



ARDUINO

CHO NGƯỜI MỚI BẮT ĐẦU

Arduino

cho người mới bắt đầu

IoT Maker Viet Nam

Mục lục

Lời mở đầu	1
Đôi lời về tác giả	1
Thuật ngữ hay sử dụng	1
Giải thích code trong bài	1
Giới thiệu nội dung	3
Ai có thể sử dụng?	3
Mục tiêu mang lại cho người đọc	4
Chuẩn bị	4
Kiến thức cơ bản	5
Arduino	6
Arduino là gì ?	6
Những board mạch Arduino trên thị trường	9
Giới thiệu board IoT Maker UnoX và IoT Arduino STEM Kit	13
Arduino IDE	18
Khái niệm	18
Cài đặt	18
Ứng dụng mang lại	29
Arduino và C/C++	32
Tổng kết	34
Hello World	35
Giới thiệu một số khái niệm và linh kiện điện tử cơ bản	36
Điện áp, dòng điện và điện trở	36
Tụ điện	39
Cuộn cảm	40
Breadboard	41
Chớp tắt bóng LED	43
Kiến thức	43
Đầu nối	43
Mã nguồn chớp tắt dùng Delay	45
Mã nguồn chớp tắt dùng định thời	45
Kiến thức	46
Analog và Digital	54
PWM	55
Fade LED	56
Điều khiển LED RGB	57
Giới thiệu module cảm biến ánh sáng	61
Điều khiển LED RGB theo cường độ ánh sáng của môi trường	63
Tổng kết	67
Truyền thông nối tiếp	68
Giao tiếp Serial	70

Những khái niệm cơ bản.....	70
Sử dụng chuẩn giao tiếp Serial với board IoT Maker UnoX	72
Ứng dụng	73
Tổng kết	86
Giao tiếp I2C	87
Mô hình Master/slave	88
Giao tiếp I2C	89
Giới thiệu	89
Hoạt động	89
Truyền nhận bit trong I2C	89
Sử dụng giao thức I2C.....	90
Viết chương trình cho I2C	91
Xác định địa chỉ của thiết bị trong giao tiếp I2C	91
Giới thiệu về LCD và OLED	95
Giao tiếp giữa 2 board IoT Maker UnoX	100
Tổng kết	104
Chuẩn giao tiếp truyền nhận dữ liệu SPI	105
Giao thức SPI.....	106
Giới thiệu	106
SPI, ưu và nhược điểm	107
Nguyên lý hoạt động.....	108
SPI, các ví dụ mẫu	109
Hiển thị chữ trên LED matrix	109
Đọc dữ liệu từ cảm biến BMP280, hiển thị trên OLED.....	112
Tổng kết	117
Chuẩn giao tiếp 1-Wire	118
1-Wire	119
1-Wire là gì?	119
1-Wire hoạt động như thế nào?	120
Tiến trình hoạt động (Workflow)	121
Ví dụ chuẩn giao tiếp 1-Wire	122
Một master và một slave	122
Một master và nhiều slave.....	127
Tổng kết	130
Timer - Interrupt	131
Interrupt	132
Giới thiệu	132
Ví dụ	134
Timer/Counter	138
Giới thiệu	138
Thư viện TimerOne	139
Một số ví dụ	140
Summary.....	151
Một số dự án tham khảo	152

Điều khiển xe tự động bằng module Bluetooth	153
Cơ bản về ứng dụng điều khiển xe tự động	153
Mở đầu về điều khiển động cơ DC	154
Xe điều khiển từ xa với 4 động cơ DC	160
Điều khiển xe từ xa bằng Bluetooth	162
Giám sát nhiệt độ, độ ẩm và bật tắt thiết bị thông qua WiFi	172
Cheatsheet	178
Arduino Cheatsheet	179
C - Cheatsheet	182
Lời kết	185
Các thành viên tham gia đóng góp	186
Lời kết	187
Giấy phép sử dụng tài liệu	188

Lời mở đầu

Trong thời đại công nghệ phát triển như vũ bão hiện nay, ở mảng điện tử lập trình, việc dùng Arduino trở nên rất phổ biến. Chúng ta có thể gặp các ứng dụng của Arduino trong rất nhiều lĩnh vực đời sống.

Nhận thấy nhu cầu lớn về việc tìm hiểu Arduino nên chúng tôi biên soạn cuốn sách này nhằm mục đích giúp cho những người không chuyên, những người mới bắt đầu với lập trình vi điều khiển rút ngắn thời gian tìm hiểu ban đầu, tạo ra những dự án với Arduino một cách nhanh chóng và hữu ích.

Mặc dù đã cố gắng tạo ra sản phẩm tốt nhất đến tay người dùng, tuy nhiên trong quá trình biên soạn sách không khỏi có những thiếu sót, chúng tôi rất hoan nghênh nếu nhận được những phản hồi chỉnh sửa hoặc đóng góp ý kiến để chất lượng về nội dung sách được tốt hơn. Sách được public trên github tại đường dẫn : github.com/iotmakervn/arduino-for-beginners

Đôi lời về tác giả

Chủ biên của cuốn sách là ông [Phạm Minh Tuấn](#)(TuanPM), người có nhiều năm kinh nghiệm làm việc trong lập trình vi điều khiển, IoT và phát triển các thư viện mã nguồn mở cho cộng đồng. Tác giả xây dựng cuốn sách này với mục đích đóng góp 1 phần nhỏ những kiến thức của mình để cho những người mới bắt đầu tiếp cận với lập trình vi điều khiển thông qua nền tảng phát triển Arduino.

Thuật ngữ hay sử dụng

- [Arduino](#) - Đèn cập đến các board mạch Arduino trên thị trường.
- [Arduino IDE](#) - Viết tắt của Arduino Integrated Development Environment. Nói 1 cách đơn giản, đó là công cụ để lập trình với các board Arduino bao gồm trình soạn thảo code, gỡ lỗi và nạp chương trình cho board.
- [Git](#) - Trình quản lý phiên bản.
- [Github](#) - Mạng xã hội dành cho lập trình viên.
- [Compiler](#) - Trình biên dịch.
- [Logic Level](#) - Mức điện áp để chip hiểu được (1 hay 0).

Giải thích code trong bài

```
void setup()
{
    //comment ①
    int a = 1;
    a++; ②
}
```

① Dòng này giải thích đây là comment (chú thích).

② Dòng này giải thích biến `a` tăng thêm 1 đơn vị.

Giới thiệu nội dung

- Nội dung quyển sách này bao gồm các hướng dẫn chi tiết cho người đọc về cách sử dụng và những tính năng của nền tảng phát triển Arduino thông qua board mạch IoT Maker UnoX, hiểu được các chức năng và chuẩn giao tiếp thông dụng trong truyền nhận dữ liệu, đồng thời bám sát nội dung đã học bằng các bài thực hành.
- Phần cứng sử dụng là board mạch phần cứng mở IoT Maker UnoX do IoT Maker VietNam thiết kế, hoàn toàn tương thích với chuẩn Arduino Uno trên thị trường. Đây là 1 dự án open source hardware nên chúng tôi rất hoan nghênh nếu có những nhận xét hoặc đóng góp nhằm cải thiện các tính năng cho board mạch.
- Phần mềm sử dụng lập trình trên máy tính là [Arduino](#), ngôn ngữ lập trình [C/C++](#).

Ngoài ra, bạn sẽ cần tìm hiểu một số công cụ và khái niệm thường xuyên được sử dụng trong quyển sách này như sau:

- **Git** - Trình quản lý phiên bản sử dụng rất rộng rãi trên thế giới. Git giúp bạn quản lý được mã nguồn, làm việc nhóm, xử lý các thao tác hợp nhất, lịch sử mã nguồn, ... Có thể trong quá trình làm việc với quyển sách này, bạn sẽ cần sử dụng các thư viện mã nguồn mở cho Arduino từ Github, nên việc cài đặt và sử dụng công cụ khá cần thiết cho việc đó. Chưa kể, nó sẽ giúp bạn quản lý mã nguồn và dự án ngày càng chuyên nghiệp hơn.
- **Github** - Là một mạng xã hội cho lập trình viên dựa trên Git.
- **Firmware** - là 1 phần mềm (software) được nhúng (embedded) vào phần cứng (hardware) của thiết bị, giúp điều khiển, cập nhật các tính năng mới cho phần cứng.



Tuy phần cứng chính thức sử dụng là board mạch phần cứng mở IoT Maker UnoX, nhưng bạn hoàn toàn có thể sử dụng bất kỳ board Arduino Uno nào khác trên thị trường cho cuốn sách này, ví dụ như: [Arduino Uno R3](#), [Arduino-nano-3.0](#), ...



Tất cả các mã nguồn đều hạn chế giải thích rõ chi tiết API cho mỗi tính năng, thay vào đó được cung cấp tại phụ lục Cheat Sheet (Arduino).

Ai có thể sử dụng?

- Học sinh (cấp 2, 3), sinh viên muốn bổ sung kiến thức, nâng cao kỹ năng.
- Các giáo viên THCS, THPH muốn sử dụng Arduino trong giảng dạy STEM
- Các lập trình viên phần mềm muốn tham gia làm sản phẩm điện tử.
- Các kỹ sư không chuyên lập trình muốn làm các sản phẩm tự động

- Cá nhân muốn tự mình làm các sản phẩm phục vụ công việc và cuộc sống.
- Các công ty không chuyên về phần cứng hoặc phần mềm.
- Và tất cả các lập trình viên yêu thích về phần cứng.

Mục tiêu mang lại cho người đọc

- Hiểu được cách thức hoạt động của nền tảng phát triển Arduino.
- Hiểu được cách hoạt động, các chuẩn giao tiếp trong truyền nhận dữ liệu của 1 vi điều khiển.
- Giúp cho người không chuyên về phần cứng tiếp cận để làm sản phẩm 1 cách dễ dàng.
- Có thể tự phát triển hệ thống tích hợp cho sản phẩm.

Chuẩn bị

- Bạn cần ít nhất 1 board mạch Arduino lập trình được, tốt nhất nên sử dụng các board mạch đã có các module nạp cho chip.
- Nên có thêm các module khác như cảm biến, động cơ để thực hành, một bộ [Arduino Kit](#) là phù hợp.
- 1 máy tính cá nhân (Windows, MacOS hoặc Linux).
- C & Arduino CheatSheet (Mục lục cuối quyển sách này).

Kiến thức cơ bản

Để có cái nhìn tổng quan khi bắt đầu với nền tảng phát triển Arduino, chúng ta cùng điểm qua những nội dung sẽ tìm hiểu phần này như sau:

- Khái niệm về Arduino.
- Tìm hiểu các dòng chip và các board Arduino trên thị trường.
- Giới thiệu board mạch **IoT Maker UnoX**.
- Tìm hiểu về Arduino IDE và cách cài đặt.
- Giới thiệu sơ lược về ngôn ngữ C/C++.

Arduino

Arduino là gì ?

Khái niệm, lịch sử hình thành và phát triển.

Theo định nghĩa từ www.arduino.cc, Arduino là nền tảng điện tử mã nguồn mở, dựa trên phần cứng và phần mềm, linh hoạt và dễ sử dụng, các board Arduino có khả năng đọc dữ liệu từ môi trường (ánh sáng, nhiệt độ, độ ẩm,...), trạng thái nút nhấn, tin nhắn từ Twitter,... và điều khiển trở lại với các thiết bị như động cơ, đèn LED, gửi thông tin đến 1 nơi khác,... Chúng ta có thể điều khiển các vi điều khiển trên board Arduino bằng cách sử dụng ngôn ngữ lập trình C++, được điều khiển biên dịch bởi **Arduino IDE** và các trình biên dịch đi kèm ra mã máy nhị phân. Lúc này Vi điều khiển có thể dễ dàng thực thi chương trình.

Hiện tại, Arduino là một công ty hoạt động trong lĩnh vực phần cứng và phần mềm máy tính mã nguồn mở. Dự án Arduino được sinh ra tại học viện Interaction Design ở Ivrea, Italy vào năm 2003. Mục đích ban đầu của board Arduino là giúp cho các sinh viên ở học viện - những người không có nền tảng kiến thức về điện tử có thể tạo ra các sản phẩm 1 cách nhanh chóng với chi phí thấp và dễ sử dụng. Đó là 1 dự án mã nguồn mở, Arduino phát triển thông qua việc cho phép người dùng trên toàn thế giới có thể xây dựng, phát triển và đóng góp vào dự án.



Tên Arduino là tên của 1 quán bar ở Ivrea, Italy. Đây là nơi những nhà sáng lập ra dự án arduino gặp mặt để bắt đầu ý tưởng hình thành dự án này. Tên của quán bar này đặt theo tên của người chỉ huy quân đội (như lãnh chúa thời phong kiến) tại Ivrea và sau đó ông này là vua của nước Italy từ năm 1002 đến năm 1014.

Tại sao là Arduino ?

Hiện nay, Arduino được sử dụng trong rất nhiều dự án và trong nhiều lĩnh vực khác nhau. Chính vì sự đơn giản, dễ sử dụng và đặc biệt là mã nguồn và phần cứng mở nên nó nhận được sự hỗ trợ rất lớn từ các lập trình viên trên toàn thế giới. Phần mềm rất dễ cho những người mới bắt đầu nhưng cũng không thiếu sự linh hoạt cho những lập trình viên lâu năm. Cộng đồng Arduino rất lớn nên khi sử dụng với Arduino, theo cách nói vui là bạn được "support tới tận răng", có nghĩa là vấn đề bạn gặp phải bây giờ, người dùng Arduino trên thế giới cũng đã gặp phải, giải quyết nó và đưa ra các thư viện tốt nhất cho bạn, với cộng đồng lớn và đặc biệt là tất cả đều open-source nên sẽ dễ dàng hơn nếu bạn chọn Arduino thay vì nền tảng lập trình khác.

Với những người "ngoại đạo" (đề cập đến những người không có nhiều kiến thức về điện tử) thì Arduino quả thực rất tuyệt vời, nó giúp họ dễ dàng tạo ra những sản phẩm liên quan đến điện tử. Những kiến trúc sư, giáo viên, nghệ sĩ có thể chỉ mất vài ngày để tạo ra sản phẩm điện tử nhằm phục

vụ cho nhu cầu của họ, trong khi việc này trước đây dường như là bất khả thi. Tuy nhiên, có một câu hỏi đặt ra là: **Nếu là 1 kỹ sư điện tử lập trình, có nên dùng Arduino cho các dự án của mình hay không?** Bởi nó quá đơn giản và nhìn giống như là "đồ chơi của trẻ con" ???

Đây là chủ đề được bàn luận khá sôi nổi và thật khó để tìm ra câu trả lời chính xác. Arduino che đi sự phức tạp của việc lập trình cho vi điều khiển bằng cách phủ lên mình lớp "vỏ bọc" bên trên. Chỉ 1 vài câu lệnh đơn giản là có thể chớp, tắt được 1 con LED trong khi với các nền tảng lập trình khác, muốn làm được chuyện này thì bạn phải hiểu kiến trúc của vi điều khiển, hiểu cách truy cập, setup giá trị các bit trong thanh ghi,... từ đó mới có thể dùng tập lệnh của nó để viết code điều khiển LED. Và sự phức tạp, tinh vi của nền tảng Arduino cũng không thua kém gì các thư viện của nhà sản xuất, có chăng nó làm cho người dùng cảm giác đơn giản hơn thôi.

Những người xây dựng nền tảng Arduino đã tạo ra những lệnh vô cùng đơn giản, giúp cho người dùng dễ tiếp cận. Tuy nhiên cách học đối với "**những người trong nghề**" không gì tốt hơn nếu muốn nắm rõ về lập trình vi điều khiển là **đào sâu tìm hiểu**. Ví dụ, đối với 1 người đang làm việc ở lĩnh vực IT, muốn tạo hiệu ứng cho các bóng đèn LED qua 1 ứng dụng trên điện thoại. Với họ, việc tạo ứng dụng trên điện thoại là không thành vấn đề, liên quan đến điều khiển LED, chỉ cần kết nối board Arduino với LED, "google search" để tìm kiếm 1 thư viện phù hợp, lấy những hiệu ứng họ cần trong thư viện đó. Việc này khá đơn giản nếu dùng Arduino. Vấn đề của họ đã được giải quyết thành công mà không cần biết quá nhiều về kiến thức điện tử.

Tuy nhiên, vấn đề phát sinh ở đây là: tôi muốn LED sáng mờ hơn, tôi muốn tạo 1 số hiệu ứng theo ý mình, tôi cần kết nối nó với các cảm biến khác, tôi cần truyền, nhận dữ liệu giữa các module mà tôi đã kết hợp,... bài toán đặt ra đã trở nên thực sự phức tạp và vượt tầm hiểu biết của họ. Lúc này, giá trị của 1 kỹ sư điện tử lập trình sẽ được thể hiện. Để làm được những yêu cầu ấy, bạn phải hiểu rõ cách thức hoạt động của vi điều khiển, các chức năng, các chuẩn truyền dữ liệu,... để từ đó có thể hiểu chỉnh lại thư viện đang có, tối ưu hóa và tùy biến theo yêu cầu của người dùng.

Câu hỏi đặt ra tiếp theo ở đây là: **rất nhiều nền tảng lập trình khác cũng làm được điều này, vậy đâu là những lợi ích của Arduino ?**

Ích lợi chính ở đây đó là sự đơn giản của các tập lệnh, cộng đồng lớn và open-source. Thư viện dành cho Arduino không quá khó tìm, lệnh không quá nhiều để học nhưng cái chính là bạn phải hiểu nó hoạt động như thế nào. Để tìm hiểu sâu hơn về nó, những người phát triển Arduino đã cung cấp cho chúng ta 1 thư viện **Hardware Abstraction Library** (gọi tắt là HAL) dành cho những ai muốn tìm hiểu sâu hơn về cách mà Arduino hoạt động.

Ví dụ, với Arduino, để bật 1 bóng LED, chúng ta sẽ dùng 2 lệnh đơn giản là `pinMode(PIN_LED)` và lệnh `digitalWrite(PIN_LED)` với `PIN_LED` là định nghĩa chân được đấu nối với đèn LED ngoài thực tế. Tìm hiểu chi tiết hơn thì `pinMode()` là 1 hàm :

- Nhằm cấu hình chân giống như ngõ vào (input) hoặc ngõ ra (output), nó sẽ cấu hình thanh ghi hướng dữ liệu DDR (Data Direction Register), nếu 1 bit của thanh ghi DDR là 0 thì chân đó sẽ được cấu hình là input, giá trị bit bằng 1 là output.

- Giá trị mặc định ban đầu của các bit này là 0 (input).
- DDR là tên gọi chung của các thanh ghi ở chip ATmega328P, nó bao gồm 3 thanh ghi DDRB, DDRC, DDRD. Các thanh ghi này liên quan đến các chân của chip ATmega328P (board Arduino Uno R3) như bảng bên dưới:

DDRB								
bit	7	6	5	4	3	2	1	0
pin	-	-	D13	D12	D11	D10	D9	D8
DDRC								
bit	7	6	5	4	3	2	1	0
pin	-	-	A5	A4	A3	A2	A1	A0
DDRD								
bit	7	6	5	4	3	2	1	0
pin	D7	D6	D5	D4	D3	D2	D1	D0

Hình 1. Bảng các thanh ghi DDR tương ứng với các chân của board Arduino Uno R3

Tiếp theo, ATmega328P có 3 thanh ghi PORT để cài đặt giá trị cho các bit, các bit này tương ứng với các chân I/O của chip ATmega328P. Chân A0 - A5 là các chân Analog, các chân D0 - D13 là các chân Digital. Giá trị bit bằng 0 là LOW (mức thấp, điện áp 0V DC), bằng 1 là HIGH (mức cao, điện áp 5VDC). Giá trị mặc định ban đầu của bit là 0.

PORTB								
bit	7	6	5	4	3	2	1	0
pin	-	-	D13	D12	D11	D10	D9	D8
PORTC								
bit	7	6	5	4	3	2	1	0
pin	-	-	A5	A4	A3	A2	A1	A0
PORTD								
bit	7	6	5	4	3	2	1	0
pin	D7	D6	D5	D4	D3	D2	D1	D0

Hình 2. Bảng các thanh ghi PORT tương ứng với các chân của board Arduino Uno R3

Như vậy, nếu chúng ta muốn cho các chân từ D0 - D7 là ngõ ra thì ta cần cài đặt DDRD = 0b00000000 (0b là định dạng kiểu nhị phân, được hiểu là gán giá trị 0 hoặc 1 cho từng bit), cho các chân này mức HIGH thì cài đặt giá trị thanh ghi PORTD = 0b11111111.

Đó là cách hoạt động chung khi cài đặt hướng và setup giá trị cho các chân GPIO của các nền tảng lập trình hiện nay. Nếu bạn là 1 kĩ sư điện tử lập trình thì cách học nên theo hướng như vậy. Một khi bạn đã hiểu cách hoạt động của vi điều khiển thì sử dụng Arduino sẽ giúp chúng ta xây dựng dự án 1 cách nhanh chóng do tập lệnh khá đơn giản để dùng.



Chúng ta có thể xem các tập lệnh của Arduino tại www.arduino.cc/reference/en và tìm hiểu sâu hơn các tập lệnh của Arduino tại link Arduino hardware core



Một số hàm cơ bản được giải thích chi tiết hơn tại garretlab.web.fc2.com

Bên cạnh đó, một số điểm mạnh của Arduino nữa là:

- Các ví dụ mẫu đi kèm với thư viện và tất cả đều open-source nên khi tìm đến các ví dụ mẫu, chúng ta sẽ hiểu cách thức thư viện hoạt động. đồng thời có thể xem source code của họ viết

để có thể hiệu chỉnh, tối ưu thư viện theo cách của mình.

