[1] = 0xFF;  address[2] = 0xFF; address[3] = 0xFF;  if(length<64)  memcpy(this->data, data, length);  else  memcpy(this->data, data, 64);  }  uint16\_t start\_code; ///< "Wakeup" code for packet detection  uint8\_t address[4]; ///< 32-bit Fingerprint sensor address  uint8\_t type; ///< Type of packet  uint16\_t length; ///< Length of packet  uint8\_t data[64]; ///< The raw buffer for packet payload  };  ///! Helper class to communicate with and keep state for fingerprint sensors  class Adafruit\_Fingerprint {  public:  #if defined(\_\_AVR\_\_) || defined(ESP8266) || defined(FREEDOM\_E300\_HIFIVE1)  Adafruit\_Fingerprint(SoftwareSerial \*ss, uint32\_t password = 0x0);  #endif  Adafruit\_Fingerprint(HardwareSerial \*hs, uint32\_t password = 0x0);  void begin(uint32\_t baud);  boolean verifyPassword(void);  uint8\_t getImage(void);  uint8\_t image2Tz(uint8\_t slot = 1);  uint8\_t createModel(void);  uint8\_t emptyDatabase(void);  uint8\_t storeModel(uint16\_t id);  uint8\_t loadModel(uint16\_t id);  uint8\_t getModel(void);  uint8\_t deleteModel(uint16\_t id);  uint8\_t fingerFastSearch(void);  uint8\_t getTemplateCount(void);  uint8\_t setPassword(uint32\_t password);  void writeStructuredPacket(const Adafruit\_Fingerprint\_Packet & p);  uint8\_t getStructuredPacket(Adafruit\_Fingerprint\_Packet \* p, uint16\_t timeout=DEFAULTTIMEOUT);  /// The matching location that is set by fingerFastSearch()  uint16\_t fingerID;  /// The confidence of the fingerFastSearch() match, higher numbers are more confidents  uint16\_t confidence;  /// The number of stored templates in the sensor, set by getTemplateCount()  uint16\_t templateCount;  private:  uint8\_t checkPassword(void);  uint32\_t thePassword;  uint32\_t theAddress;  uint8\_t recvPacket[20];  Stream \*mySerial;  #if defined(\_\_AVR\_\_) || defined(ESP8266) || defined(FREEDOM\_E300\_HIFIVE1)  SoftwareSerial \*swSerial;  #endif  HardwareSerial \*hwSerial;  };  #endif |