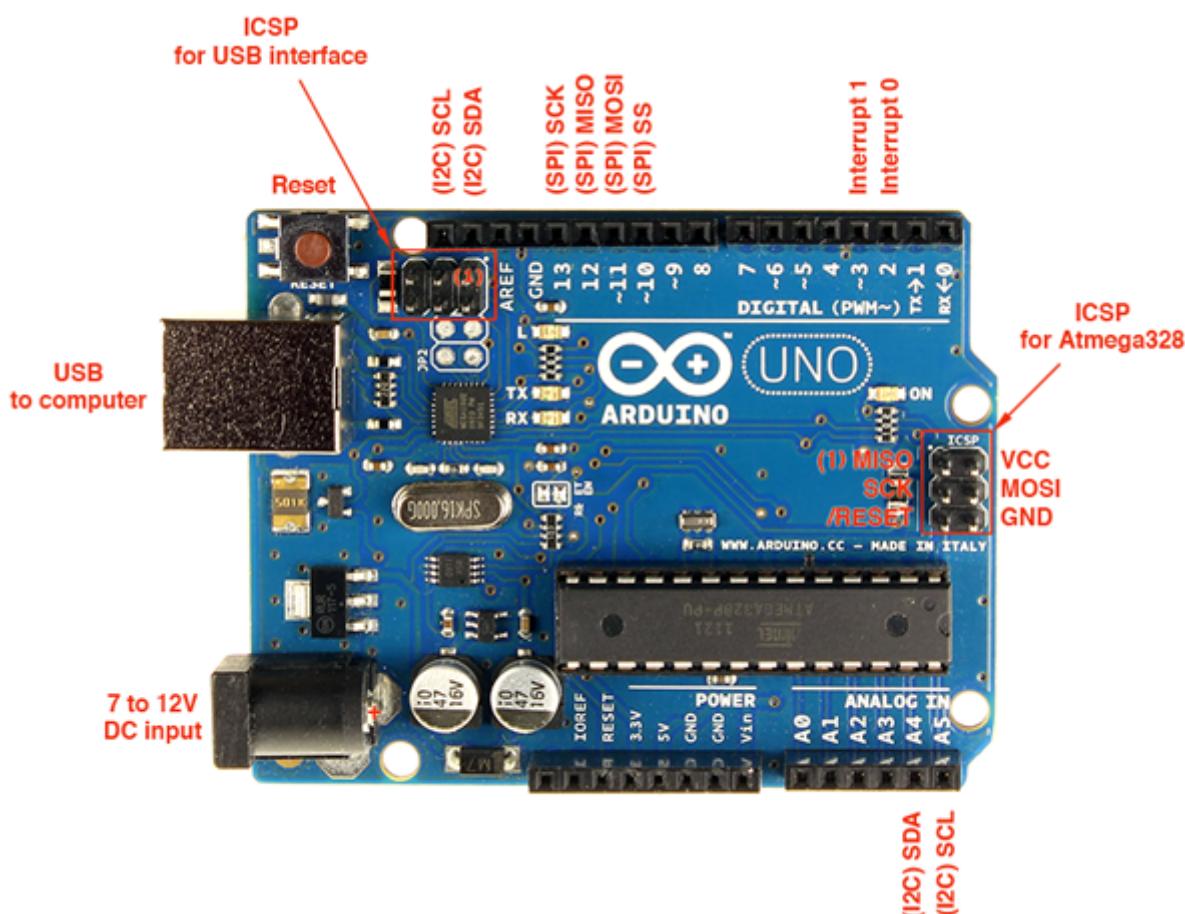
- Việc upload code thông qua cổng USB, giúp đơn giản quá trình nạp code.

Những board mạch Arduino trên thị trường

Hiện nay trên thị trường có hàng trăm board mạch Arduino khác nhau, Chúng đa phần là các biến thể PCB (các board mạch điện) của những board mạch chính đến từ nhà sản xuất Arduino. Những board mạch này hoặc có thêm 1 số tính năng cải tiến nào đó hoặc đơn giản là được thiết kế lại nhằm giảm giá thành sản phẩm để có thể tới tay người dùng nhiều hơn. Chúng ta hãy cùng điểm qua 1 số board mạch Arduino chính như bên dưới :

1. Arduino Uno R3

Đây là board mạch được đánh giá là tốt nhất cho những người mới bắt đầu về điện tử và lập trình. Nó được sử dụng nhiều nhất trong các board mạch thuộc họ Arduino. Hình ảnh và các chức năng của board Arduino Uno R3 :



Hình 3. Hình ảnh và các chức năng của board Arduino Uno R3 (Nguồn www.arduino.cc)

Điểm qua 1 số thông tin chính của board:

- Giá thành : €20.00 (theo www.arduino.cc).

- Sử dụng vi điều khiển ATmega328 của hãng Atmel.
- Lập trình thông qua giao diện cổng USB.
- Header cho các chân GPIO.
- Gồm 4 LED: nguồn, RX, TX và Debug.
- Nút nhấn Reset board mạch.
- Có jack để cắm nguồn khi không dùng nguồn ở cổng USB.
- Các header cho In-circuit serial programmer (ICSP), hiểu đơn giản thì đây là các header để kết nối với mạch nạp cho chip nếu không nạp thông qua cổng USB.

Giới thiệu về vi điều khiển ATmega328

Vi điều khiển (tiếng Anh là microcontroller hoặc microcontroller unit) là trái tim của các board mạch lập trình được. Nó có khả năng thực thi code khi chúng ta yêu cầu. Bên trong vi điều khiển bao gồm rất nhiều các mạch điện với các khối chức năng như CPU (Central Processing Unit), RAM (Random Access Memory), ROM (Read Only Memory), Input/output ports, các bus giao tiếp (I2C, SPI)...

Vi điều khiển giúp chúng ta có thể giao tiếp với sensor, điều khiển thiết bị.

Board Arduino Uno R3 sử dụng vi điều khiển ATmega328 của hãng Atmel (một công ty thiết kế và chế tạo vật liệu bán dẫn ở Mỹ). ATmega328 là vi điều khiển thuộc dòng vi điều khiển 8 bits (data bus là 8 bit)

Bảng 1. Bảng thông số kỹ thuật của ATmega328 (theo wikipedia.org)

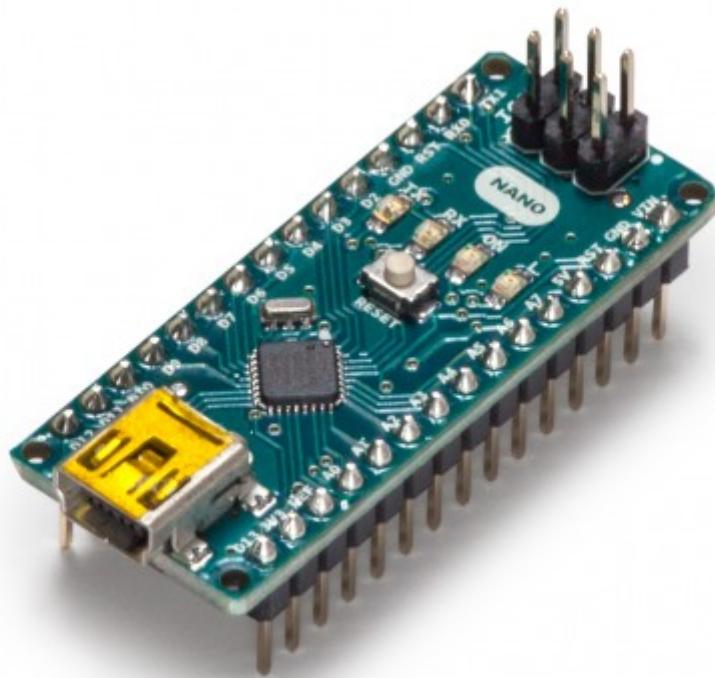
Parameter	Value
CPU type	8-bit AVR
Performance	20 MIPS at 20 MHz
Flash memory	32 kB
SRAM	2 kB
EEPROM	1 kB
Pin count	28-pin PDIP, MLF, 32-pin TQFP, MLF
Maximum operating frequency	20 MHz
Number of touch channels	16
Hardware QTouch Acquisition	No
Maximum I/O pins	26
External interrupts	2
USB Interface	No

Chúng ta sẽ tìm hiểu sâu hơn về ATmega328 ở các chương tiếp theo của sách. Dưới đây là bảng thông số kỹ thuật của board Arduino Uno R3

Bảng 2. Bảng thông số kỹ thuật của ATmega32 (theo www.arduino.cc)

Parameter	Information
Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 [of which 6 provide PWM output]
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length x Width	68.6 mm x 53.4 mm
Weight	25 g

2. Arduino Nano



Hình 4. Hình ảnh board Arduino Nano (Nguồn www.arduino.cc)

Arduino Nano là một board mạch sử dụng chip ATmega328 (loại Arduino Nano 3.x) hoặc dùng ATmega168 (Arduino Nano 2.x), tuy nhiên có kích thước nhỏ gọn hơn để có thể tích hợp vào các hệ thống, đi kèm với đó là 1 vài điểm khác khi so sánh với board Arduino Uno R3 :

- Sử dụng cổng Mini-B USB thay vì cổng USB chuẩn.
- Bổ sung thêm 2 chân Analog.
- Không có jack nguồn DC.

3. Arduino Leonardo



Hình 5. Hình ảnh board Arduino Leonardo (Nguồn www.arduino.cc)

Arduino Leonardo sử dụng vi điều khiển ATmega32u4, một số điểm khác biệt chính so với board Arduino Uno được liệt kê bên dưới:

- Bên trong chip ATmega32u4 được tích hợp 1 chip usb to serial thay vì phải dùng 2 mcu trên board.
- Có thể giả lập board Leonardo như chuột, bàn phím, joystick thay vì phải dùng 1 thiết bị serial khác. Chúng ta sẽ tìm hiểu tính năng này ở phần USB-serial.
- Giá thành rẻ hơn (€18.00 trên www.arduino.cc)
- 20 digital I/O (7 chân PWM).
- 12 chân Analog (các chân PWM có thể được dùng như Analog)

4. Arduino mega2560



Hình 6. Hình ảnh board Arduino Mega 2560 (Nguồn www.arduino.cc)

Arduino mega2560 sử dụng chip ATmega2560 với 54 chân digital I/O (15 chân có thể dùng với PWM), 16 chân Analog, 4 UARTs,... board mạch này là phiên bản nâng cao của Arduino Uno, được dùng trong các dự án phức tạp như máy in 3D, robot,...

Giới thiệu board IoT Maker UnoX và IoT Arduino STEM Kit

Board IoT Maker UnoX

Giới thiệu

Board [IoT Maker UnoX](#) được thiết kế và sản xuất bởi [IoT Maker VietNam](#). Về cơ bản, nó tương tự như board mạch Arduino Uno R3 với 1 số tính năng bổ sung :

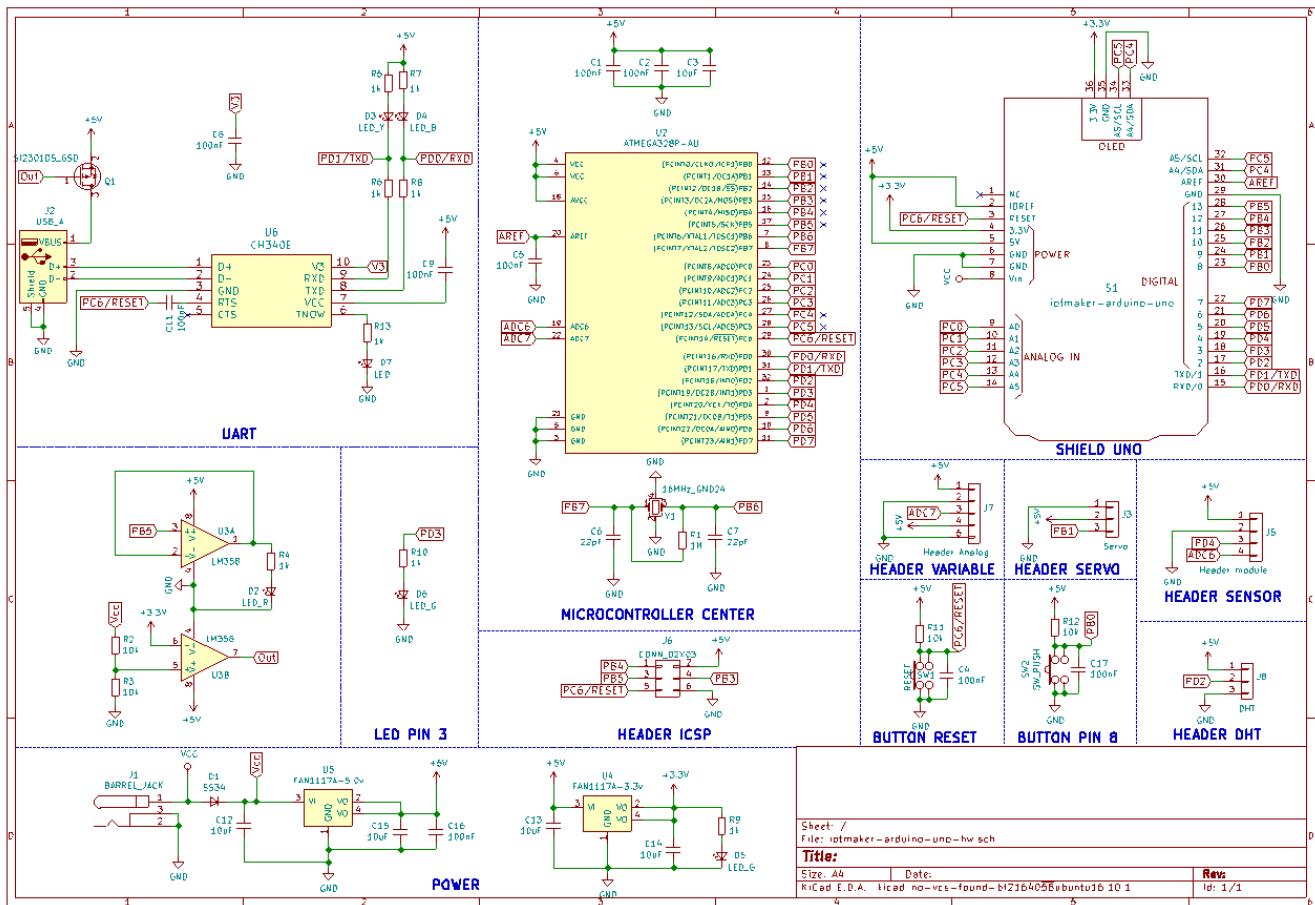
- Giá thành thấp, độ ổn định cao.
- Sử dụng chip nạp CH340E có kích thước nhỏ gọn và tốc độ upload cực nhanh.
- Được trang bị các header giúp việc giao tiếp với các module liên quan đến truyền nhận dữ liệu theo chuẩn I2C một cách dễ dàng.
- Sử dụng cổng micro USB thay cho cổng USB truyền thống.
- Sử dụng chip SMD (chip dán) nên có thêm 2 chân Analog so với board Arduino Uno.

- Bổ sung thêm nút nhấn kết nối với chân D2 để lập trình.

Thông tin

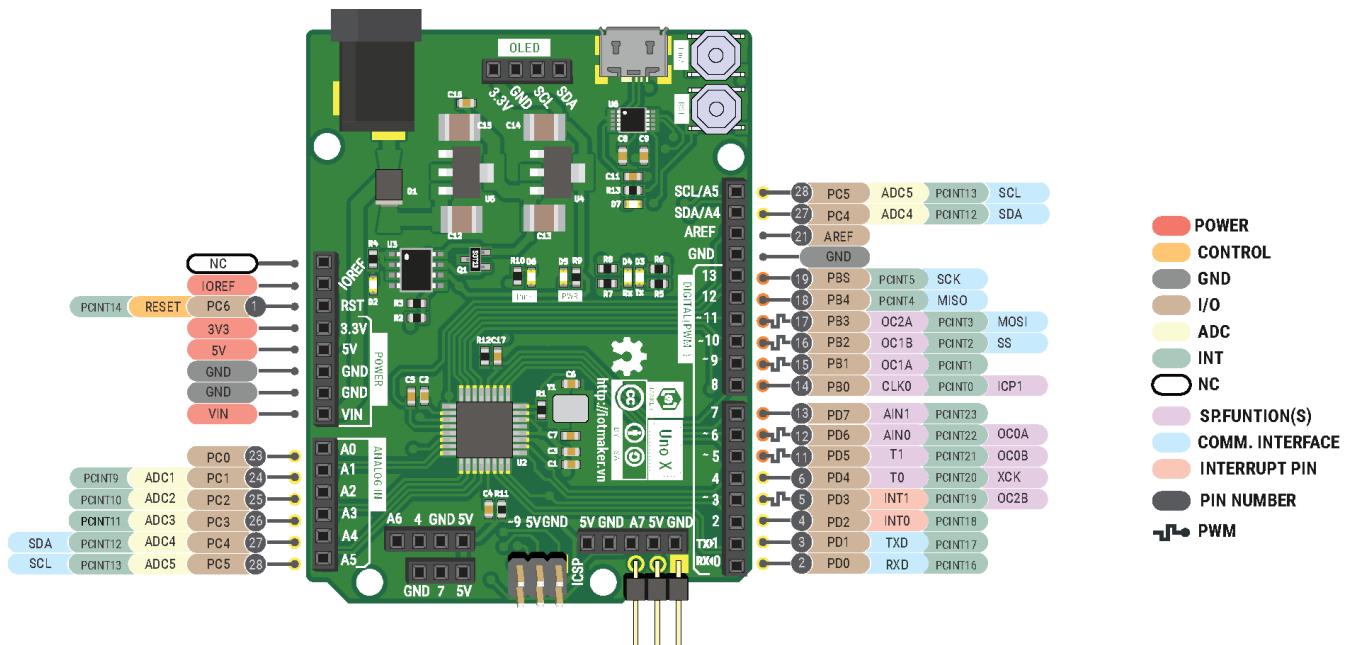
Đây là dự án Open-source hardware, các bạn có thể xem đầy đủ nội dung của dự án tại đường dẫn github.com/iotmakervn/iotmaker-arduino-uno-hw

Hình ảnh schematic



Hình 7. Hình ảnh sơ đồ nguyên lý board IoT Maker UnoX.

Pinout

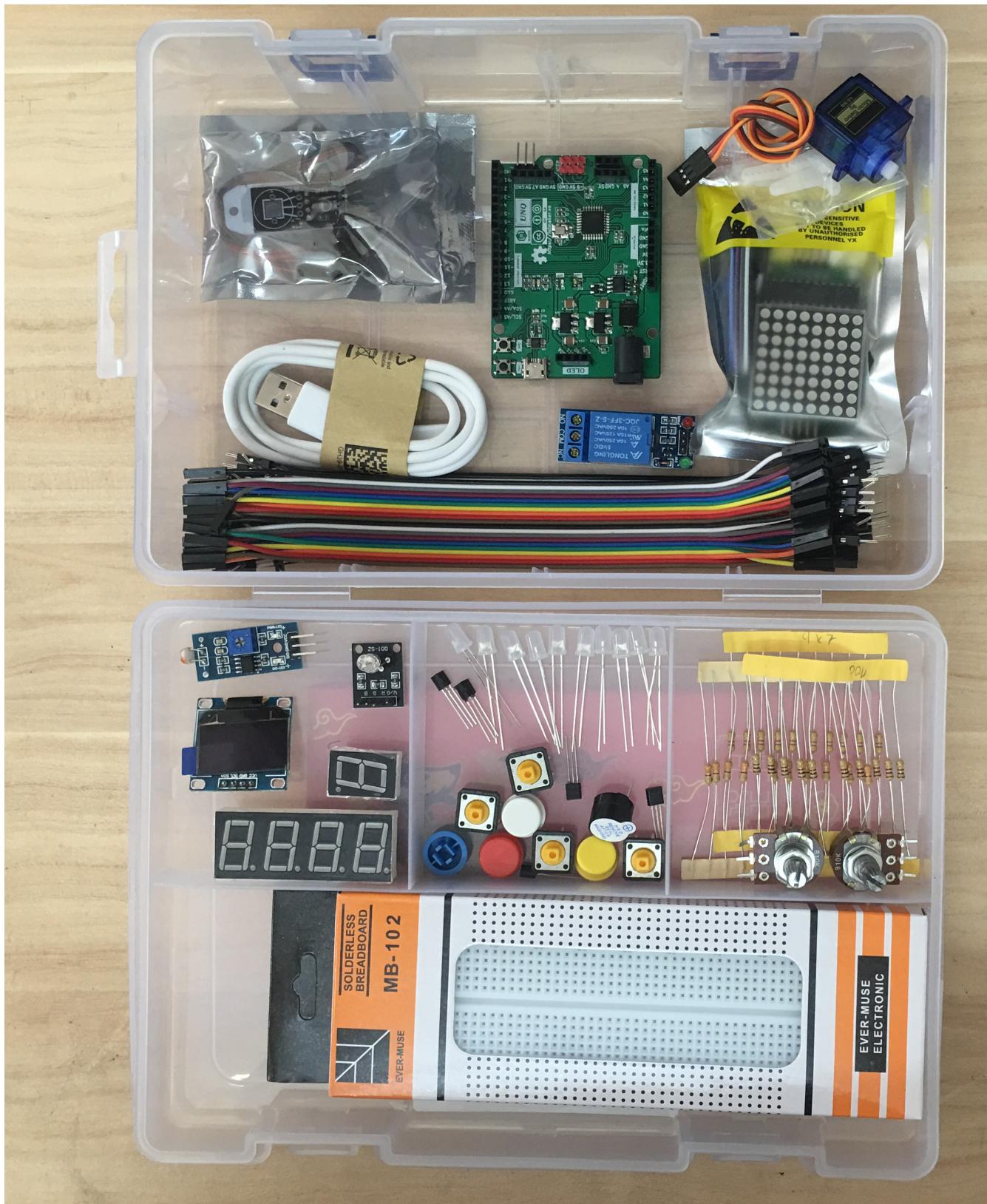


Hình 8. Hình ảnh pinout board IoT Maker UnoX.

Arduino STEM Kit

Nhằm mục đích cho việc thực hành các nội dung trong cuốn sách này với board IoT Maker UnoX, [IoT Maker VietNam](#) thiết kế bộ [Arduino STEM Kit](#) để người dùng có đầy đủ công cụ thực hành và lập trình các nội dung trong sách. Nó cũng giúp chúng ta giảm chi phí và thời gian để tìm mua các linh kiện.

Hình ảnh



Hình 9. Hình ảnh các linh kiện của IoT Arduino STEM Kit.

Thông tin sản phẩm

Bảng 3. Bảng danh sách chi tiết về các linh kiện trong bộ IoT Arduino STEM Kit

STT	Tên module/linh kiện	Số lượng(pcs)
1	Board IoT Maker UnoX	1

STT	Tên module/linh kiện	Số lượng(pcs)
2	Màn hình OLED	1
3	Breadboard	1
4	Cảm biến DHT11	1
5	IC cảm biến nhiệt độ LM35	1
6	Dây cắm female-female	20
7	Dây cắm female-male	10
8	Dây cắm male-male	10
9	Cable micro USB	1
10	LED đơn	10
11	Điện trở cảm	50
12	Cảm biến khoảng cách HC-SR04	1
13	Relay 1 kênh 5V	1
14	Module LED RGB	1
15	Module cảm biến ánh sáng	1
16	Động cơ RC Servo 9G SG90	1
17	LED 7 đoạn Anode chung	5
18	Buzzer	1
19	Module LED matrix max7219	1
20	Module Điều khiển từ xa dùng hồng ngoại HX1838	1
21	Module cảm biến độ ẩm đất	1

Arduino IDE

Khái niệm.

Hiểu một cách đơn giản, Arduino IDE là 1 phần mềm giúp chúng ta nạp code đã viết vào board mạch và thực thi ứng dụng. Arduino IDE là chữ viết tắt của Arduino Integrated Development Environment, một công cụ lập trình với các board mạch Arduino. Nó bao gồm các phần chính là Editor (trình soạn thảo văn bản, dùng để viết code), Debugger (công cụ giúp tìm kiếm và sửa lỗi phát sinh khi build chương trình), Compiler hoặc interpreter (công cụ giúp biên dịch code thành ngôn ngữ mà vi điều khiển có thể hiểu và thực thi code theo yêu cầu của người dùng).



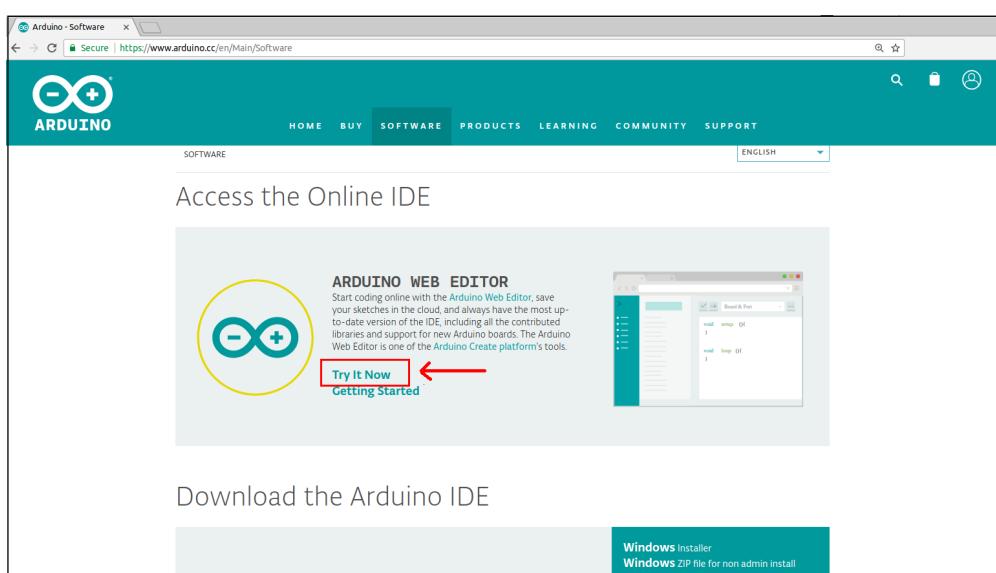
Hiện nay, ngoài các board thuộc họ Arduino, thì Arduino IDE còn hỗ trợ lập trình với nhiều dòng vi điều khiển khác như ESP, ARM, PIC, ...

Cài đặt

Chúng ta đã đề cập đến tính năng và lợi ích mang lại ở mục trước, phần này sẽ hướng dẫn các bạn cách cài đặt Arduino IDE. Có 2 cách sử dụng, bao gồm sử dụng online (nếu có kết nối internet ổn định) và cài đặt offline trên máy. Khuyến cáo nên sử dụng cài đặt offline.

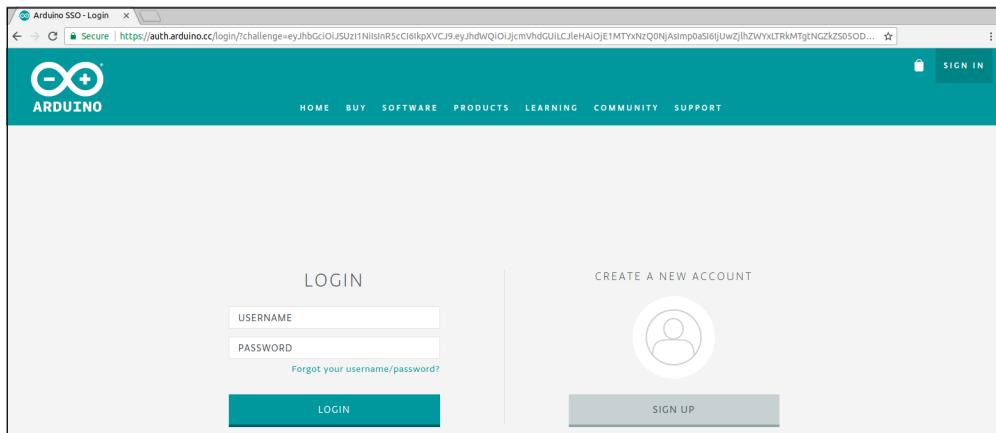
1. Dùng Online IDE

Bước 1: Truy cập vào đường dẫn <https://www.arduino.cc/en/Main/Software>. Chọn **try it now**.



Hình 10. Giao diện Online IDE

Bước 2: Tạo tài khoản bằng cách nhấn vào **Signup** nếu lần đầu sử dụng, hoặc đăng nhập bằng cách nhấn vào **Login** nếu đã tạo tài khoản trước đó. Giao diện như hình bên dưới:



Hình 11. Giao diện đăng ký, đăng nhập.

Bước 3: Cài đặt Arduino plugin

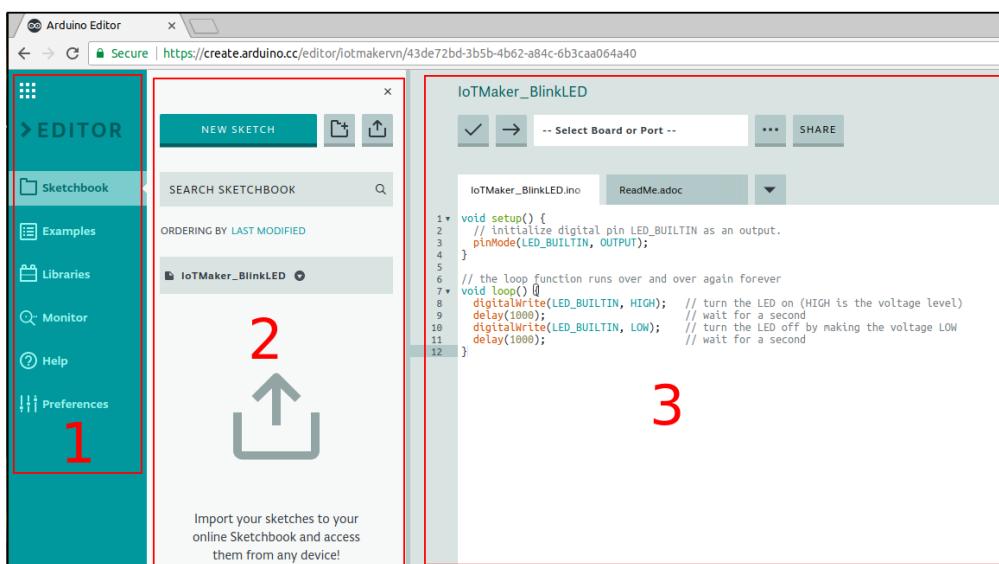
Mục đích của việc cài đặt này là để cho phép trình duyệt Web tải các chương trình của bạn vào board Arduino. Download phần mềm và cài đặt theo các hướng dẫn của phần mềm như hình ảnh bên dưới:



Hình 12. Cài đặt Arduino plugin

Bước 4: Lập trình trên Web Editor.

Truy cập vào Arduino Web Editor như ở bước 2, thực hiện đăng nhập. Giao diện chia làm 3 phần như hình bên dưới:



Hình 13. Giao diện Arduino Web Editor

Mô tả các tính năng ở mục số 1:

- **Your Sketchbook:** Giúp chúng ta có thể thấy các sketch (các chương trình trong Arduino được gọi là Sketch)
- **Examples:** Đây là những sketch tham khảo nhằm giúp người dùng xem các chức năng cơ bản của các thư viện Arduino, các thư viện này mặc định ở chế độ read-only (chỉ đọc và không cho phép chỉnh sửa).
- **Libraries:** Cho phép chúng ta "include" những thư viện vào trong sketch để thực hiện các chức năng theo nhu cầu sử dụng.
- **Serial monitor:** Cho phép truyền nhận dữ liệu của board thông qua USB cable.
- **Help:** Cung cấp các hướng dẫn để bắt đầu lập trình Arduino Web Editor.
- **Preferences:** Những cài đặt về các thuộc tính của trình soạn thảo code đang sử dụng, như cỡ chữ, màu nền,...

Mô tả các tính năng ở mục số 2:

Trong mục này hiển thị các folder, sketch, nó cũng bao gồm các tùy chọn như tạo 1 folder mới, tạo sketch mới, import 1 sketch khác từ máy tính lên Arduino Web Editor.

Mô tả các tính năng ở mục số 3:

Phần này bao gồm trình soạn thảo code và các option để có thể nạp code vào board, bao gồm:

- Nút **Verify:** Giúp biên dịch các file của chương trình, sẽ có thông báo lỗi nếu phát sinh lỗi trong code.
- Nút **Upload:** Upload code vào board Arduino, quá trình này bao gồm cả biên dịch các file trong sketch.

- Hộp thoại **Select board and port**: Cần chọn board và port để nạp code.
- Nút ...: Có các tùy chọn liên quan đến sketch như lưu, xóa, đổi tên, ...
- Nút **Share**: Cho phép chúng ta chia sẻ sketch tới người khác.

Việc tạo 1 dự án trên Arduino Web Editor là khá đơn giản, chỉ cần tạo 1 sketch mới, viết code trên vùng soạn thảo, chọn đúng board, port và nạp chương trình. Chi tiết các bước thực hiện có thể xem tại [getting-started-with-arduino-web-editor](#)

2. Dùng Offline

Cài đặt

Bước 1: Download Arduino IDE.

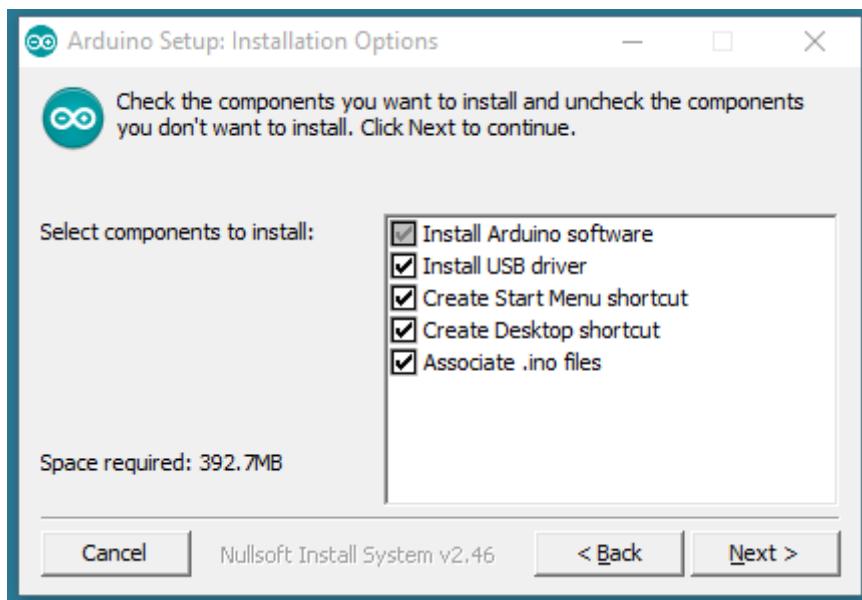
Truy cập đến trang chủ www.arduino.cc, tùy hệ điều hành đang làm việc mà chọn gói cài đặt thích hợp. Chúng ta có nhiều sự lựa chọn để cài đặt, như "STABLE version", "BETA BUILDS" hay "HOURLY BUILDS". Stable là phiên bản ổn định, đã được Arduino kiểm chứng và được đưa vào sử dụng. Các phiên bản như "BETA BUILDS" hay "HOURLY BUILDS" được các nhà phát triển xây dựng và có nhiều tính năng mới, tuy nhiên có thể phát sinh 1 số lỗi. Vì chúng ta mới bắt đầu nên khuyên dùng phiên bản STABLE.

Bước 2: Cài đặt Arduino IDE vào máy tính.

Với hệ điều hành Windows

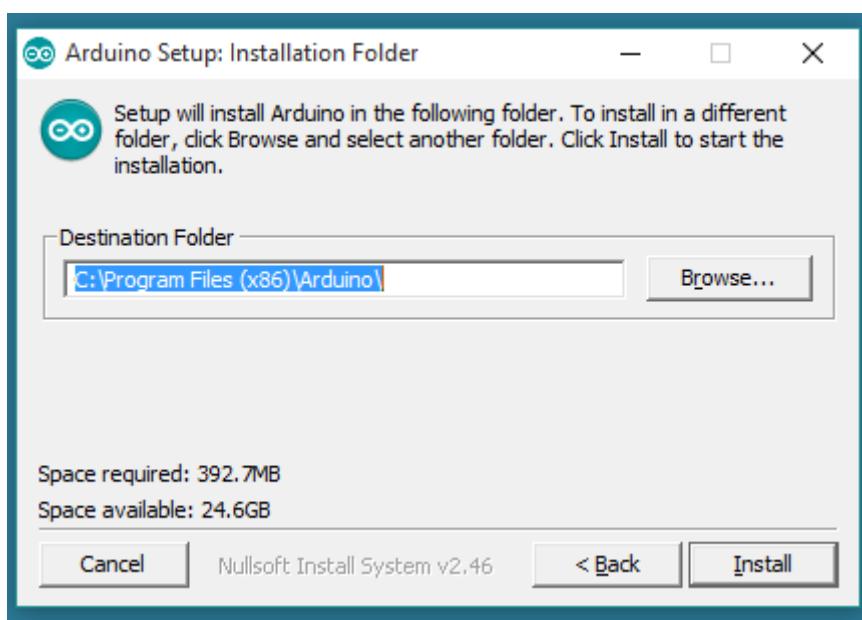
Chúng ta có thể download phiên bản Windows Installer (.exe) hoặc Windows Zip package. Installer giúp cài đặt trực tiếp mọi thứ ta cần bao gồm cả driver (khuyên dùng), với Zip package thì cần giải nén tập tin và cài đặt bằng tay. Tuy nhiên Zip package hữu ích khi muốn cài đặt với [phiên bản Portable](#).

Sau khi download hoàn thành, tiến hành cài đặt chương trình. Cho phép quá trình cài đặt driver lên máy tính nếu có thông báo từ hệ điều hành. Check vào các hộp thoại để cài đặt các thành phần đi kèm như hình bên dưới:

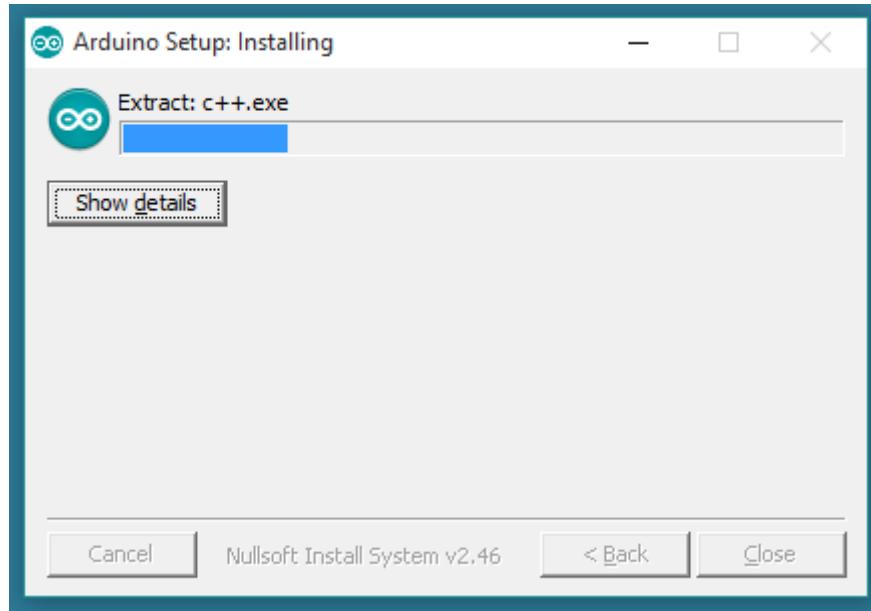


Hình 14. Cài đặt Arduino IDE (Nguồn www.arduino.cc)

Tiếp theo, chọn thư mục cài đặt (nên để theo mặc định) và chờ quá trình cài đặt hoàn tất.



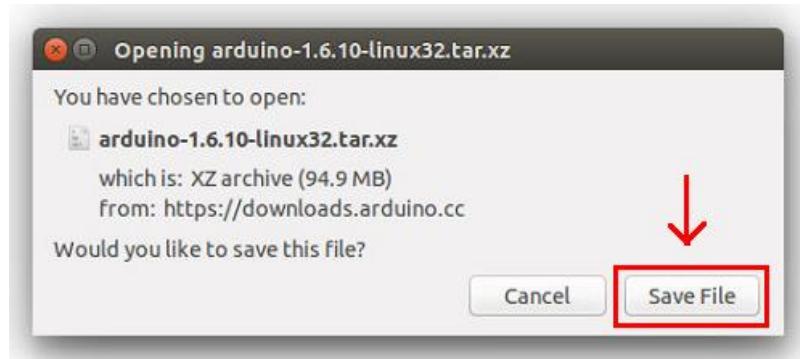
Hình 15. Chọn thư mục để cài đặt Arduino IDE (Nguồn www.arduino.cc)



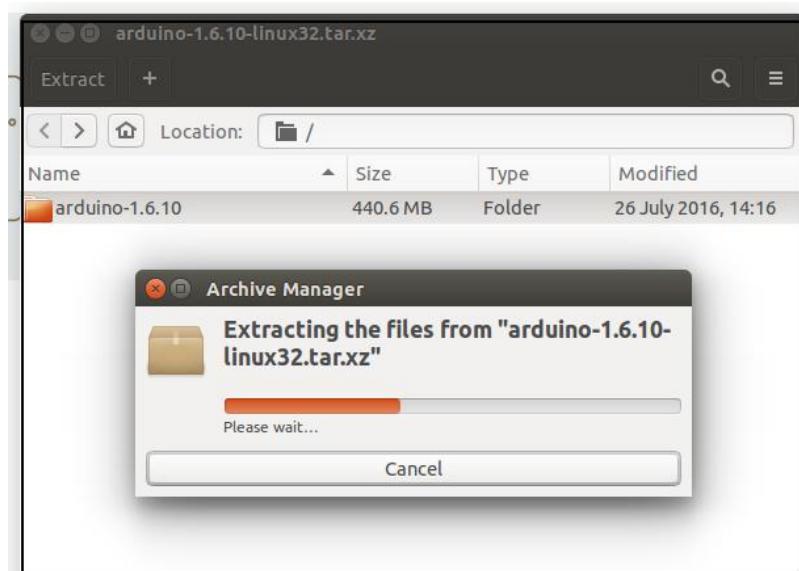
Hình 16. Quá trình giải nén và cài đặt Arduino lên máy tính (Nguồn www.arduino.cc)

Với hệ điều hành Linux

Lựa chọn phiên bản của hệ điều hành để chọn gói cài đặt thích hợp (Linux 32 bits, Linux 64 bits hoặc Linux ARM). Chọn **Save File** để download phần mềm về máy như hình dưới:

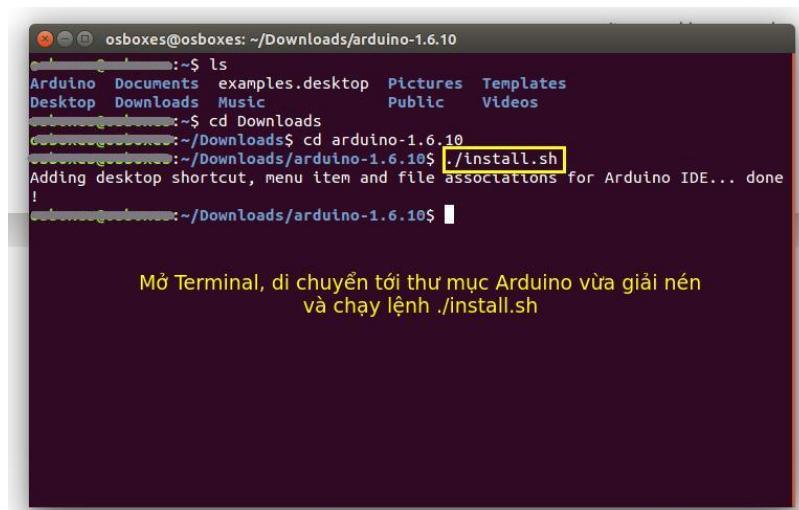


Hình 17. Download phần mềm Arduino (Nguồn www.arduino.cc)



Hình 18. Giải nén file Arduino vừa download về (Nguồn www.arduino.cc)

Click chuột phải vào thư mục vừa giải nén và chọn **Open in Terminal** (phím tắt Ctrl + Alt + T). Chạy lệnh **./install.sh** như hình bên dưới và chờ quá trình cài đặt hoàn tất. Sau khi cài đặt hoàn tất, sẽ có icon Arduino trên màn hình Desktop.



Hình 19. Chạy Arduino IDE (Nguồn www.arduino.cc)

Chú ý: Khắc phục lỗi sử dụng port khi upload sketch trên Linux.

Khi gặp 1 lỗi **Error opening serial port ...**, xử lý bằng cách thiết lập quyền sử dụng serial port.