**LiquidCrystal\_I2C**

|  |
| --- |
| #ifndef FDB\_LIQUID\_CRYSTAL\_I2C\_H  #define FDB\_LIQUID\_CRYSTAL\_I2C\_H  #include <inttypes.h>  #include <Print.h>  // commands  #define LCD\_CLEARDISPLAY 0x01  #define LCD\_RETURNHOME 0x02  #define LCD\_ENTRYMODESET 0x04  #define LCD\_DISPLAYCONTROL 0x08  #define LCD\_CURSORSHIFT 0x10  #define LCD\_FUNCTIONSET 0x20  #define LCD\_SETCGRAMADDR 0x40  #define LCD\_SETDDRAMADDR 0x80  // flags for display entry mode  #define LCD\_ENTRYRIGHT 0x00  #define LCD\_ENTRYLEFT 0x02  #define LCD\_ENTRYSHIFTINCREMENT 0x01  #define LCD\_ENTRYSHIFTDECREMENT 0x00  // flags for display on/off control  #define LCD\_DISPLAYON 0x04  #define LCD\_DISPLAYOFF 0x00  #define LCD\_CURSORON 0x02  #define LCD\_CURSOROFF 0x00  #define LCD\_BLINKON 0x01  #define LCD\_BLINKOFF 0x00  // flags for display/cursor shift  #define LCD\_DISPLAYMOVE 0x08  #define LCD\_CURSORMOVE 0x00  #define LCD\_MOVERIGHT 0x04  #define LCD\_MOVELEFT 0x00  // flags for function set  #define LCD\_8BITMODE 0x10  #define LCD\_4BITMODE 0x00  #define LCD\_2LINE 0x08  #define LCD\_1LINE 0x00  #define LCD\_5x10DOTS 0x04  #define LCD\_5x8DOTS 0x00  // flags for backlight control  #define LCD\_BACKLIGHT 0x08  #define LCD\_NOBACKLIGHT 0x00  #define En B00000100 // Enable bit  #define Rw B00000010 // Read/Write bit  #define Rs B00000001 // Register select bit  /\*\*  \* This is the driver for the Liquid Crystal LCD displays that use the I2C bus.  \*  \* After creating an instance of this class, first call begin() before anything else.  \* The backlight is on by default, since that is the most likely operating mode in  \* most cases.  \*/  class LiquidCrystal\_I2C : public Print {  public:  /\*\*  \* Constructor  \*  \* @param lcd\_addr I2C slave address of the LCD display. Most likely printed on the  \* LCD circuit board, or look in the supplied LCD documentation.  \* @param lcd\_cols Number of columns your LCD display has.  \* @param lcd\_rows Number of rows your LCD display has.  \* @param charsize The size in dots that the display has, use LCD\_5x10DOTS or LCD\_5x8DOTS.  \*/  LiquidCrystal\_I2C(uint8\_t lcd\_addr, uint8\_t lcd\_cols, uint8\_t lcd\_rows, uint8\_t charsize = LCD\_5x8DOTS);  /\*\*  \* Set the LCD display in the correct begin state, must be called before anything else is done.  \*/  void begin();  /\*\*  \* Remove all the characters currently shown. Next print/write operation will start  \* from the first position on LCD display.  \*/  void clear();  /\*\*  \* Next print/write operation will will start from the first position on the LCD display.  \*/  void home();  /\*\*  \* Do not show any characters on the LCD display. Backlight state will remain unchanged.  \* Also all characters written on the display will return, when the display in enabled again.  \*/  void noDisplay();  /\*\*  \* Show the characters on the LCD display, this is the normal behaviour. This method should  \* only be used after noDisplay() has been used.  \*/  void display();  /\*\*  \* Do not blink the cursor indicator.  \*/  void noBlink();  /\*\*  \* Start blinking the cursor indicator.  \*/  void blink();  /\*\*  \* Do not show a cursor indicator.  \*/  void noCursor();  /\*\*  \* Show a cursor indicator, cursor can blink on not blink. Use the  \* methods blink() and noBlink() for changing cursor blink.  \*/  void cursor();  void scrollDisplayLeft();  void scrollDisplayRight();  void printLeft();  void printRight();  void leftToRight();  void rightToLeft();  void shiftIncrement();  void shiftDecrement();  void noBacklight();  void backlight();  bool getBacklight();  void autoscroll();  void noAutoscroll();  void createChar(uint8\_t, uint8\_t[]);  void setCursor(uint8\_t, uint8\_t);  virtual size\_t write(uint8\_t);  void command(uint8\_t);  inline void blink\_on() { blink(); }  inline void blink\_off() { noBlink(); }  inline void cursor\_on() { cursor(); }  inline void cursor\_off() { noCursor(); }  // Compatibility API function aliases  void setBacklight(uint8\_t new\_val); // alias for backlight() and nobacklight()  void load\_custom\_character(uint8\_t char\_num, uint8\_t \*rows); // alias for createChar()  void printstr(const char[]);  private:  void send(uint8\_t, uint8\_t);  void write4bits(uint8\_t);  void expanderWrite(uint8\_t);  void pulseEnable(uint8\_t);  uint8\_t \_addr;  uint8\_t \_displayfunction;  uint8\_t \_displaycontrol;  uint8\_t \_displaymode;  uint8\_t \_cols;  uint8\_t \_rows;  uint8\_t \_charsize;  uint8\_t \_backlightval;  };  #endif // FDB\_LIQUID\_CRYSTAL\_I2C\_H |