Bước 1: Mở Terminal và gõ lệnh:

```
ls -l /dev/ttyUSB*
```

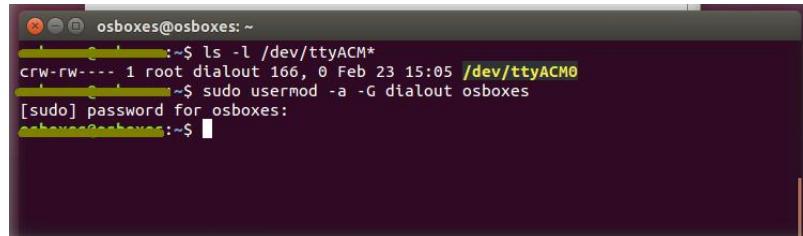
Chúng ta sẽ thấy một số nội dung giống như:

```
crw-rw---- 1 root dialout 188, 0 5 apr 23.01 ttyUSB0
```

Bước 2: Thiết lập quyền sử dụng serial port bằng cách thêm username vào group owner của file (dialout), sử dụng lệnh:

```
sudo usermod -a -G dialout <username>
```

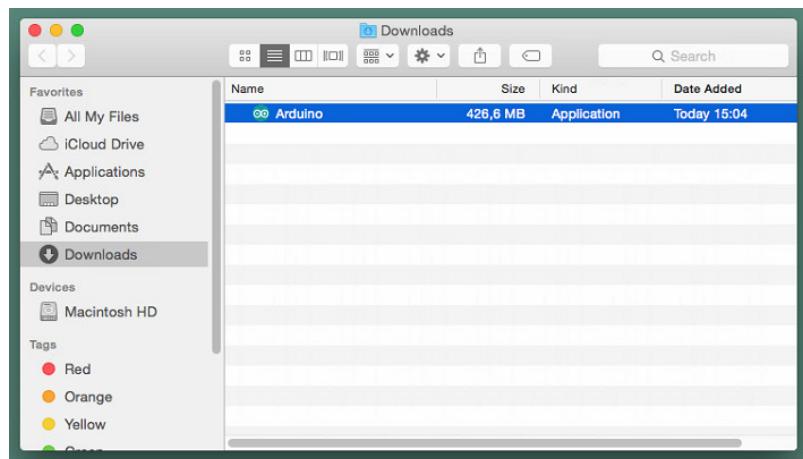
username là tên username khi sử dụng Linux. Thực hiện đăng xuất (log out) sau đó đăng nhập (log in) lại username để thay đổi có hiệu lực.



Hình 20. Hình ảnh xử lý lỗi opening serial port (Nguồn www.arduino.cc)

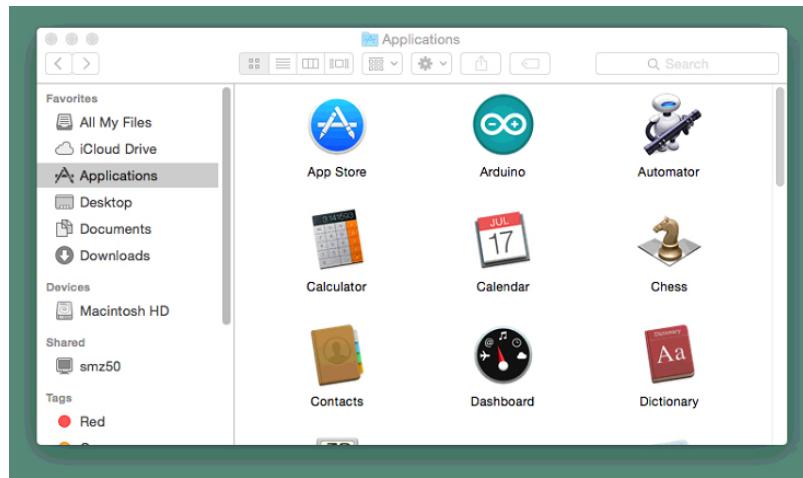
Với hệ điều hành MacOSX

Download phần mềm về máy và giải nén nó. Nếu sử dụng trình duyệt Safari thì sau khi download nó sẽ tự động giải nén.



Hình 21. Hình ảnh phần mềm Arduino sau khi download (Nguồn www.arduino.cc)

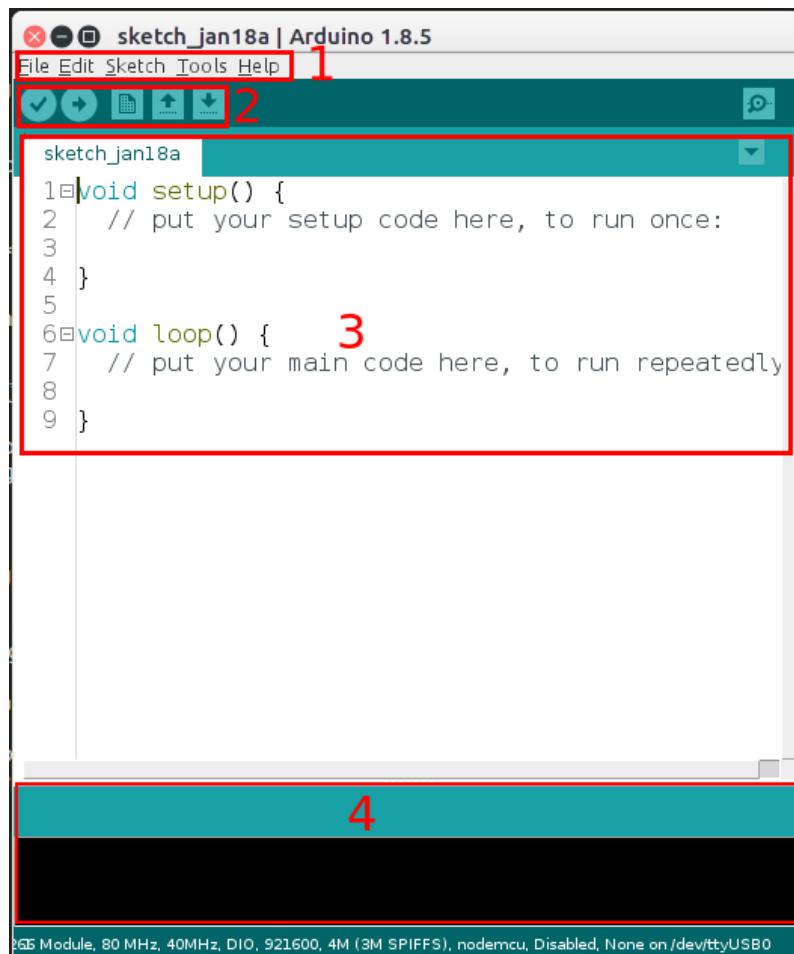
Copy Arduino vào thư mục Applications hay bất kì thư mục nào khác trên máy tính để hoàn tất quá trình cài đặt.



Hình 22. Hình ảnh Arduino trong mục Applications (Nguồn www.arduino.cc)

Sử dụng Arduino với board IoT Maker UnoX

Click vào icon Arduino để khởi động Arduino IDE, sau khi khởi động, phần mềm sẽ có giao diện như bên dưới:



Hình 23. Hình ảnh giao diện Arduino IDE

- Mục 1: Là thanh menu bar, bao gồm các tùy chọn thiết lập cho phần mềm Arduino và cho sketch đang thực hiện.

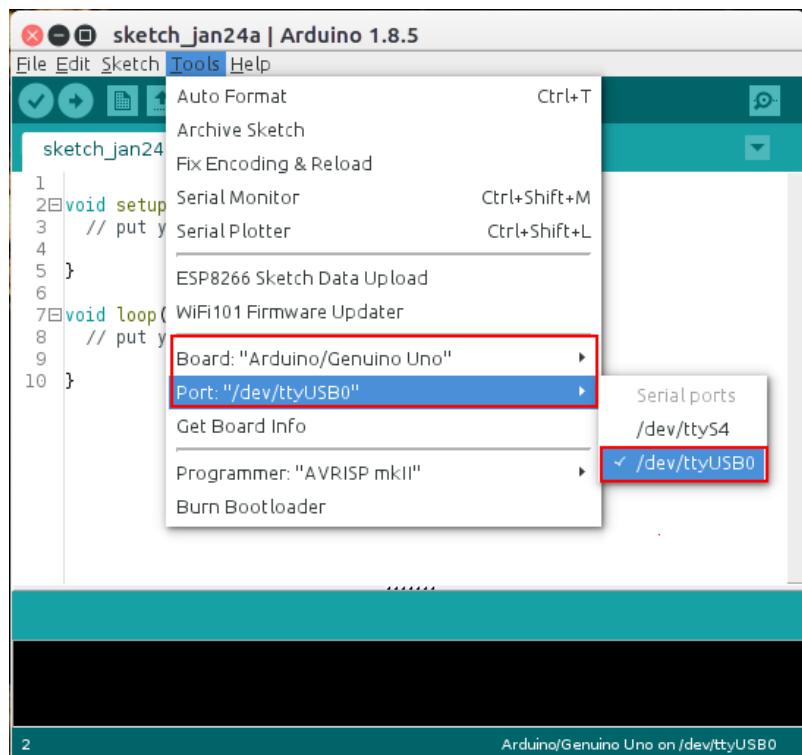
- **Mục 2:** Là thanh symbol bar, gồm các nút nhấn **Verify** để biên dịch sketch, **upload** để nạp sketch vào board, **New** để tạo sketch mới, **Open** để mở sketch, **Save** lưu sketch và **Serial Monitor** để mở serial port.
- **Mục 3:** Vùng để soạn thảo code cho sketch.
- **Mục 4:** Vùng hiển thị thông tin khi biên dịch, hiển thị quá trình nạp sketch và các thông báo lỗi khi biên dịch sketch (nếu có).

Sử dụng cable micro USB kết nối với máy tính và board IoT Maker UnoX như hình bên dưới:



Hình 24. Hình ảnh kết nối board IoT Maker UnoX với máy tính.

Trên thanh menu bar chọn **Tools**, ở mục **Boards** chọn **Arduino/Genuino Uno**, mục **Port** chọn cổng micro USB đã kết nối vào máy tính, tùy thuộc vào hệ điều hành mà các port này có tên gọi khác nhau, trên Linux thường là `/dev/ttyUSB0`, `/dev/ttyUSB0,...`, với Windows thường là `COM1`, `COM2,...`, với Mac OS thường là `/dev/tty.wchusbserial1420,...`.

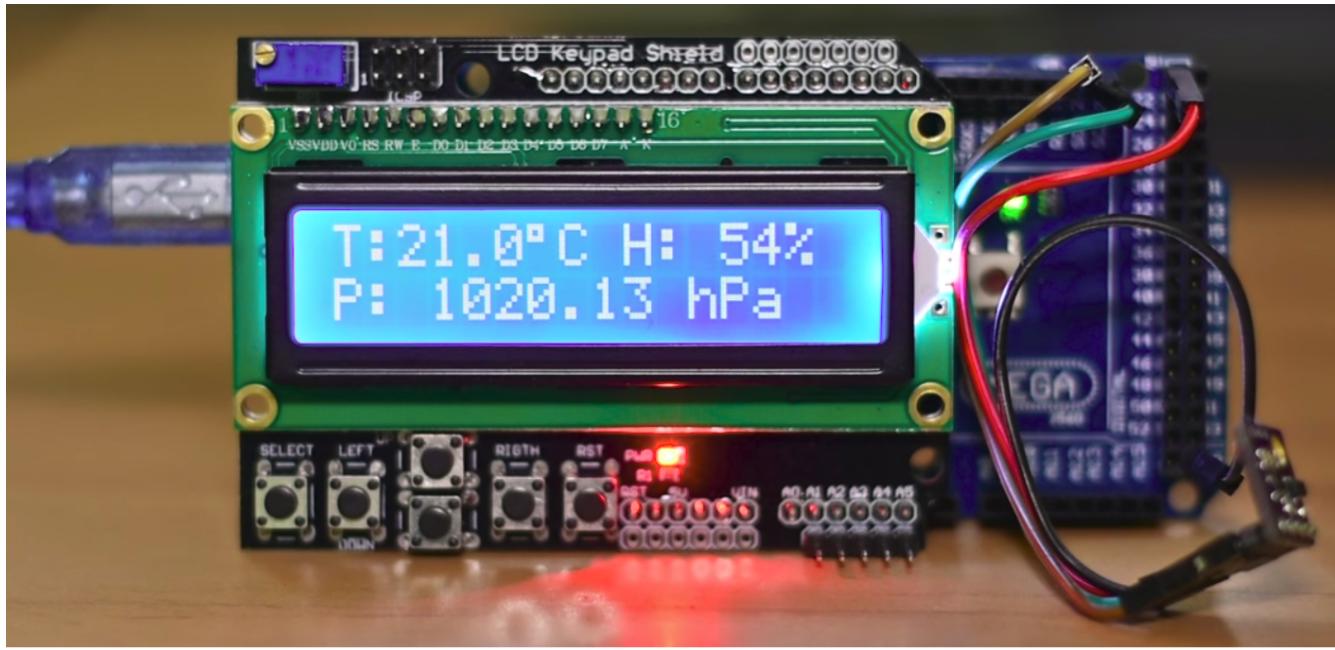


Hình 25. Hình ảnh cấu hình cho board IoT Maker UnoX trên Arduino

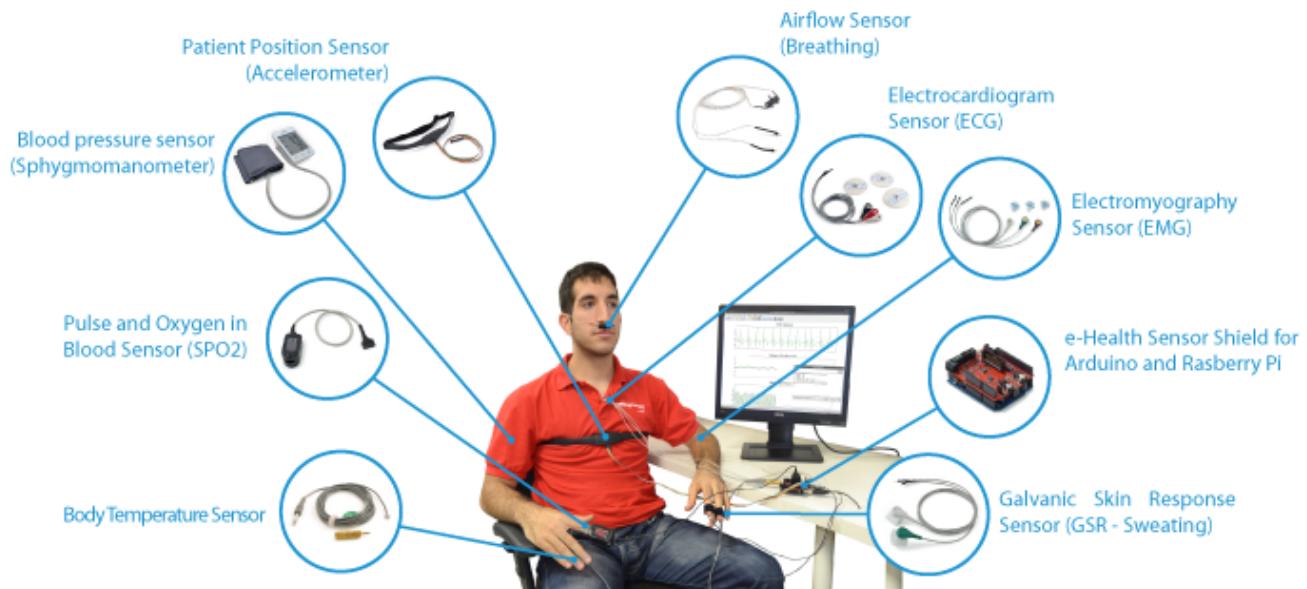
Sau đó, chúng ta đã có thể viết source code và sử dụng Arduino với board IoT Maker UnoX.

Ứng dụng mang lại

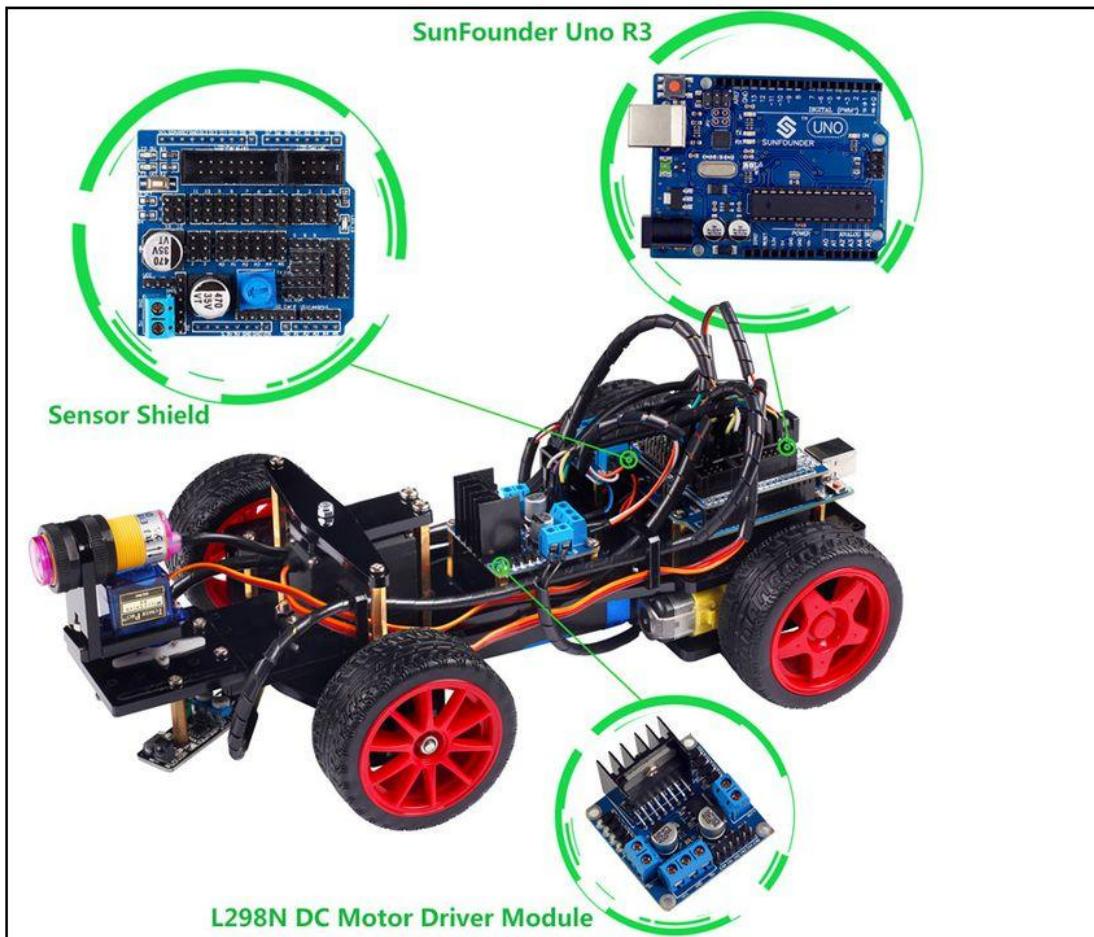
Hiện nay, Arduino được sử dụng rất rộng rãi trong rất nhiều dự án và rất nhiều lĩnh vực trong đời sống, từ giám sát, điều khiển môi trường, thu thập dữ liệu, thời trang, y tế,... . Một số hình ảnh bên dưới cho chúng ta thấy phần nào những ứng dụng mà Arduino mang lại trong cuộc sống.



Hình 26. Arduino trong thu thập và điều khiển nhiệt độ, độ ẩm



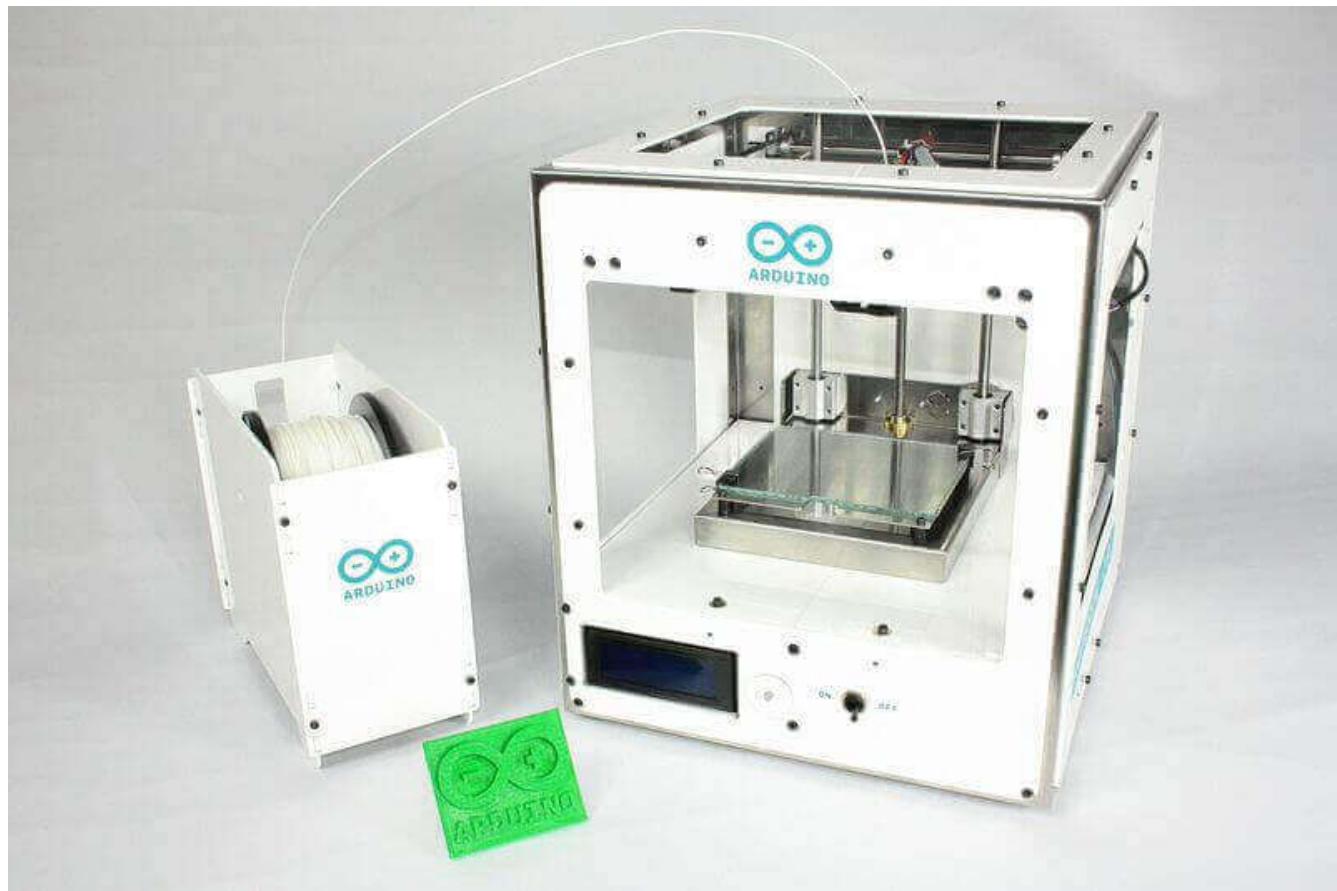
Hình 27. Arduino trong hệ thống chăm sóc sức khỏe