**Keypad**

|  |
| --- |
| /\*  ||  || @file Keypad.h  || @version 3.1  || @author Mark Stanley, Alexander Brevig  || @contact mstanley@technologist.com, alexanderbrevig@gmail.com  ||  || @description  || | This library provides a simple interface for using matrix  || | keypads. It supports multiple keypresses while maintaining  || | backwards compatibility with the old single key library.  || | It also supports user selectable pins and definable keymaps.  || #  ||  || @license  || | This library is free software; you can redistribute it and/or  || | modify it under the terms of the GNU Lesser General Public  || | License as published by the Free Software Foundation; version  || | 2.1 of the License.  || |  || | This library is distributed in the hope that it will be useful,  || | but WITHOUT ANY WARRANTY; without even the implied warranty of  || | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU  || | Lesser General Public License for more details.  || |  || | You should have received a copy of the GNU Lesser General Public  || | License along with this library; if not, write to the Free Software  || | Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA  || #  ||  \*/  #ifndef KEYPAD\_H  #define KEYPAD\_H  #include "Key.h"  // bperrybap - Thanks for a well reasoned argument and the following macro(s).  // See http://arduino.cc/forum/index.php/topic,142041.msg1069480.html#msg1069480  #ifndef INPUT\_PULLUP  #warning "Using pinMode() INPUT\_PULLUP AVR emulation"  #define INPUT\_PULLUP 0x2  #define pinMode(\_pin, \_mode) \_mypinMode(\_pin, \_mode)  #define \_mypinMode(\_pin, \_mode) \  do { \  if(\_mode == INPUT\_PULLUP) \  pinMode(\_pin, INPUT); \  digitalWrite(\_pin, 1); \  if(\_mode != INPUT\_PULLUP) \  pinMode(\_pin, \_mode); \  }while(0)  #endif  #define OPEN LOW  #define CLOSED HIGH  typedef char KeypadEvent;  typedef unsigned int uint;  typedef unsigned long ulong;  // Made changes according to this post http://arduino.cc/forum/index.php?topic=58337.0  // by Nick Gammon. Thanks for the input Nick. It actually saved 78 bytes for me. :)  typedef struct {  byte rows;  byte columns;  } KeypadSize;  #define LIST\_MAX 10 // Max number of keys on the active list.  #define MAPSIZE 10 // MAPSIZE is the number of rows (times 16 columns)  #define makeKeymap(x) ((char\*)x)  //class Keypad : public Key, public HAL\_obj {  class Keypad : public Key {  public:  Keypad(char \*userKeymap, byte \*row, byte \*col, byte numRows, byte numCols);  virtual void pin\_mode(byte pinNum, byte mode) { pinMode(pinNum, mode); }  virtual void pin\_write(byte pinNum, boolean level) { digitalWrite(pinNum, level); }  virtual int pin\_read(byte pinNum) { return digitalRead(pinNum); }  uint bitMap[MAPSIZE]; // 10 row x 16 column array of bits. Except Due which has 32 columns.  Key key[LIST\_MAX];  unsigned long holdTimer;  char getKey();  bool getKeys();  KeyState getState();  void begin(char \*userKeymap);  bool isPressed(char keyChar);  void setDebounceTime(uint);  void setHoldTime(uint);  void addEventListener(void (\*listener)(char));  int findInList(char keyChar);  int findInList(int keyCode);  char waitForKey();  bool keyStateChanged();  byte numKeys();  private:  unsigned long startTime;  char \*keymap;  byte \*rowPins;  byte \*columnPins;  KeypadSize sizeKpd;  uint debounceTime;  uint holdTime;  bool single\_key;  void scanKeys();  bool updateList();  void nextKeyState(byte n, boolean button);  void transitionTo(byte n, KeyState nextState);  void (\*keypadEventListener)(char);  };  #endif  /\*  || @changelog  || | 3.1 2013-01-15 - Mark Stanley : Fixed missing RELEASED & IDLE status when using a single key.  || | 3.0 2012-07-12 - Mark Stanley : Made library multi-keypress by default. (Backwards compatible)  || | 3.0 2012-07-12 - Mark Stanley : Modified pin functions to support Keypad\_I2C  || | 3.0 2012-07-12 - Stanley & Young : Removed static variables. Fix for multiple keypad objects.  || | 3.0 2012-07-12 - Mark Stanley : Fixed bug that caused shorted pins when pressing multiple keys.  || | 2.0 2011-12-29 - Mark Stanley : Added waitForKey().  || | 2.0 2011-12-23 - Mark Stanley : Added the public function keyStateChanged().  || | 2.0 2011-12-23 - Mark Stanley : Added the private function scanKeys().  || | 2.0 2011-12-23 - Mark Stanley : Moved the Finite State Machine into the function getKeyState().  || | 2.0 2011-12-23 - Mark Stanley : Removed the member variable lastUdate. Not needed after rewrite.  || | 1.8 2011-11-21 - Mark Stanley : Added test to determine which header file to compile,  || | WProgram.h or Arduino.h.  || | 1.8 2009-07-08 - Alexander Brevig : No longer uses arrays  || | 1.7 2009-06-18 - Alexander Brevig : This library is a Finite State Machine every time a state changes  || | the keypadEventListener will trigger, if set  || | 1.7 2009-06-18 - Alexander Brevig : Added setDebounceTime setHoldTime specifies the amount of  || | microseconds before a HOLD state triggers  || | 1.7 2009-06-18 - Alexander Brevig : Added transitionTo  || | 1.6 2009-06-15 - Alexander Brevig : Added getState() and state variable  || | 1.5 2009-05-19 - Alexander Brevig : Added setHoldTime()  || | 1.4 2009-05-15 - Alexander Brevig : Added addEventListener  || | 1.3 2009-05-12 - Alexander Brevig : Added lastUdate, in order to do simple debouncing  || | 1.2 2009-05-09 - Alexander Brevig : Changed getKey()  || | 1.1 2009-04-28 - Alexander Brevig : Modified API, and made variables private  || | 1.0 2007-XX-XX - Mark Stanley : Initial Release  || #  \*/ |