Hình 28. Xe điều khiển từ xa



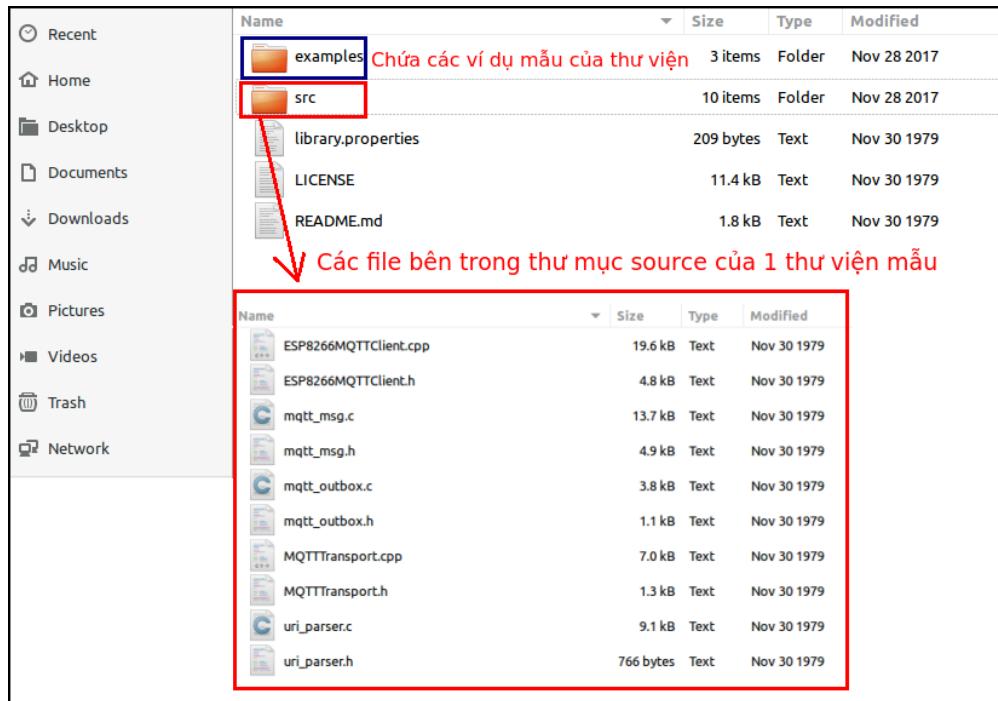
Hình 29. Arduino trong thời trang



Hình 30. Arduino với máy in 3D

Arduino và C/C++

Một thư viện mẫu của Arduino thường có cấu trúc như hình bên dưới :



Hình 31. Cấu trúc 1 thư viện mẫu trong Arduino

Một phần source code của folder src như hình :

```

OPEN FILES
FOLDERS
  ▾ ESP8266MQTTClient
    ▾ examples
    ▾ MQTTClient
    ▾ MQTTOverWebsocketClient
    ▾ MQTTSecureClient
    ▾ src
      ▷ ESP8266MQTTClient.cpp
      ▷ ESP8266MQTTClient.h
      ▷ MQTTTransport.cpp
      ▷ MQTTTransport.h
      ▷ mqtt_msg.c
      ▷ mqtt_msg.h
      ▷ mqtt_outbox.c
      ▷ mqtt_outbox.h
      ▷ uri_parser.c
      ▷ uri_parser.h
    ▾ LICENSE
    ▾ README.md
    ▾ library.properties

mqtt_msg.c
25 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
26 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
27 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
28 * POSSIBILITY OF SUCH DAMAGE.
29 *
30 */
31
32 #include <stdlib.h>
33 #include <string.h>
34 #include "mqtt_msg.h"
35 #define MQTT_MAX_FIXED_HEADER_SIZE 3
36 #define PROTOCOL_NAMEv311
37 enum mqtt_connect_flag
38 {
39   MQTT_CONNECT_FLAG_USERNAME = 1 << 7,
40   MQTT_CONNECT_FLAG_PASSWORD = 1 << 6,
41   MQTT_CONNECT_FLAG_WILL_RETAIN = 1 << 5,
42   MQTT_CONNECT_FLAG_WILL = 1 << 2,
43   MQTT_CONNECT_FLAG_CLEAN_SESSION = 1 << 1
44 };
45
46 struct __attribute__((__packed__)) mqtt_connect_variable_header
47 {
48   uint8_t lengthMsb;
49   uint8_t lengthLsb;
50   #if defined(PROTOCOL_NAMEv31)
51   uint8_t magic[6];
52   #elif defined(PROTOCOL_NAMEv311)
53   uint8_t magic[4];
54   #else
55   #error "Please define protocol name"
56   #endif
57   uint8_t version;
58   uint8_t flags;
59   uint8_t keepaliveMsb;
60   uint8_t keepaliveLsb;
61 };

```

Hình 32. Một phần source code của 1 thư viện mẫu

Để tạo ra những hàm đơn giản cho chúng ta sử dụng khi dùng Arduino, các nhà phát triển (developer) viết nên những thư viện, những thư viện đều viết bằng C/C++, vì vậy muốn giỏi lập trình

Arduino hay bất kì nền tảng lập trình với vi điều khiển nào hiện nay thì điều kiện tiên quyết là bạn phải sử dụng tốt ngôn ngữ lập trình C/C++.



Một số nền tảng lập trình vi điều khiển khác có thể sử dụng các ngôn ngữ lập trình khác như Python, Java,... tuy nhiên C được sử dụng rất rộng rãi trong việc lập trình vi điều khiển để phát triển các hệ thống nhúng. Ngoài ra, khi có kiến thức tốt về C/C++, việc học 1 ngôn ngữ khác cũng sẽ dễ dàng và nhanh hơn rất nhiều.

Trong tài liệu này sẽ không hướng dẫn về C/C++, thay vào đó, sẽ có phụ lục về 1 số thuộc tính cơ bản tại mục cheasheet ở cuối sách.



Một số tài liệu học lập trình C cho người mới bắt đầu được nhiều người sử dụng là [Head First C](#) của 2 tác giả Dawn Griffiths, David Griffiths và sách [learn-c-the-hard-way](#) của tác giả Zed A Shaw.

Tổng kết

Qua phần này, chúng ta đã hiểu về Arduino là gì cũng như các công cụ, môi trường cần thiết để xây dựng 1 dự án với nền tảng phát triển Arduino, đồng thời đã có thể bắt đầu phát triển ứng dụng với Arduino. Các công cụ được lựa chọn đều là đa nền tảng, dễ dàng được sử dụng cho các hệ điều hành Mac OS, Windows, hay Linux.

Một số website giúp chúng ta hiểu rõ hơn về Arduino và các công cụ hỗ trợ đi kèm:

Arduino

- Trang chủ www.arduino.cc, bao gồm tất cả các thông tin liên quan đến Arduino của thế giới.
- Trang arduino.vn, diễn đàn trao đổi các thông tin liên quan đến các dự án Arduino tại Việt Nam.
- Trang openstem.vn, Các dự án nhỏ sử dụng Arduino bằng tiếng Việt.

Ngôn ngữ lập trình C

- Tài liệu [Head first C](#) và [learn-c-the-hard-way](#) có thể phù hợp để bắt đầu với ngôn ngữ lập trình C.

Hello World

Bất kỳ một chương trình học nào cũng cần nên bắt đầu một cách từ từ. Bởi vì thời điểm này chúng ta đều mới bắt đầu, nhiều khái niệm, kiến thức về lĩnh vực này gần như không có nhiều. **Helloworld** giúp các bạn có thể nắm được các kiến thức cơ bản về điện tử, làm sao để biên dịch, nạp được chương trình trong Arduino, cũng như nắm được một số kiến thức về kiến trúc chương trình của Arduino. Nội dung sẽ tìm hiểu ở phần này như sau:

- Giới thiệu các khái niệm, linh kiện điện tử cơ bản.
- Blink LED.
- Nút nhấn và các ứng dụng.
- Sử dụng chức năng PWM trong Arduino.
- Đọc dữ liệu Analog từ cảm biến.

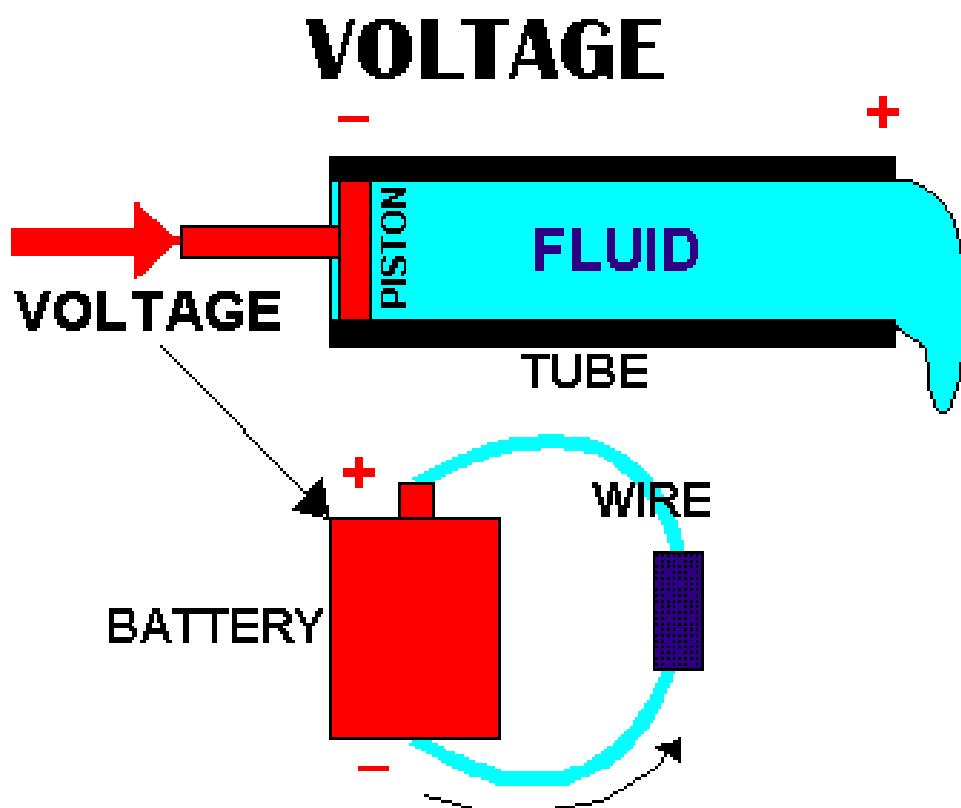
Giới thiệu một số khái niệm và linh kiện điện tử cơ bản

Trong phần này chúng ta sẽ tìm hiểu về 1 số khái niệm cũng như 1 số linh kiện điện tử cơ bản. Các kiến thức này cũng đã được trình bày chi tiết ở chương trình vật lí bậc phổ thông. Mỗi khái niệm hay linh kiện sẽ có những video giúp chúng ta dễ hiểu hơn những khái niệm cũng như cách hoạt động của các linh kiện điện tử thông dụng.

Điện áp, dòng điện và điện trở

Điện áp

Điện áp hay còn gọi là hiệu điện thế (từ tiếng Anh là voltage) là sự chênh lệch về điện áp giữa 2 điểm, nó là công thực hiện được để di chuyển một hạt điện tích trong trường tĩnh điện từ điểm này đến điểm kia. Hiệu điện thế có thể đại diện cho nguồn năng lượng (lực điện), hoặc sự mất đi, sử dụng, hoặc năng lượng lưu trữ (giảm thế).



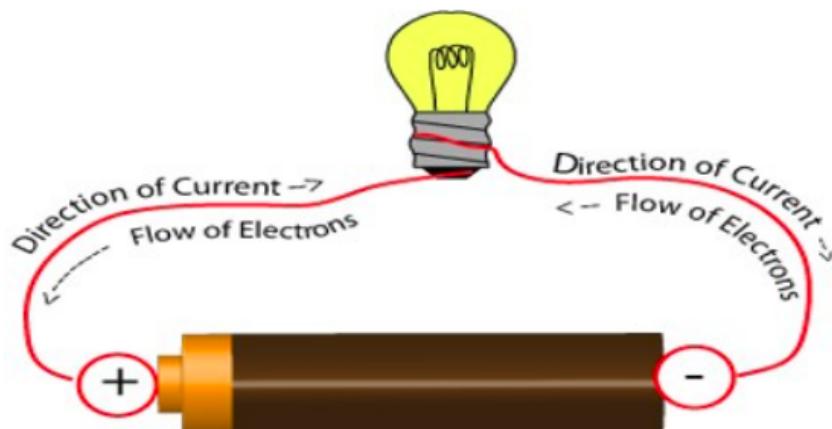
Hình 33. Hình ảnh minh họa hiệu điện thế sinh ra dòng điện tương ứng với dòng chảy của nước trong ống dẫn.

Dòng điện

Dòng điện (tiếng Anh: electric current) là dòng chuyển dịch có hướng của các hạt mang điện. Trong mạch điện, các hạt mang điện phần lớn là các electron chuyển động bên trong dây dẫn. Kim loại là chất dẫn điện phổ biến nhất, kim loại có hạt nhân mang điện tích dương không thể di chuyển, chỉ có các electron tích điện âm có khả năng di chuyển tự do trong vùng dẫn, do đó, trong kim loại các electron là các hạt mang điện.

Chiều dòng điện được quy ước là chiều đi từ cực dương qua dây dẫn và các thiết bị điện đến cực âm của nguồn. Do dòng điện được qui ước là dòng chuyển dời có hướng của các điện tích dương, chính vì thế, trong mạch điện với dây dẫn kim loại, các electron tích điện âm dịch chuyển **ngược chiều** với chiều của dòng điện trong dây dẫn.

Sự chuyển dịch có hướng của các điện tích sinh ra do tác động của điện trường gây ra bởi hiệu điện thế. Do đó, có thể hiểu là điện áp sinh ra dòng điện trong một mạch điện, hay nói đơn giản là "điện áp có trước dòng điện" trong mạch điện.



Hình 34. Hình ảnh minh họa hiệu ứng điện thế sinh ra dòng điện trong mạch điện (Nguồn Assignment Point)

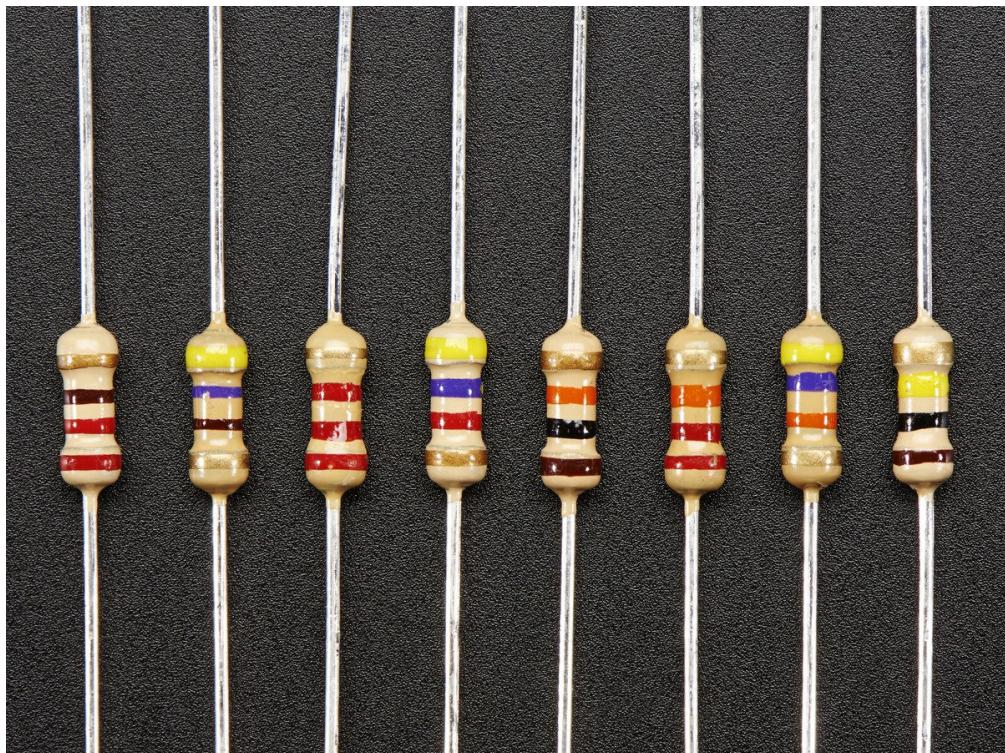
Chúng ta có thể xem giải thích về dòng điện và điện áp theo cách dễ hiểu hơn với một số video:

- What is Voltage: www.youtube.com/watch?v=V1ulri4s_E8
- Electricity and Circuits: www.youtube.com/watch?v=D2m9nVkcKX

Điện trở

Điện trở (tiếng Anh: electric resistance) là một đại lượng đặc trưng cho khả năng cản trở dòng điện của một vật. Đơn vị của điện trở là Ω . Khái niệm điện trở của vật xuất phát từ định luật Ohm.

Điện trở gồm 2 tiếp điểm kết nối, thường được dùng để hạn chế cường độ dòng điện chạy trong mạch, điều chỉnh mức độ tín hiệu, dùng để chia điện áp, kích hoạt các linh kiện điện tử chủ động như transistor và có trong rất nhiều ứng dụng khác.



Hình 35. Hình ảnh của điện trở trong thực tế.

Định luật Ohm (Ohm's law)

Đây là 1 định luật cơ bản nhưng rất quan trọng trong lĩnh vực điện, điện tử. Định luật này đề cập đến mối quan hệ giữa dòng điện, điện áp và điện trở. Tên định luật được đặt theo nhà phát minh ra nó, nhà vật lí nổi tiếng người Đức, Georg Simon Ohm (1789–1854).

Phát biểu định luật:

Cường độ dòng điện chạy qua dây dẫn tỉ lệ thuận với hiệu điện thế đặt vào 2 đầu dây dẫn và tỉ lệ nghịch với điện trở của dây dẫn.

Công thức của định luật Ohm:

$$I = \frac{V}{R}$$

Trong đó:

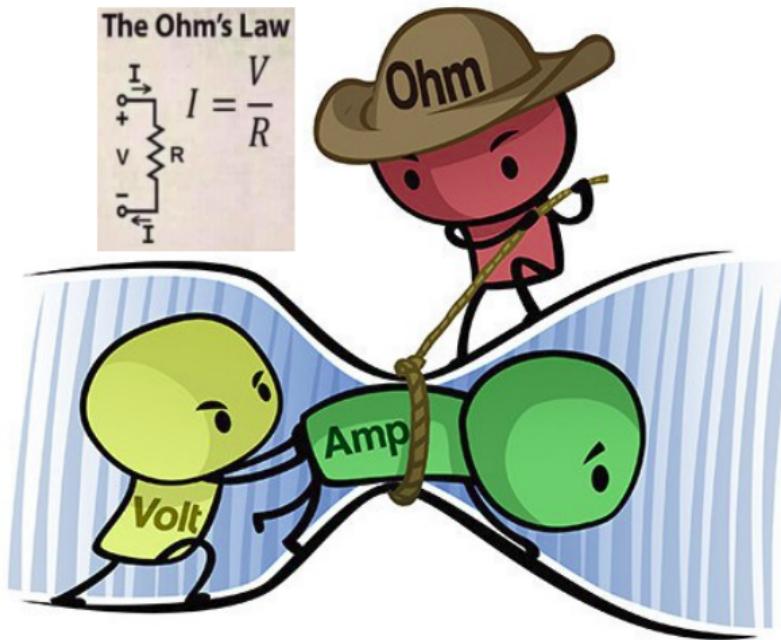
- **V**: Ký hiệu của điện áp (Voltage).
- **I**: Ký hiệu của dòng điện (Current).

- R: Ký hiệu của điện trở (Resistor).



Ký hiệu của dòng điện là I bởi vì nó lấy chữ cái đầu của từ Intensity, có nghĩa là cường độ - đại lượng đặc trưng của dòng điện.

Một hình ảnh minh họa về định luật Ohm:



Hình 36. Hình ảnh minh họa định luật Ohm (Nguồn: www.build-electronic-circuits.com)

Một video giải thích về định luật Ohm: www.youtube.com/watch?v=iLzfe_HxrWI

Tụ điện



Hình 37. Hình ảnh các loại tụ điện trong thực tế.

Khái niệm

Tụ điện là linh kiện điện tử gồm 2 vật dẫn đặt gần nhau. Mỗi vật dẫn đó gọi là một bản của tụ điện. Khoảng không gian giữa hai bản có thể là chân không hay bị chiếm bởi một chất điện môi nào đó.

Tụ điện là một linh kiện được sử dụng rất phổ biến và gần như không thể thiếu trong các mạch điện tử, mỗi tụ điện đều có một công dụng nhất định như lọc nhiễu cho mạch, tạo dao động, truyền dẫn tín hiệu,...

Các thông số đặc trưng

- **Điện dung:** Đại diện cho khả năng tích điện của tụ. Đơn vị là Fara (F).
- **Điện áp làm việc:** Đó là giá trị điện áp cao nhất mà tụ điện có thể chịu được, thông thường giá trị này sẽ được ghi trên thân của tụ điện (nếu tụ đủ lớn). Nếu giá trị điện áp trên tụ lớn hơn giá trị điện áp làm việc thì lớp điện môi bên trong tụ điện sẽ bị đánh thủng và gây ra sự chập tụ, nổ tụ. Hiện tượng này khá nguy hiểm nên chúng ta cần cẩn thận khi chọn tụ điện cho mạch điện của mình.
- **Nhiệt độ làm việc:** Đó là nhiệt độ ở vùng đặt tụ điện trong mạch điện. Tụ điện nên được chọn với nhiệt độ làm việc cao nhất của nơi chúng ta đặt tụ điện phải cao hơn nhiệt độ này.