**Servo**

|  |
| --- |
| /\*  Servo.h - Interrupt driven Servo library for Arduino using 16 bit timers- Version 2  Copyright (c) 2009 Michael Margolis. All right reserved.  This library is free software; you can redistribute it and/or  modify it under the terms of the GNU Lesser General Public  License as published by the Free Software Foundation; either  version 2.1 of the License, or (at your option) any later version.  This library is distributed in the hope that it will be useful,  but WITHOUT ANY WARRANTY; without even the implied warranty of  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU  Lesser General Public License for more details.  You should have received a copy of the GNU Lesser General Public  License along with this library; if not, write to the Free Software  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA  \*/  /\*  A servo is activated by creating an instance of the Servo class passing  the desired pin to the attach() method.  The servos are pulsed in the background using the value most recently  written using the write() method.  Note that analogWrite of PWM on pins associated with the timer are  disabled when the first servo is attached.  Timers are seized as needed in groups of 12 servos - 24 servos use two  timers, 48 servos will use four.  The sequence used to sieze timers is defined in timers.h  The methods are:  Servo - Class for manipulating servo motors connected to Arduino pins.  attach(pin ) - Attaches a servo motor to an i/o pin.  attach(pin, min, max ) - Attaches to a pin setting min and max values in microseconds  default min is 544, max is 2400    write() - Sets the servo angle in degrees. (invalid angle that is valid as pulse in microseconds is treated as microseconds)  writeMicroseconds() - Sets the servo pulse width in microseconds  read() - Gets the last written servo pulse width as an angle between 0 and 180.  readMicroseconds() - Gets the last written servo pulse width in microseconds. (was read\_us() in first release)  attached() - Returns true if there is a servo attached.  detach() - Stops an attached servos from pulsing its i/o pin.  \*/  #ifndef Servo\_h  #define Servo\_h  #include <inttypes.h>  /\*  \* Defines for 16 bit timers used with Servo library  \*  \* If \_useTimerX is defined then TimerX is a 16 bit timer on the current board  \* timer16\_Sequence\_t enumerates the sequence that the timers should be allocated  \* \_Nbr\_16timers indicates how many 16 bit timers are available.  \*/  // Architecture specific include  #if defined(ARDUINO\_ARCH\_AVR)  #include "avr/ServoTimers.h"  #elif defined(ARDUINO\_ARCH\_SAM)  #include "sam/ServoTimers.h"  #elif defined(ARDUINO\_ARCH\_SAMD)  #include "samd/ServoTimers.h"  #elif defined(ARDUINO\_ARCH\_STM32F4)  #include "stm32f4/ServoTimers.h"  #elif defined(ARDUINO\_ARCH\_NRF52)  #include "nrf52/ServoTimers.h"  #elif defined(ARDUINO\_ARCH\_MEGAAVR)  #include "megaavr/ServoTimers.h"  #elif defined(ARDUINO\_ARCH\_MBED)  #include "mbed/ServoTimers.h"  #else  #error "This library only supports boards with an AVR, SAM, SAMD, NRF52 or STM32F4 processor."  #endif  #define Servo\_VERSION 2 // software version of this library  #define MIN\_PULSE\_WIDTH 544 // the shortest pulse sent to a servo  #define MAX\_PULSE\_WIDTH 2400 // the longest pulse sent to a servo  #define DEFAULT\_PULSE\_WIDTH 1500 // default pulse width when servo is attached  #define REFRESH\_INTERVAL 20000 // minumim time to refresh servos in microseconds  #define SERVOS\_PER\_TIMER 12 // the maximum number of servos controlled by one timer  #define MAX\_SERVOS (\_Nbr\_16timers \* SERVOS\_PER\_TIMER)  #define INVALID\_SERVO 255 // flag indicating an invalid servo index  #if !defined(ARDUINO\_ARCH\_STM32F4)  typedef struct {  uint8\_t nbr :6 ; // a pin number from 0 to 63  uint8\_t isActive :1 ; // true if this channel is enabled, pin not pulsed if false  } ServoPin\_t ;  typedef struct {  ServoPin\_t Pin;  volatile unsigned int ticks;  } servo\_t;  class Servo  {  public:  Servo();  uint8\_t attach(int pin); // attach the given pin to the next free channel, sets pinMode, returns channel number or 0 if failure  uint8\_t attach(int pin, int min, int max); // as above but also sets min and max values for writes.  void detach();  void write(int value); // if value is < 200 its treated as an angle, otherwise as pulse width in microseconds  void writeMicroseconds(int value); // Write pulse width in microseconds  int read(); // returns current pulse width as an angle between 0 and 180 degrees  int readMicroseconds(); // returns current pulse width in microseconds for this servo (was read\_us() in first release)  bool attached(); // return true if this servo is attached, otherwise false  private:  uint8\_t servoIndex; // index into the channel data for this servo  int8\_t min; // minimum is this value times 4 added to MIN\_PULSE\_WIDTH  int8\_t max; // maximum is this value times 4 added to MAX\_PULSE\_WIDTH  };  #endif  #endif |

# **KẾT LUẬN**

1. **Kết quả đạt được**

* Hiểu được một số thành phần của KIT Arduino Uno.
* Hiểu được cấu trúc bộ cảm biến vân tay R305 và cách giao tiếp giữa cảm biến vân tay R305 và KIT.
* Hiểu được cách thức hoạt động của Keypad, LCD, động cơ Servo.
* Biết cách nhúng Code vào KIT.

1. **Ưu, nhược điểm**
2. **Ưu điểm**

* Tài liệu về Arduino dễ tìm kiếm, dễ nghiên cứu, nhiều thư viện hỗ trợ.
* Các hướng dẫn về Arduino nhiều, đa dạng.
* Tốc độ xử lý nhanh, ổn định.
* Xử dụng ngôn ngữ C không quá phức tạp.
* Tài nguyên trên KIT lớn, phù hợp với các ứng dụng khác nhau.

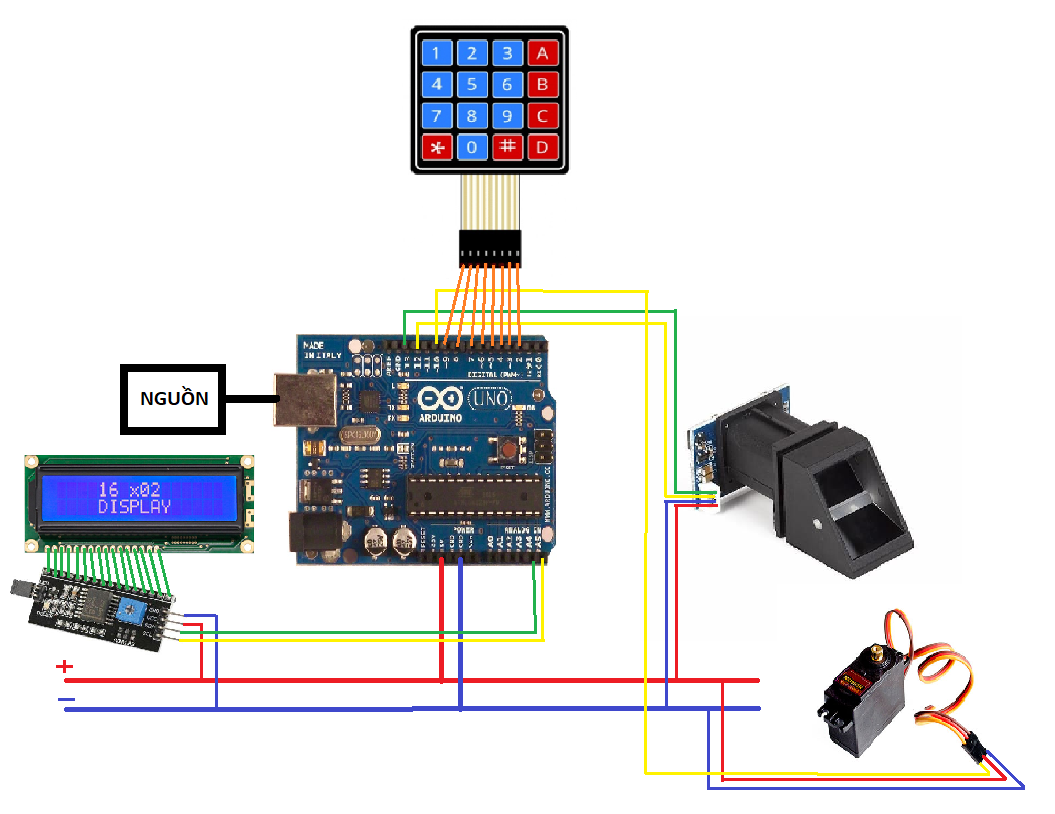
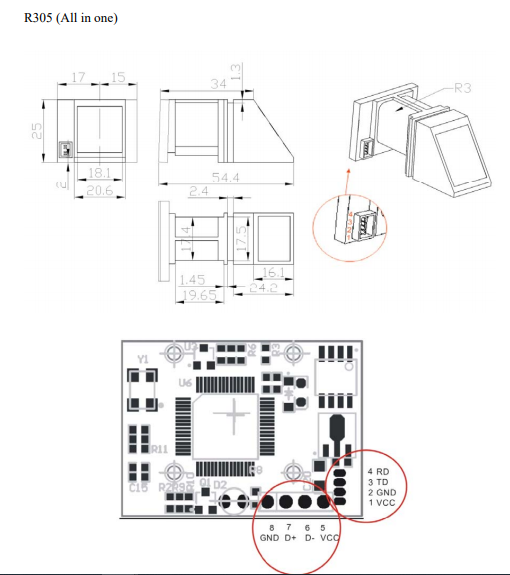
1. **Nhược điểm**

* Chưa từng tiếp xúc với các thiết bị này trong những kì trước nên mất khá nhiều thời gian để nghiên cứu.