Để hiểu rõ hơn về cách hoạt động của tụ điện, có thể tham khảo đường dẫn www.youtube.com/watch?v=5hFC9ugTGLs

Cuộn cảm



Hình 38. Hình ảnh về cuộn cảm trong thực tế (Nguồn: www.talkingelectronics.com)

Khái niệm

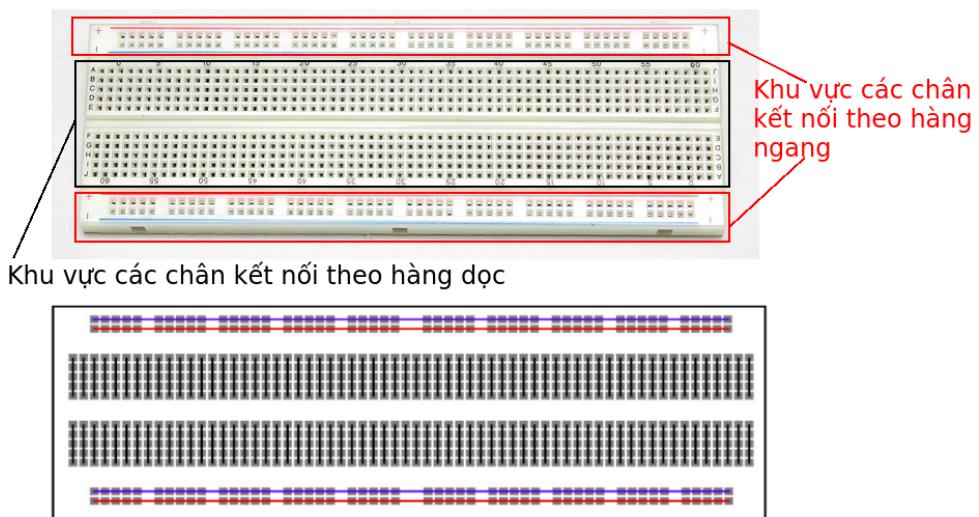
Cuộn cảm (tiếng Anh là coil hay inductor) là cuộn dây bao gồm nhiều vòng dây dẫn điện quấn quanh một lõi vật liệu từ. Dựa vào hiện tượng cảm ứng điện từ và hiện tượng từ hóa vật liệu từ mà người ta sử dụng cuộn cảm cho các mục đích khác nhau như làm phàn ứng (stator) trong các máy phát điện xoay chiều, lọc nhiễu trong các mạch điện tử, tạo ra các nam châm điện, các công tắc điện tử, ...

Mỗi cuộn cảm có một độ tự cảm (hay hệ số tự cảm hoặc từ dung), kí hiệu là L, đo bằng đơn vị Henry (H) đặc trưng cho khả năng sinh suất điện động cảm ứng và tích lũy năng lượng điện từ.

Để hiểu rõ hơn về cách hoạt động và chức năng của cuộn cảm, chúng ta có thể tham khảo đường dẫn www.youtube.com/watch?v=NgwXkUt3XxQ

Breadboard

Breadboard là 1 dụng cụ giúp kết nối các thiết bị điện tử lại với nhau thông qua các lỗ cắm. Hình ảnh của breadboard cũng như cách hoạt động của nó được mô tả ở hình bên dưới:

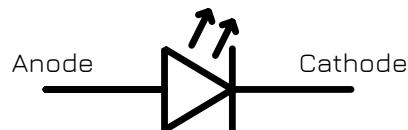


Chớp tắt bóng LED

Kiến thức

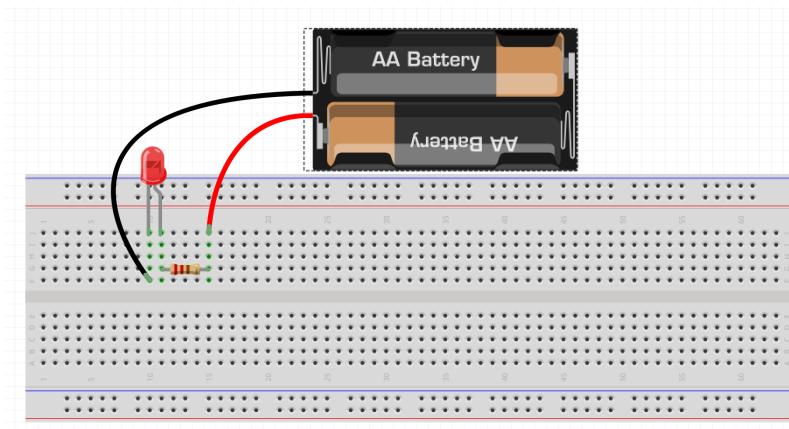
LED là chữ viết tắt của Light Emitting Diodes, nó là bóng bán dẫn có thể phát sáng với màu sắc khác nhau tùy thuộc vào chất liệu bán dẫn. Để điều khiển được bóng LED cần cung cấp mức điện áp chênh lệch giữa cực âm và cực dương của bóng LED cao hơn mức điện áp V_f (datasheet), thường là 3.2VDC, và dòng điện nhỏ hơn mức chịu đựng của nó, thường là 15mA.

Chúng ta có thể hiểu rõ hơn cách hoạt động của LED tại đường dẫn www.youtube.com/watch?v=BH9Ll973H8w



Hình 40. Ký hiệu LED trên mạch điện. (Anode +, Cathode -)

Mạch có thể chạy được như sau:



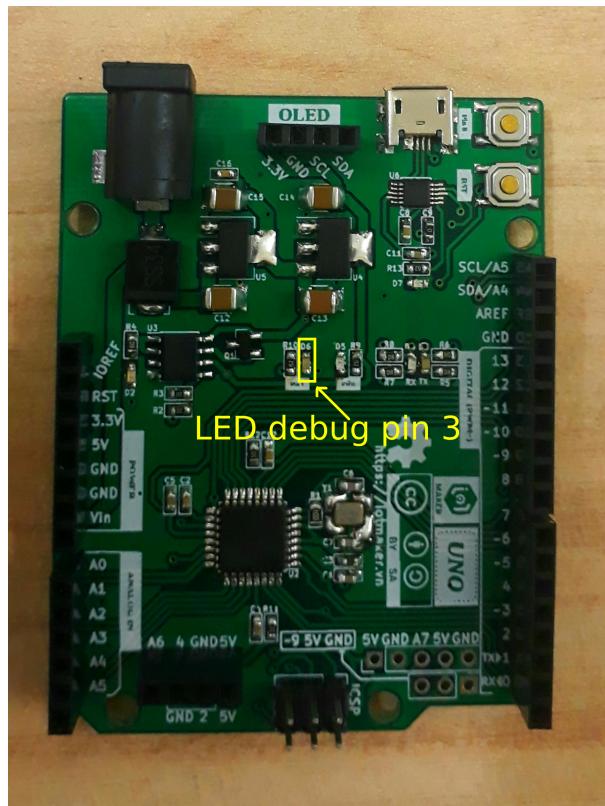
Hình 41. Hình ảnh 1 mạch điện đơn giản của LED.



Điện trở giúp hạn chế dòng điện qua LED, để nó ở trạng thái hoạt động bình thường.

Đầu nối

Trên board IoT Maker UnoX có sẵn 1 LED kết nối với chân D2 của chip Atmega328 (chân số 3 trên board IoT Maker UnoX) nhằm debug chương trình.



Hình 42. Hình ảnh LED trên board IoT Maker UnoX.

Sử dụng dây micro USB để cấp nguồn và nạp chương trình cho board IoT Maker UnoX.



Hình 43. Kết nối board IoT Maker UnoX với máy tính.

Mở phần mềm Arduino để tạo sketch và viết source code, tham khảo thông tin về [Sử dụng Arduino với board IoT Maker UnoX](#) và [Giới thiệu board IoT Maker UnoX và IoT Arduino STEM Kit](#).

Có 2 phương pháp để điều khiển chớp, tắt LED được giới thiệu ở mục này đó là dùng hàm `delay()` hoặc dùng định thời với hàm `millis()`.

Mã nguồn chớp tắt dùng Delay

Cách hoạt động chương trình được giải thích trong source code.

```
#define pinLed 3 // Định nghĩa pinLed là chân số 3.

void setup()      // Hàm setup() được gọi 1 lần duy nhất khi bật nguồn hoặc reset board
{
    pinMode(pinLed, OUTPUT); // Cấu hình chân pinLed là ngõ ra.
}

void loop() { // Hàm loop() sẽ được gọi liên tục khi chương trình hoạt động.

    digitalWrite(pinLed, HIGH); // Bật Led (HIGH - có nghĩa là mức điện áp 5VDC)
    delay(1000);               // Chờ 1000 mili giây = 1s
    digitalWrite(pinLed, LOW); // Tắt Led (LOW có nghĩa là mức điện áp 0VDC)
    delay(1000);               // Chờ 1000 mili giây = 1s
}
```

Giải thích về 2 hàm `pinMode()` và `digitalWrite()` tại [\[pinMode-digitalWrite\]](#)

Mã nguồn chớp tắt dùng định thời

Khi thực hiện chương trình có sử dụng hàm `delay()`, vì điều khiển phải chờ cho đến khi hết thời gian `delay` mới thực hiện các tác vụ khác, thời gian `delay` nhỏ thì không sao, tuy nhiên nếu giá trị này lớn sẽ làm ảnh hưởng đến các tác vụ khác khi chạy cùng thời điểm, làm tăng độ trễ khi thực thi chương trình hoặc làm cho chương trình chạy không chính xác, việc dùng định thời với hàm `millis()` sẽ khắc phục tình trạng này.

```
#define pinLed 3      // Định nghĩa pinLed là chân số 3.

int ledState = LOW;    // khai báo biến lưu trạng thái của LED
unsigned long previousMillis = 0;
const long interval = 1000;

void setup()
{
  pinMode(pinLed, OUTPUT);
}

void loop()
{
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    if (ledState == LOW)
      ledState = HIGH; // Đổi trạng thái
    else
      ledState = LOW; // Đổi trạng thái
    digitalWrite(pinLed, ledState);
  }
}
```

Giải thích code

- Biến `ledState` nhằm lưu trữ trạng thái của LED tại thời điểm hiện tại.
- Biến `interval` là giá trị của 1 bước thời gian tính theo ms.
- Lúc đầu giá trị `previousMillis = 0`; hàm `millis()` đã bắt đầu hoạt động và đếm thời gian, nó trả về số mili giây từ khi board được cấp nguồn hoặc reset board.
- Lệnh `currentMillis = millis()` sẽ gán giá trị của biến `currentMillis` bằng với giá trị hiện tại của hàm `millis()` trả về. Nếu `thời gian hiện tại - thời gian bắt đầu > interval`, chương trình sẽ thực hiện 2 việc:
 - `previousMillis = currentMillis` nhằm reset giá trị đếm, để bắt đầu tính lại thời gian (chú ý rằng `millis()` vẫn tiếp tục chạy và biến `currentMillis` vẫn đang được gán giá trị của hàm `millis()`).
 - Đổi trạng thái của LED (nếu đang mức `LOW` thì chuyển sang `HIGH`).



Việc bật tắt LED chỉ thực hiện khi `currentMillis - previousMillis >= interval`, trong khoảng thời gian khác thì ta có thể thực thi các tác vụ khác của chương trình.



Chúng ta có thể tìm hiểu chi tiết về hàm `millis()` tại đường dẫn garretlab.web.fc2.com/en/arduino/inside/arduino/wiring.c/millis.html == Nút nhấn

Kiến thức

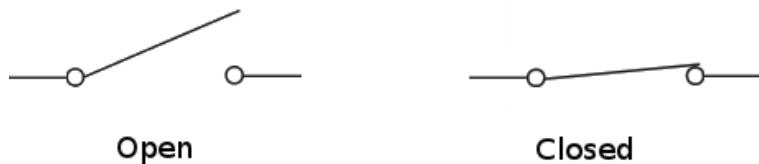
Nút nhấn sẽ giúp khởi động 1 hành động nào đó khi cần thiết. Những ứng dụng thực tế hầu như đều

cần những kích hoạt từ bên ngoài thông qua các loại nút nhấn như nút nhấn cảm ứng, nút nhấn lưu trạng thái on/off, nút nhấn nhiều trạng thái... Trong phần này, chúng ta sẽ tìm hiểu những vấn đề về sử dụng nút nhấn thông qua các ví dụ mẫu. Các loại nút nhấn trong thực tế như hình bên dưới:



Hình 44. Hình ảnh về các loại nút nhấn trong thực tế.

Với nút nhấn không giữ trạng thái, khi nhấn nút và giữ thì sẽ cho dòng điện chạy qua. Ngược lại, khi nhả hoặc không nhấn sẽ không cho dòng điện chạy qua. Nguyên lý hoạt động của nút nhấn không giữ trạng thái như hình dưới:



Hình 45. Mạch nguyên lý của nút nhấn không giữ trạng thái.

Chớp, tắt LED sử dụng nút nhấn

Yêu cầu

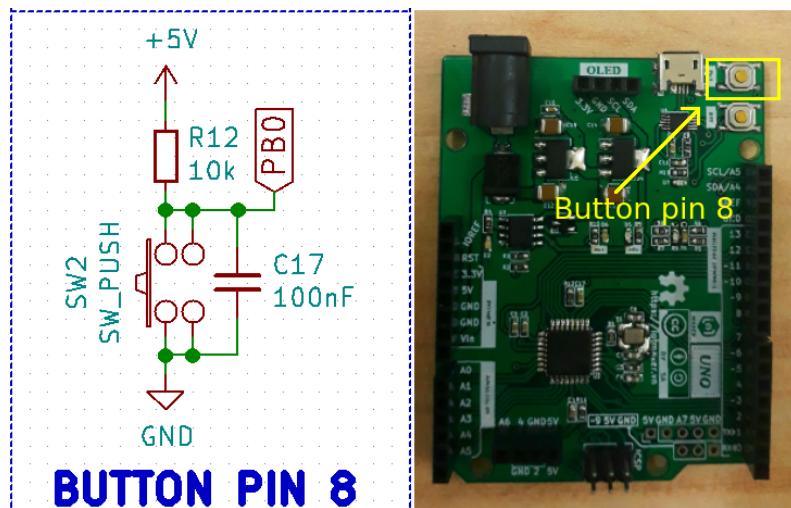
Nhấn nút thì đèn LED sáng, không nhấn nút đèn LED sẽ tắt.

Linh kiện cần dùng

- Board IoT Maker UnoX

Đầu nối

Trên board IoT Maker UnoX đã có sẵn nút nhấn kết nối với chân D8 nên không cần đấu nối thêm nút nhấn.



Hình 46. Hình ảnh sơ đồ nguyên lý và nút nhấn thực tế trên board.

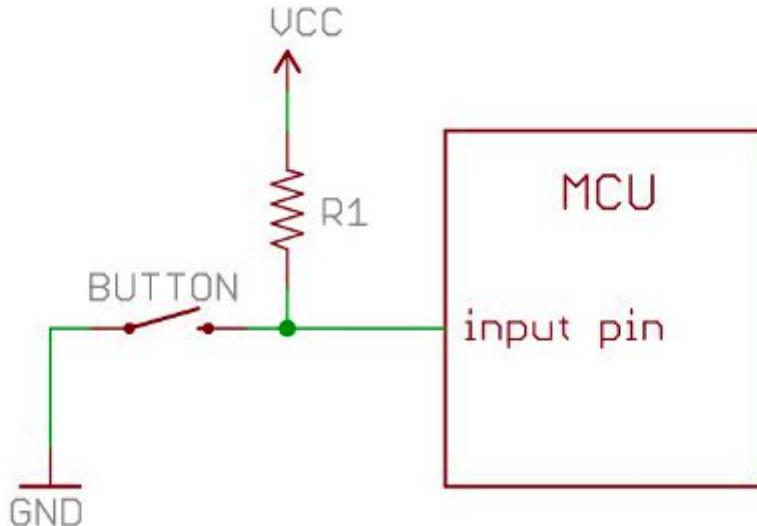
Điện trở kéo

Trong hình đấu nối của nút nhấn trên board IoT Maker UnoX có điện trở R12 nối lên nguồn. Đây được gọi là điện trở kéo lên. Điện trở kéo lên nguồn, hoặc kéo xuống ground (mass) được sử dụng rất thường xuyên trong vi điều khiển hoặc trong các mạch số (digital circuit).

Với vi điều khiển, 1 chân được cấu hình là INPUT (ngõ vào), nếu không có thiết bị hay mạch điện nào kết nối với nó. Khi vi điều khiển đọc trạng thái của chân đó thì chúng ta không xác định mức điện áp của nó được bởi nó có thể ở mức HIGH hoặc mức LOW, thuật ngữ thường được sử dụng để diễn tả trạng thái này là floating (trôi nổi).

Để khắc phục hiện tượng này, chúng ta sẽ kết nối chân này với 1 điện trở kéo lên nguồn (pull-up) hoặc 1 điện trở kéo xuống ground hay mass (pull-down) để đảm bảo chân đó chỉ ở 1 trong 2 trạng thái, hoặc HIGH, hoặc LOW.

Trên thực tế, điện trở kéo lên (pull-up) thường được sử dụng hơn so với điện trở kéo xuống (pull-down).



Hình 47. Hình ảnh mạch điện khi sử dụng điện trở kéo lên.

Như vậy, khi không nhấn button, trạng thái của pin sẽ ở mức HIGH và khi button được nhấn thì trạng thái sẽ ở mức LOW.



Nếu mắc mạch điện như trên mà không dùng điện trở kéo thì sẽ xảy ra hiện tượng ngắn mạch.

Thông thường, giá trị điện trở kéo nằm trong khoảng 4.7KΩ đến 10KΩ là phù hợp.

Source code

```
const int buttonPin = 8; // Chân kết nối với nút nhấn trên board
const int ledPin = 3; // Chân kết nối với LED trên board
int buttonState = 0; // Biến đọc trạng thái của nút nhấn.

void setup()
{
    pinMode(ledPin, OUTPUT);
    pinMode(buttonPin, INPUT);
    Serial.begin(115200);
    Serial.println("init serial");
}

void loop()
{
    buttonState = digitalRead(buttonPin); // Đọc trạng thái của nút nhấn.

    if (buttonState == LOW) { // Kiểm tra, nếu button đã được nhấn
        digitalWrite(ledPin, HIGH); // Bật LED
        Serial.println("PRESSED, LED ON"); // In ra màn hình Serial chữ PRESSED, LED ON
    } else {
        digitalWrite(ledPin, LOW);
        Serial.println("NOTHING, LED OFF");
    }
}
```

Giải thích source code

- Lệnh `const int buttonPin = 8` khai báo chân số 8 được kết nối với button trên board, từ khóa `const` để chỉ rằng đây là 1 biến không thể thay đổi, thường thì nó được dùng để định nghĩa các chân kết nối hoặc các biến mà người dùng không muốn thay đổi giá trị của biến đó trong chương trình.
- Lệnh `Serial.begin(115200)` nhằm khởi tạo giao tiếp Serial của vi điều khiển, giá trị 115200 là tốc độ truyền nhận dữ liệu.
- Lệnh `Serial.println("init serial")` Nhằm in ra chữ `init serial` trên Serial monitor của Arduino IDE. Chúng ta sẽ tìm hiểu kĩ hơn về giao tiếp Serial ở chương tiếp theo [Truyền thông nối tiếp](#)

Kết quả

- Khi nhấn Button trên board IoT Maker UnoX thì LED debug trên board sẽ sáng. Không nhấn sẽ tắt.
- Trên Serial monitor hiện ra dòng chữ `HIGH` khi nhấn nút và `LOW` khi không nhấn nút.

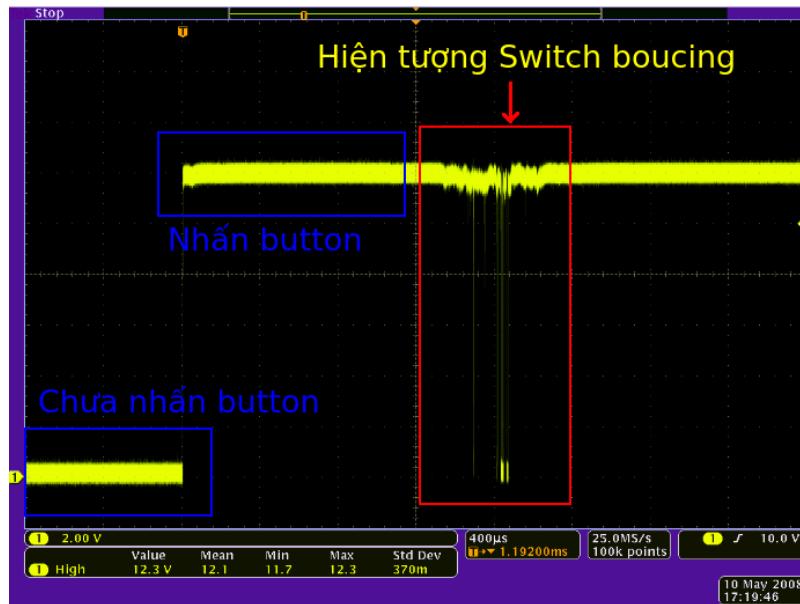


Hình 48. Hình ảnh trên Serial terminal của Arduino IDE

Chớp, tắt LED và chống dội phím khi nhấn (switch debouncing)

Một số board mạch khác, khi dùng nút nhấn, chúng ta nhận thấy có vài lần chương trình in ra dòng chữ "PRESSED, LED ON" và "NOTHING, LED OFF" xen kẽ nhau như khung màu vàng ở hình bên trên, nguyên nhân của hiện tượng này được giải thích như sau:

- Nút nhấn là 1 linh kiện cơ khí, tại thời điểm bắt đầu nhấn nút, các tiếp điểm cơ khí tiếp xúc với nhau tạo ra sự nhiễu điện áp. Mặc dù chúng ta đã nhấn nút, tuy nhiên khi tiến hành đọc trạng thái của chân kết nối với nút nhấn, giá trị điện áp có lúc HIGH, có lúc LOW. Hiện tượng này được gọi là **SWITCH BOUNCING**.
- Hiện tượng này diễn ra rất nhanh và mắt thường không nhìn thấy được, tuy nhiên vi điều khiển có tốc độ xử lý lệnh rất nhanh, nếu sử dụng serial terminal ta có thể thấy kết quả của hiện tượng này. Ngoài ra, chúng ta có thể sử dụng máy Oscilloscope để thấy rõ kết quả thực tế:



Hình 49. Một kết quả đo trên máy Oscilloscope mô tả hiện hiện Switch bouncing(Nguồn www.pololu.com)



Nếu không khắc phục hiện tượng này, nó có thể làm hỏng chương trình của bạn.