1. **Hướng phát triển**

* 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 của các module với mạch**
  2.  **Module đọc vân tay với Arduino**

|  |  |
| --- | --- |
| Aruino Uno | Module Fingerprint |
| 12 | TXD |
| 13 | RXD |
| 5V | VCC |
| GND | GND |

* 1. **LCD 16x02**

Với LCD ta có mạch chuyển đổi còn 4 chân.

|  |  |
| --- | --- |
| Aruino Uno | Mạch chuyển đổi |
| A5 | SCL |
| A4 | SDA |
| 5V | VCC |
| GND | GND |

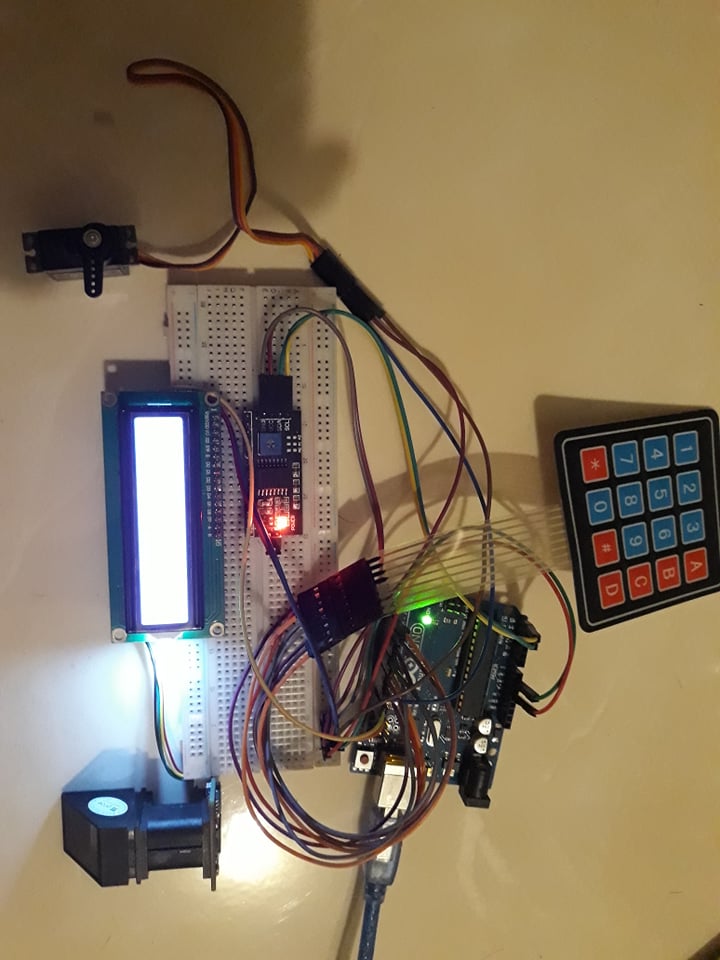
* 1. **Keypad 4x4**

Lần lượt từ chân 3 đến chân 9.

* 1. **Động cơ Servo**

|  |  |
| --- | --- |
| STM32F103C8T6 Kit | Module Fingerprint |
| 9 | Control Pin |
| 5V | VCC |
| GND | GND |