Có 2 phương pháp để khắc phục tình trạng này đó là:

- **Sử dụng bằng phần cứng:** Mặc thêm tụ điện song song với nút nhấn. Bình thường, tụ điện được nối với VCC và GND nên ở trạng thái tích lũy năng lượng (charge). Khi nhấn nút, tụ điện sẽ chuyển sang chế độ giải phóng năng lượng (discharge), giá trị điện áp tại chân kết nối với nút nhấn sẽ giảm 1 cách từ từ xuống 0V, việc làm chậm quá trình giảm điện áp này sẽ ngăn ngừa hiện tượng switch bouncing. Nút nhấn trên board IoT Maker UnoX có sử dụng chống dội phím bằng phần cứng này.
- **Sử dụng bằng phần mềm:** Cũng dựa trên cách tạo 1 thời gian trễ nhất định (delay) đồng thời kiểm tra trạng thái trước đó của nút nhấn để có kết quả chính xác. Source code khi sử dụng phần mềm mô tả như bên dưới:

```

const int buttonPin = 8; // Chân kết nối với button trên board Iotmaker Uno X
const int ledPin = 3; // Chân kết nối với LED trên board
int ledState = HIGH; // Biến lưu trạng thái hiện tại của chân kết nối đến LED
int buttonState; // Biến lưu trạng thái hiện của nút nhấn
int lastButtonState = HIGH; // Biến lưu trạng thái trước đó của nút nhấn.

unsigned long lastDebounceTime = 0; // Biến lưu thời gian delay chống dội phím ở lần cuối cùng.
unsigned long debounceDelay = 50; // Thời gian delay để chống dội phím nhấn

void setup()
{
    pinMode(buttonPin, INPUT);
    pinMode(ledPin, OUTPUT);
    Serial.begin(115200);
    digitalWrite(ledPin, ledState);
}

void loop()
{
    int reading = digitalRead(buttonPin);
    if (reading != lastButtonState) {
        lastDebounceTime = millis();
    }

    if (((millis() - lastDebounceTime) > debounceDelay) &&
        (reading != buttonState)) {
        buttonState = reading;
        if (buttonState == LOW) {
            ledState = !ledState;
        }
    }
}

digitalWrite(ledPin, ledState);

lastButtonState = reading;
}

```

Flow-work của chương trình

Biến **reading** đọc trạng thái của button ở thời điểm hiện tại và kiểm tra, nếu giá trị này khác trạng thái cuối của button (Giá trị thiết lập ban đầu là HIGH - không nhấn), có nghĩa là chúng ta đã nhấn nút thì chương trình bắt đầu đếm thời gian chống dội phím . Nếu thời gian lớn hơn giá trị đã cài đặt, chương trình sẽ kiểm tra lại 1 lần nữa trạng thái của nút nhấn, nếu nó vẫn giữ nguyên và không thay đổi (button vẫn đang được nhấn) thì sẽ :

- Gán biến trạng thái của nút nhấn bằng biến **reading**.
- Đảo trạng thái LED nếu button được nhấn (mức LOW).

Sau đó, điều khiển LED sáng bằng lệnh **digitalWrite()**, gán giá trị trạng thái nút nhấn bằng với biến **reading** để kiểm tra cho những vòng lặp sau. Như vậy, cứ mỗi lần nhấn nút và LED sẽ thay đổi trạng thái.

Sử dụng ngắn (interrupt) để điều khiển LED,

Trong 2 ví dụ trước, chúng ta sử dụng nút nhấn để điều khiển LED trên board với cách thức hỏi vòng

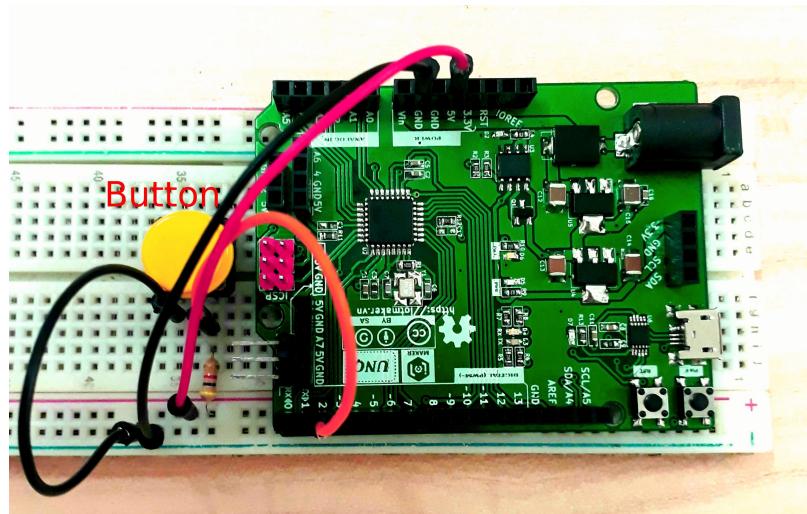
(polling), nghĩa là vi điều khiển sẽ kiểm tra liên tục trạng thái của nút nhấn trong vòng lặp loop, cách này thường chiếm dụng nhiều tài nguyên của CPU đồng thời độ đáp ứng của chương trình cũng không nhanh bằng cách sử dụng ngắt (interrupt).

Chip Atmega328P có 2 ngắt trên các chân D2 và D3. Nếu bạn muốn sử dụng chức năng ngắt cho các chân khác thì phải cài đặt thêm 1 số lệnh nữa, vấn đề này tương đối phức tạp và chúng ta không đề cập ở đây.

Yêu cầu

Nhấn button thì đèn LED trên board IoT Maker UnoX sẽ đảo trạng thái [nếu LED đang sáng thì sẽ tắt và ngược lại].

Đầu nối



Hình 50. Hình ảnh kết nối chân ngắt trên mạch IoT Maker UnoX.

Source code

```

int ledPin = 3;      // Chân kết nối với LED trên board Iotmaker Uno X
int btnPin = 2;      // Chấn có chức năng interrupt trên board
int ledState = LOW; // Gán trạng thái LED ban đầu là LOW

void blink()
{
    if (ledState == LOW) {
        ledState = HIGH;
    } else {
        ledState = LOW;
    }

    digitalWrite(ledPin, ledState);
}

void setup()
{
    pinMode(ledPin, OUTPUT);
    pinMode(btnPin, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterruption(btnPin), blink, FALLING);
}

void loop()
{
    // Không phải làm gì
}

```

Giải thích chương trình

Lệnh `attachInterrupt()` bao gồm 3 đối số:

- Lệnh `digitalPinToInterruption(btnPin)`: Chuyển chân digital hiện tại sang chức năng ngắn.
- `Blink()`: Làm sẽ được gọi khi có sự kiện ngắn xảy ra.
- `FALLING`: Phát hiện ngắn xảy ra nếu có 1 xung cạnh xuống ở chân ngắn. Đây là 1 trong các chế độ (mode) phát hiện có ngắn xảy ra, các mode là : LOW, RISING, FALLING, HIGH.

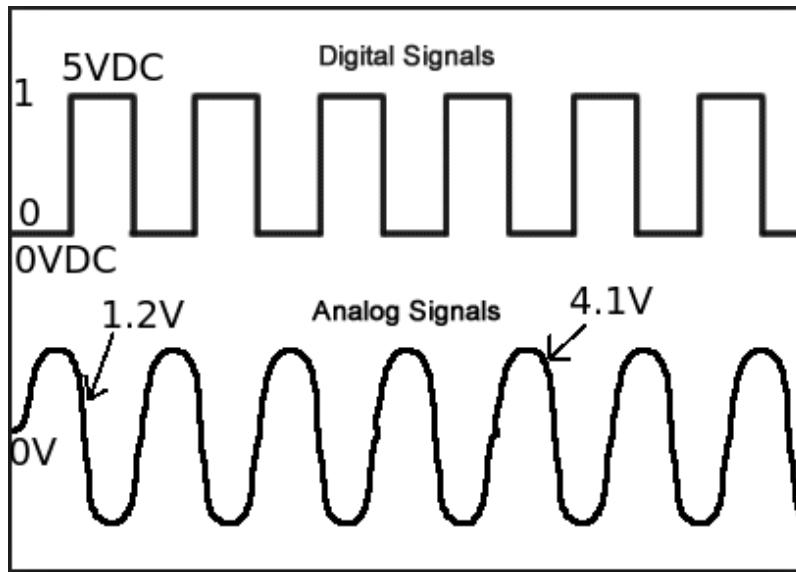


Chúng ta sẽ tìm hiểu sâu hơn về interrupt tại phần [\[timer-interrupt\]](#) == Sử dụng chức năng PWM

Analog và Digital

Trước khi tìm hiểu về PWM, chúng ta nên bắt đầu với 1 khái niệm đơn giản hơn nhưng vô cùng quan trọng trong điện tử, đó là [Digital signal](#) và [Analog signal](#).

Từ đầu cuốn sách đến giờ, chúng ta chỉ sử dụng các mức điện áp VCC hoặc HIGH (mức 1) và 0V hoặc GND (mức 0). Trên thực tế, thế giới ta đang sống hầu như là các tín hiệu Analog. Ví dụ như màu sắc, nhiệt độ, độ ẩm hay cường độ ánh sáng của môi trường,...



Hình 51. Hình ảnh so sánh tín hiệu Analog và Digital.

Hiểu 1 cách đơn giản, Analog là những tín hiệu liên tục. Ví dụ trong dải điện áp từ 0-5VDC, tại 1 thời điểm, giá trị điện áp có thể là 1 số bất kì giữa 0 và 5 và thường được biểu diễn dưới giống như dạng sóng. Nó hiện hữu trong đời sống như âm thanh ta nghe được, hình ảnh ta thấy ... Với Digital thì ngược lại, nó là những tín hiệu rời rạc. Ví dụ trong dải điện áp từ 0-5VDC, tại 1 thời điểm, giá trị điện áp chỉ có thể là 0V hoặc 5V, nó giống như 1 cái công tắc, 1 bức ảnh đen-trắng,...

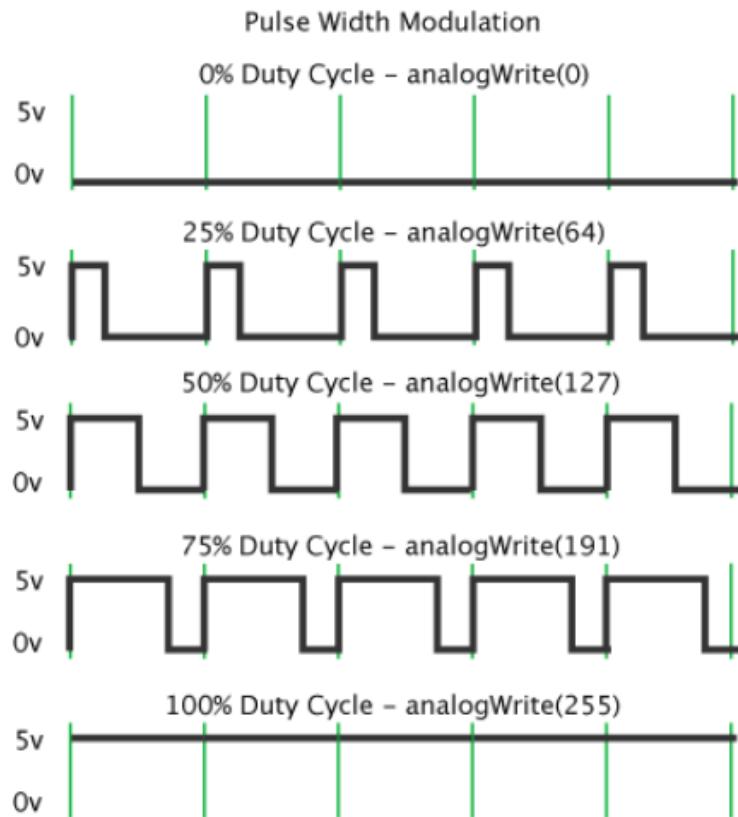
Trong máy tính, vi điều khiển hoặc các thiết bị điện tử, tín hiệu số thường được sử dụng bởi nó có thể dễ dàng lưu trữ cũng như xử lý dữ liệu. Để có thể xử lý, chuyển đổi dữ liệu tương tự sang dữ liệu số, người ta thường chia nhỏ các phần của tín hiệu tương tự (quá trình này gọi là lấy mẫu), sau đó các phần chia nhỏ này được gán cho các giá trị 0 hoặc 1.

Các vi điều khiển thường có bộ chuyển đổi dữ liệu tương tự sang dữ liệu số gọi là ADC (Analog to Digital Converter) và 1 số vi điều khiển có luôn 1 bộ chuyển đổi dữ liệu số sang tương tự (DAC).

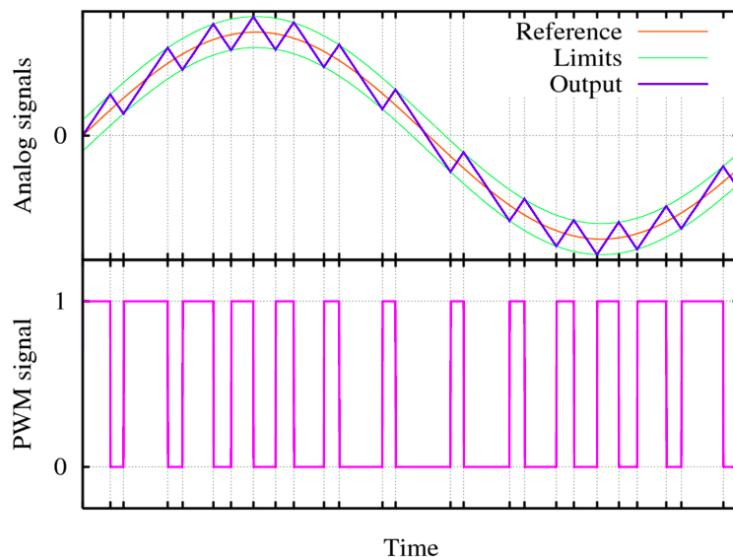
PWM

PWM là chữ viết tắt của Pulse Width Modulation, dịch theo nghĩa tiếng Việt có nghĩa là điều chế độ rộng xung. Đây là 1 công nghệ giúp các tín hiệu số cho ra kết quả gần giống như tín hiệu tương tự. Tín hiệu điều khiển số tạo ra bởi các xung (pulse), đó là sự lặp lại của việc thay đổi điện áp giữa mức 0 và mức 1. Yếu tố đặc trưng của PWM là chu kỳ và độ rộng xung.

Ví dụ, với 1 bóng LED, trong 1 chu kỳ sáng, nếu thời gian ở mức HIGH dài hơn thời gian mức LOW thì đèn LED sẽ sáng mạnh, thời gian mức HIGH thấp hơn thời gian mức LOW thì đèn sẽ sáng thấp hơn. Lợi dụng tính tăng này ta có thể thay đổi thời gian mức HIGH và mức LOW để điều chỉnh cường độ sáng của bóng LED. Tỉ số giữa thời gian của mức HIGH và mức LOW được gọi là **Duty cycle**.



Hình 52. Hình ảnh về sự thay đổi duty cycle trong PWM (nguồn www.arduino.cc).



Hình 53. Hình ảnh về mối tương quan giữa PWM và tín hiệu analogWrite (nguồn commons.wikimedia.org).

Fade LED

Fade LED là 1 ví dụ cơ bản giúp chúng ta có thể sử dụng PWM trong Arduino.

Yêu cầu

Tự động thay đổi cường độ sáng của LED trên board IoT Maker UnoX.

Linh kiện cần dùng

Sử dụng [board IoT Maker UnoX](#), LED đã tích hợp sẵn trên board tại chân D3.

Source code

```
int ledPin = 3;      // Chân kết nối với LED trên board Iotmaker Uno X
int brightness = 0;  // Biến thiết lập cường độ sáng cho LED trên board
int fadeAmount = 15; // Biến thiết lập 1 bước thay đổi cường độ sáng.

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  analogWrite(ledPin, brightness);

  brightness = brightness + fadeAmount;

  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }
  delay(30);
}
```

Giải thích source code

- Biến `brightness`: Thiết lập cường độ sáng cho led, `fadeAmount` là 1 bước thay đổi cường độ sáng. Sử dụng hàm `analogWrite()` nhằm set giá trị cường độ sáng cho `ledPin`. LED sẽ sáng dần đến giá trị 255 sau đó sẽ tắt dần khi đến giá trị 0 và lặp đi lặp lại nhờ điều kiện trong câu lệnh `if`.
- Giá trị từ 0 đến 255 tương ứng với Duty cycle từ 0% đến 100%.

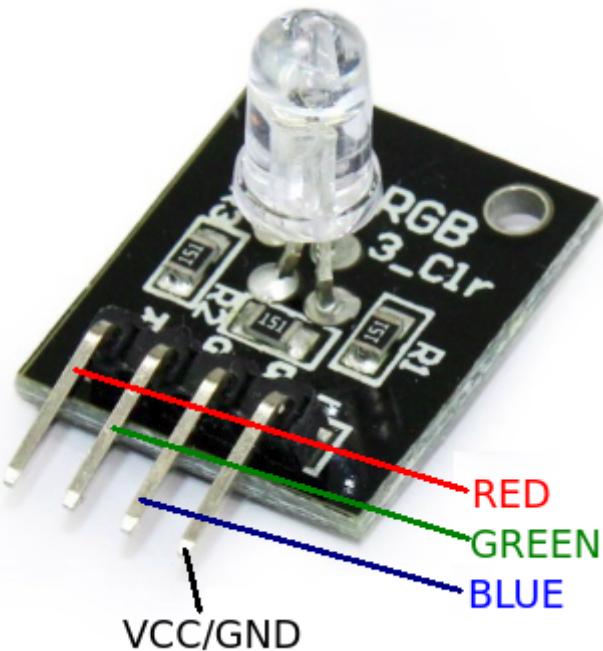


Arduino không có bộ chuyển đổi DAC nhưng chúng ta có thể dùng PWM để có thể cho ra tín hiệu ngõ ra ở dạng [gần như Analog](#), giá trị maximum = 255 là giá trị lớn nhất của chuyển đổi PWM chứ không liên quan đến bộ ADC (10 bits) của chip Atmega328.

Điều khiển LED RGB

Giới thiệu

LED RGB là module LED có 3 chân tín hiệu điều khiển tương ứng với 3 màu đỏ (Red), xanh lá (Green) và xanh dương (Blue). Chúng ta có thể phối hợp giữa các màu để tạo ra các hiệu ứng màu sắc đẹp mắt.



Hình 54. Hình ảnh module LED RGB

Yêu cầu

Sử dụng button với 4 chế độ điều khiển:

- Nhấn lần 1, LED sáng màu đỏ (Red).
- Nhấn lần 2, LED sáng màu xanh lá (Green).
- Nhấn lần 3, LED sáng màu xanh dương (Blue).
- Nhấn lần 4, LED sáng cả 3 màu.
- Nhấn lần 5, quay lại như lúc nhấn lần 1.

Linh kiện cần dùng

- [Module LED RGB](#)
- [Dây cắm breadboard male-famale](#)
- [Board IoT Maker UnoX](#) [Nút nhấn đã có sẵn trên board].

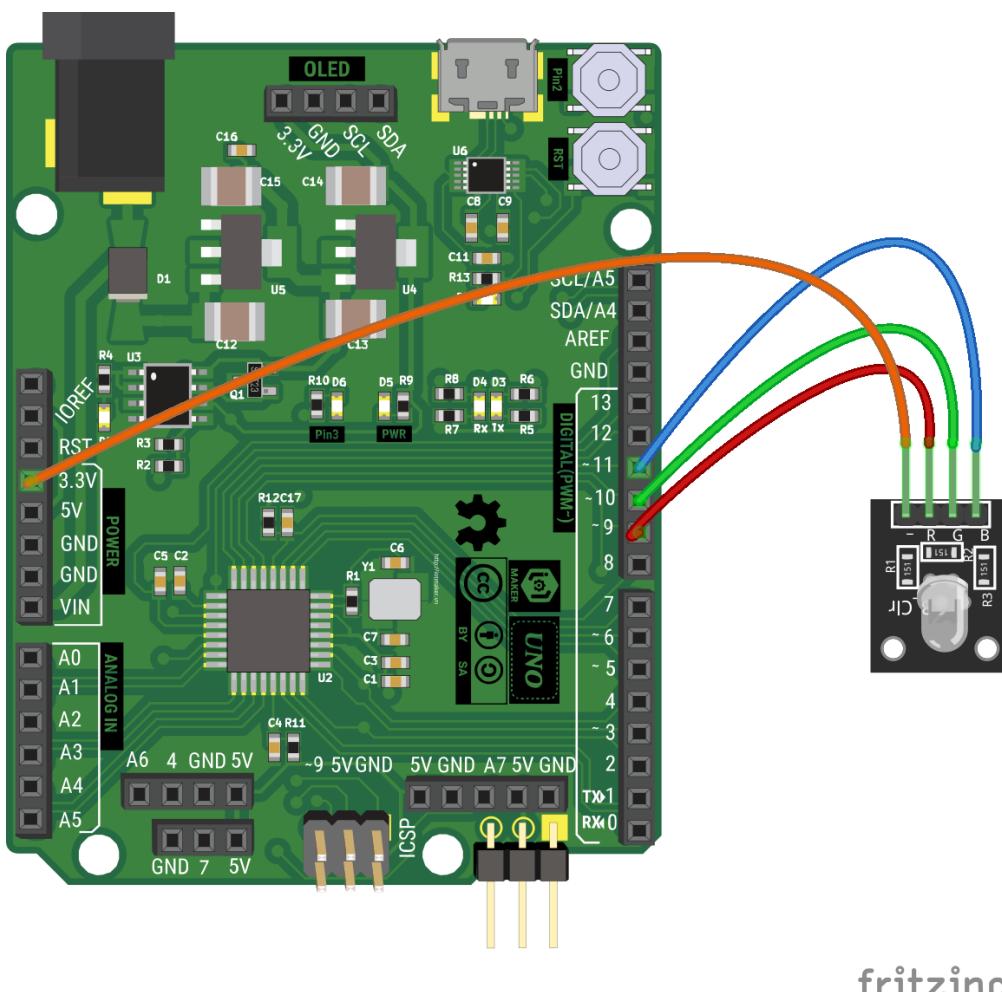
Đầu nối

Trên board IoT Maker UnoX có các chân kí hiệu ~ đều có thể sử dụng chức năng PWM.