# **Mô hình thực tế**



# **Giải thích thư viện và hàm con**

|  |  |  |
| --- | --- | --- |
| **Thư viện** | **Hàm** | **Tham số, chức năng** |
| Adafruit\_Fingerprint.h |  | Thư viện của hãng cung cấp để khởi tạo cũng như thực thi các chức năng của vân tay( thêm, xóa, xác thực vân tay ) |
| Keypad.h |  | Thư viện bao để sử dụng keypad |
| LiquidCrystal\_I2C.h |  | Thư viện cho màn hình( 2 hàng 2 cột ) |
| Servo.h |  | Thư viện khởi tạo cho motor servo |
|  | void readKeypad() | Hàm bắt sự kiện nhấn phím chính trong chương trình được lặp lại liên tuc trong hàm Loop(), dựa vào từng trường hợp của biến menu điều hướng mà chương trình sẽ thực hiện các lệnh tương ứng. Hàm được bỏ trong hàm lặp vô tận Loop |
|  | int convertCharAndInt(char A[4])  Parameter:  Input: char A[4]  Output: int | Hàm dùng để chuyển Input 1 mảng char gồm 4 phần tử tương ứng với input từ bàn phím của người nhập trong trường hợp cần nhập ID của dấu vân tay muốn xóa thành một số dạng INT |
|  | void startScreen() | Hàm hiển thị ra màn hình khi khởi tạo cũng là màn hình mặc định khi reset chương trình,  Hiển thị ra 2 lựa chọn: 1 là Authenicating( xác thực để mở cửa ) 2 là Admin Method ( truy cập các chức năng mà Admin có quyền ) |
|  | void deleteEvent() | Hàm hiển thị ra màn hình cho người dùng nhập ID của vân tay cần xóa. |
|  | void askForSelectionInAdminMethod() | Hàm hiển thị ra màn hình cho người dùng chọn các các chức năng dưới quyền Admin, gồm 3 lựa chọn Change Password, Add Fingerprint và Delete Fingerprint |
|  | void authenicateAdministrator() | Hàm để gọi xác thực vân tay, đòi hỏi người dùng phải là admin mới được thông qua và cho phép thực hiện các chức năng thuộc quyền Admin. Hàm được gọi từ trong hàm readKeypad được lặp trong hàm lặp loop nên khi sẽ luôn quét để đợi người dùng đến khi nào xác thực xong |
|  | void askForSelectionInAuthenicationMenu() | Hàm hiển thị 2 tùy chọn để xác thực ra màn hình hoặc là xác thực bằng mật khẩu hoặc là xác thực bằng vân tay |
|  | boolean checkCode(char \*a,char \*b)  Parameter:  Input: Char \*a, char \*b  Output: Boolean | Hàm hiển dùng để kiểm tra mật khẩu mà người dùng nhập vào có đúng với mật khẩu cũ không ( trương hợp menu thay đổi mật khẩu của quyền Admin )  Giá trị tham số truyền vào là 2 con trỏ đến vị trí bắt đầu của 2 mảng tương ứng chứa Password và Input từ màn hình của người dùng, giá trị Output sẽ là true hoặc false |
|  | int changeToNewCode(char \*a, char \*b)  Parameter:  Input: Char \*a, char \*b  Output: int | Hàm hiển dùng để thay mật khẩu mới ( trương hợp menu thay đổi mật khẩu của quyền Admin )  Giá trị tham số truyền vào là 2 con trỏ đến vị trí bắt đầu của 2 mảng tương ứng chứa Password và Input từ màn hình của người dùng |
|  | int doorlockCheck() | Hàm dùng để kiểm tra password khi xác thực có đúng hay không để mở cửa ( cho góc của động cơ servo bằng 0 ) |
|  | bool oldCodeCheck() | Hàm hiển thị ra màn hình khi kiểm tra mật khẩu cũ nếu nhập đúng thì yêu cầu nhập password mới |
|  | void reset() | Reset tất cả về mặt định từ mảng chứa input cho đến biến menu điều hướng chương trình, và quay về màn hình mặc định. |
|  | void fingerCheck() | Hàm dùng để xác định xem đã có tồn tại cảm biến vân tay trong thiết bị chưa |
|  | int getFingerprintIDez() | Hàm chính dùng để xác minh vân tay, nếu đang ở menu cho việc xác thực và mở cửa thì hàm sẽ có thêm công dụng hiển thị thông tin ID của người đăng nhập thành công, và motor servo sẽ chỉnh góc quay về 0 để mở cửa |
|  | uint8\_t deleteFingerprint(uint8\_t id) | Hàm dùng để xóa 1 dấu vân tay với 1 giá trị ID là tham số đầu vào |
|  | uint8\_t getFingerprintEnroll() | Hàm đăng ký mới một dấu vân tay |