Bảng 4. Bảng đấu nối chân của module LED RGB và board IoT Maker UnoX

Số thứ tự	Chân trên Module LED RGB	Chân trên board IoT Maker Uno X
1	R	9

Số thứ tự	Chân trên Module LED RGB	Chân trên board IoT Maker Uno X
2	G	10
3	B	11
4	V/G	3.3V hoặc GND



Hình 55. Hình ảnh kết nối module LED RGB với board IoT Maker | Inox

Source code

Nội dung source code đã được giải thích trong file

```
#define pinLedRed    9      // Chân kết nối với pin R của module LED-RGB
#define pinLedGreen   10     // Chân kết nối với pin G của module LED-RGB
#define pinLedBlue    11     // Chân kết nối với pin B của module LED-RGB
#define pinButton     8      // Nút nhấn trên board Iotmaker Uno X

boolean lastPinButton = LOW; // Biến lưu trạng thái cuối của nút nhấn
boolean currentPinButton = LOW; // Biến lưu trạng thái hiện tại của nút nhấn
int ledMode = 0; // Các chế độ của nút nhấn.

void setup()
{
    // Cài đặt các hướng của các chân
    pinMode (pinLedRed, OUTPUT);
    pinMode (pinLedGreen, OUTPUT);
    pinMode (pinLedBlue, OUTPUT);
```

```

pinMode (pinButton, INPUT);
}

// Function chống dội phím khi nhấn (debouncing)
boolean debounce(boolean lastState)
{
    boolean currentState;           // Biến currentState chỉ có 2 trạng thái LOW hoặc HIGH, khai báo kiểu boolean
    // nhằm tiết kiệm tài nguyên CPU.
    currentState = digitalRead(pinButton); // Đọc trạng thái của button
    if (lastState != currentState) {      // Nếu nhấn Nút, chờ 20 mili giây, sau đó đọc lại trạng thái button 1 lần nữa.

        delay(20);
        currentState = digitalRead(pinButton);
    }
    return currentState;               // Trả về trạng thái của button hiện tại
}

void setMode(int mode)
{
    if (mode == 1) {                // Màu đỏ: button nhấn lần 1

        digitalWrite(pinLedBlue, HIGH);
        digitalWrite(pinLedGreen, LOW);
        digitalWrite(pinLedRed, LOW);
    } else if (mode == 2) {          // Màu xanh lá: button nhấn lần 2

        digitalWrite(pinLedBlue, LOW);
        digitalWrite(pinLedGreen, HIGH);
        digitalWrite(pinLedRed, LOW);
    } else if (mode == 3) {          // Màu xanh dương: button nhấn lần 3

        digitalWrite(pinLedBlue, LOW);
        digitalWrite(pinLedGreen, LOW);
        digitalWrite(pinLedRed, HIGH);
    } else if (mode == 4) {          // Sáng cả 3 màu: button nhấn lần 4

        int fadeAmount = 0;           // Bước thay đổi cường độ sáng
        int brightness;
        digitalWrite(pinLedBlue, LOW);
        digitalWrite(pinLedGreen, LOW);
        digitalWrite(pinLedRed, LOW);

        for (fadeAmount = 0; fadeAmount < 255; fadeAmount += 10) {
            brightness = fadeAmount;
            analogWrite(pinLedBlue, brightness);
            analogWrite(pinLedGreen, brightness);
            analogWrite(pinLedRed, brightness);
            delay(30);                  // delay 30 mili giây để hiển thị kết quả
        }
    }
}

void loop()
{
    currentPinButton = debounce(lastPinButton); // Đọc trạng thái của nút nhấn ở function debounce()
    if (lastPinButton == HIGH && currentPinButton == LOW) // Trạng thái nút nhấn đã thay đổi => đã nhấn phím
        ledMode++;
    lastPinButton = currentPinButton;           // Reset lại trạng thái nút nhấn.
    if (ledMode == 5)                          // Đang ở chế độ sáng 3 màu và nhấn button thì chuyển sang chế
    độ sáng màu đỏ (lặp lại chu kỳ)
        ledMode = 1;
    setMode(ledMode);                         // Đưa biến ledMode vào hàm setMode() để thực thi chế độ sáng
    tương ứng.
}

```

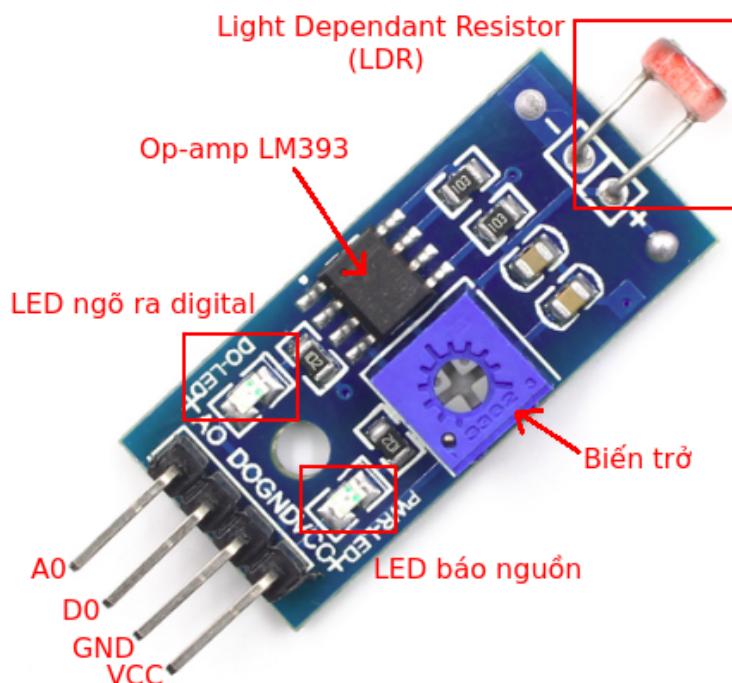


Với `ledMode == 4`, chương trình sẽ chạy hết vòng lặp for mới kiểm tra trạng thái của button. Ta có thể sử dụng interrupt để chương trình tối ưu hơn. == Đọc dữ liệu Analog

Ở phần trước chúng ta đã hiểu các khái niệm cơ bản về Analog, phần này chúng ta sẽ bắt tay vào làm 1 ứng dụng thực tế là "đọc cường độ ánh sáng của môi trường" để hiểu rõ hơn về cách hoạt động của nó.

Giới thiệu module cảm biến ánh sáng

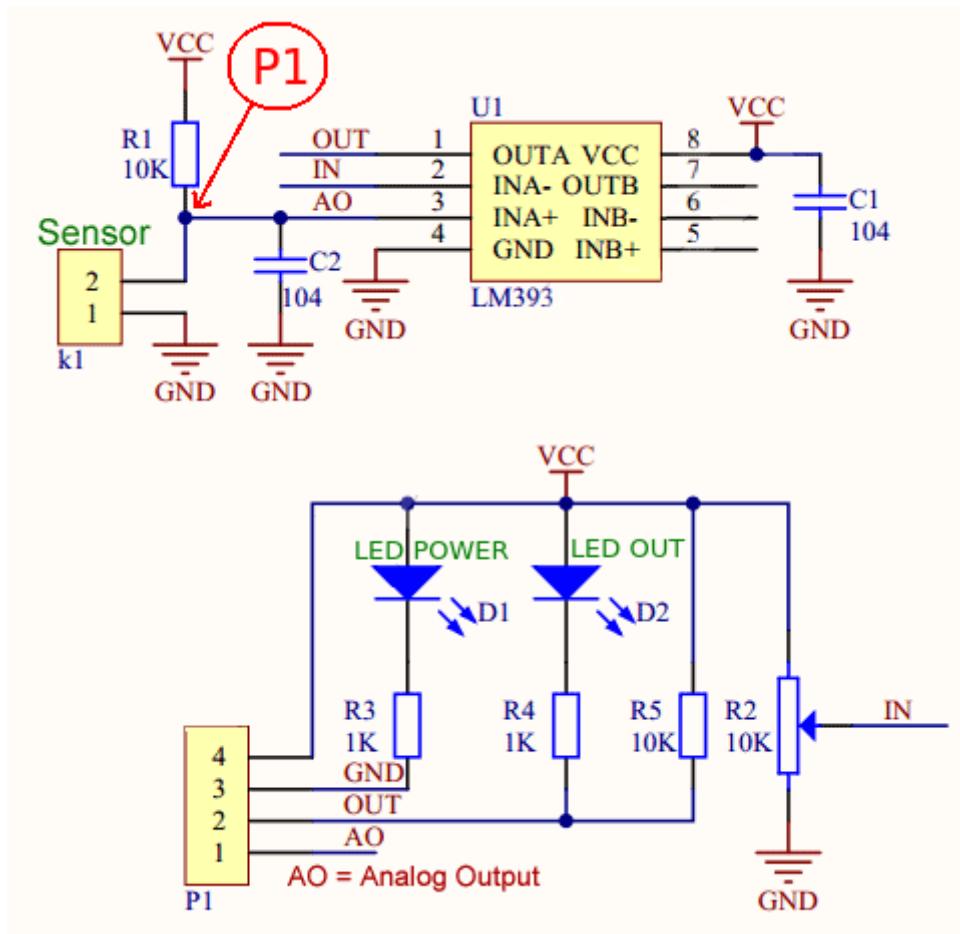
Hình ảnh



Hình 56. Hình ảnh của module cảm biến cường độ ánh sáng.

Nguyên lý hoạt động

Chúng ta cùng tìm hiểu sơ đồ nguyên lý của module cảm biến ánh sáng để hiểu rõ hơn về cách nó hoạt động.



Hình 57. Hình ảnh sơ đồ nguyên lý của module cảm biến cường độ ánh sáng.

Module cảm biến ánh sáng hoạt động dựa vào sự thay đổi của cường độ của ánh sáng môi trường. Nếu trời tối thì cường độ ánh sáng yếu, điện trở của cảm biến ánh sáng trên module tăng lên (cường độ ánh sáng 0 Lux tương ứng với giá trị điện trở khoảng 1MΩ) và ngược lại, nếu cường độ ánh sáng tăng thì điện trở của cảm biến ánh sáng sẽ giảm (cường độ ánh sáng 10 Lux tương ứng với giá trị điện trở trong khoảng 8 - 20KΩ). Điện trở của sensor thay đổi dẫn đến giá trị điện áp tại điểm P1 sẽ thay đổi theo. Điều này kéo theo 2 sự thay đổi :

- Thay đổi giá trị điện áp tại chân AO (Analog output), bằng cách kết nối các chân analog của vi điều khiển đến điểm này, ta có thể đọc giá trị điện áp analog và điều khiển được các thiết bị dựa theo cường độ ánh sáng.
- Thay đổi giá trị điện áp tại chân INA+ của IC LM393, IC này là 1 op-amp có chức năng so sánh điện áp giữa 2 chân INA+ và INA-, chân OUTA là kết quả so sánh của 2 chân INA+ và INA-, nó có 2 trạng thái là mức cao (HIGH) và mức thấp (LOW). Biến trở được nối với chân INA- giúp người dùng có thể điều chỉnh điện áp tại chân INA- nhằm cài đặt trạng thái ngõ ra cho chân OUTA.

Trên module có thêm 2 LED, LED PWR để báo trạng nguồn (có nguồn điện cung cấp cho module, LED này sẽ sáng) và LED OUT để báo chân trạng thái của chân OUTA.

Điều khiển LED RGB theo cường độ ánh sáng của môi trường

Yêu cầu

Nếu trời tối thì LED sáng màu đỏ (Red), trời sáng vừa thì LED sáng màu xanh lá (Green), trời sáng mạnh LED sẽ sáng màu xanh dương (Blue).

Chuẩn bị

- [Module cảm biến cường độ ánh sáng](#)
- [Module LED RGB](#)
- [Breadboard](#)
- [Dây cắm male-female](#)
- [Board IoT Maker UnoX](#)

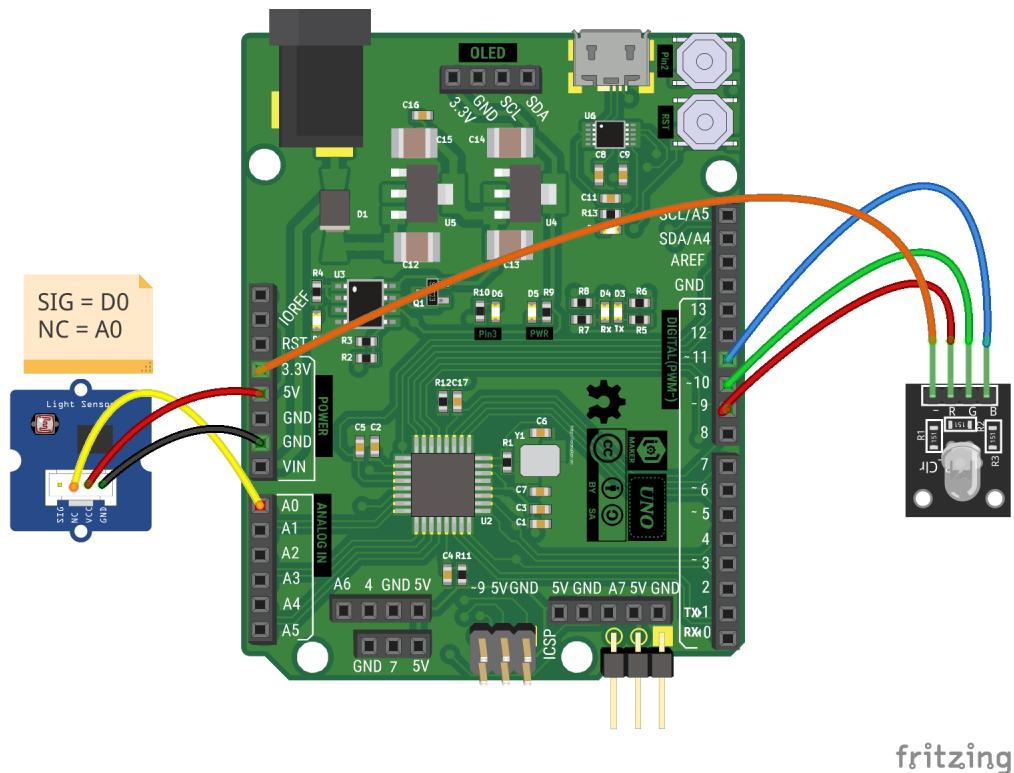
Đầu nối

Bảng 5. Bảng đấu nối chân của module LED RGB và board IoT Maker UnoX

Số thứ tự	Chân trên module LED RGB	Chân trên board IoT Maker UnoX
1	R	9
2	G	10
3	B	11
4	V/G	3.3V hoặc GND

Bảng 6. Bảng đấu nối chân của module cảm biến ánh sáng và board IoT Maker UnoX.

Số thứ tự	Chân trên cảm biến ánh sáng	Chân trên board IoT Maker UnoX
1	A0	A0
2	VCC	5V
3	GND	GND



Hình 58. Hình ảnh kết nối module LED RGB và cảm biến ánh sáng với board IoT Maker UnoX

[Source code](#)

```

#define pinLedRed    9      // Chân kết nối với pin R của module LED-RGB
#define pinLedGreen  10     // Chân kết nối với pin G của module LED-RGB
#define pinLedBlue   11     // Chân kết nối với pin B của module LED-RGB

#define pinAnaLight A0     // Chân kết nối với A0 của module cảm biến ánh sáng

void setup()
{
    // Cài đặt hướng cho các chân
    pinMode(pinLedRed, OUTPUT);
    pinMode(pinLedGreen, OUTPUT);
    pinMode(pinLedBlue, OUTPUT);
    // LED RGB OFF
    digitalWrite(pinLedRed, LOW);
    digitalWrite(pinLedGreen, LOW);
    digitalWrite(pinLedBlue, LOW);

    pinMode(pinAnaLight, INPUT);
    // Khởi tạo serial
    Serial.begin(115200);
    Serial.println("Initial serial");
}

void loop()
{
    unsigned int AnalogValue;
    AnalogValue = analogRead(pinAnaLight); // Đọc trạng thái chân A0 của module cảm biến ánh sáng
    Serial.println(AnalogValue);           // In ra giá trị điện áp Analog
    if (AnalogValue < 350) {

        digitalWrite(pinLedRed, HIGH);
        digitalWrite(pinLedGreen, LOW);
        digitalWrite(pinLedBlue, LOW);

        Serial.println("The light intensity: HIGH");
    } else if (AnalogValue > 350 && AnalogValue < 800) {

        digitalWrite(pinLedRed, LOW);
        digitalWrite(pinLedGreen, HIGH);
        digitalWrite(pinLedBlue, LOW);
        Serial.println("The light intensity: MEDIUM");
    } else if (AnalogValue > 800) {

        digitalWrite(pinLedRed, LOW);
        digitalWrite(pinLedGreen, LOW);
        digitalWrite(pinLedBlue, HIGH);
        Serial.println("The light intensity: LOW");
    }

    delay(100);
}

```

Giải thích source code

- Lệnh `Serial.begin(115200)` nhằm khởi tạo Serial nhằm hiển thị giá trị Analog đọc được ra màn hình Serial của Arduino, `Serial.println("Initial serial")` in ra chữ "Initial serial" và xuống dòng.
- Hàm `analogRead(pinAnaLight)` sẽ đọc điện áp từ chân A0 của module cảm biến ánh sáng, để đưa vào bộ chuyển đổi ADC của vi điều khiển Atmega328P. Bộ ADC này là bộ ADC 10 bit, tương ứng với giá trị Analog nhỏ nhất là 0 và lớn nhất là $2^{10} = 1024$.
- Các câu lệnh điều kiện `if- else if` ở phía dưới nhằm setup giá trị cường độ ánh sáng để điều

khiển LED RGB sáng các màu đỏ, xanh lá, và xanh dương. Cụ thể :

- Nếu giá trị Analog đọc được nhỏ hơn 350 (biểu thị cường độ ánh sáng mạnh), LED RGB sáng màu đỏ.
- Nếu giá trị Analog đọc được lớn hơn 350 và nhỏ hơn 800 (biểu thị cường độ ánh sáng vừa), LED RGB sáng màu xanh lá.
- Nếu giá trị Analog đọc được lớn hơn 800 (biểu thị cường độ ánh sáng yếu) LED RGB sáng màu xanh dương.

Tổng kết

Qua chương này, chúng ta đã có những hiểu biết cơ bản về các khái niệm, linh kiện điện tử, những ví dụ mẫu về điều khiển đèn LED, nút nhấn, đọc dữ liệu Analog, PWM. Chúng ta cũng đã hiểu về các chức năng cơ bản của vi điều khiển ATmega328 cũng như cách sử dụng board IoT Maker UnoX. Đây chỉ là những dự án mẫu và rất cơ bản, những thứ đáng học hỏi đang chờ chúng ta ở các phần tiếp theo của cuốn sách này.

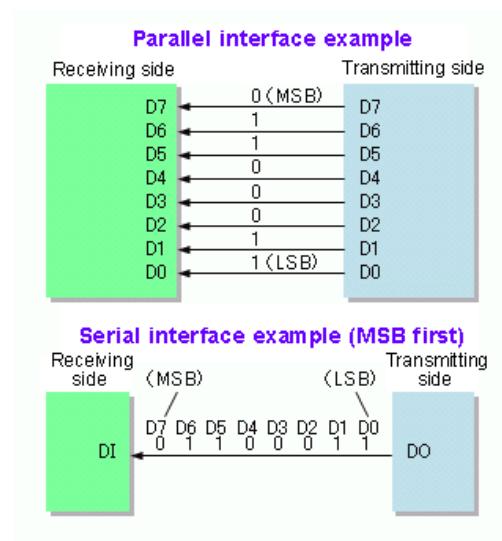


Chúng ta có thể tham khảo thêm các ví dụ trong Arduino IDE tại mục [File -> Examples](#).

Truyền thông nối tiếp

Truyền thông nối tiếp là một khái niệm quan trọng giúp chúng ta có thể dễ dàng nắm bắt những nội dung về một số chuẩn giao tiếp thông dụng của Arduino như I2C, SPI, 1-Wire, Serial... sẽ được giới thiệu ở các phần tiếp theo.

Trong thực tế, khi xây dựng một ứng dụng sử dụng Arduino, giả sử chúng ta cần trao đổi thông tin giữa 2 thiết bị (giữa máy tính với board Arduino hoặc giữa các board Arduino với nhau hay có thể là giữa Arduino với các module,...), thông tin trao đổi gồm có 8 bit, vậy đâu là giải pháp cho vấn đề này? Có lẽ cách dễ dàng tiếp cận nhất đó là sử dụng 8 dây data, ứng với mỗi dây data, mỗi bit sẽ được truyền từ bộ phát đến bộ thu, đây được gọi là phương pháp truyền song song (Parallel Communication), phương pháp này có ưu điểm tốc độ truyền tải nhanh, dễ dàng cài đặt, tuy nhiên nhược điểm lớn nhất là số lượng đường truyền cần phải lớn, thật khó chấp nhận đối với các dòng vi điều khiển có số lượng chân I/O là hạn chế. Do đó truyền thông nối tiếp (Serial Communication) được ra đời để giải quyết vấn đề này. Cụ thể, để thực hiện việc truyền tải dữ liệu, ta chỉ cần sử dụng 1 (hoặc một số) đường truyền, và dữ liệu cần trao đổi sẽ được truyền lần lượt từng bit theo đường truyền này. Do đó, trong các ứng dụng thực tế, ta thường lựa chọn phương pháp truyền thông nối tiếp thay vì truyền song song dù cho tốc độ truyền thấp hơn, nhưng làm tiết kiệm số lượng đường truyền và chân điều khiển của các loại vi điều khiển được sử dụng.



Hình 59. Giao tiếp nối tiếp và giao tiếp song song (Nguồn Wikipedia)

Ngoài ra, với phương pháp truyền thông nối tiếp, để đảm bảo tính chính xác của thông tin truyền nhận giữa bộ truyền và bộ nhận cần có những tiêu chuẩn cụ thể. Một số giao thức như I2C, SPI,... sử dụng một đường truyền phát xung (CLOCK) có chức năng đồng bộ quá trình trao đổi thông tin. Ví dụ, thiết bị thứ nhất muốn truyền một bit data, thiết bị này sử dụng thêm một dây tín hiệu chuyển từ mức thấp lên mức cao để thông báo cho thiết bị thu biết đang có dữ liệu gửi đến. Đây là một cách giúp giảm thiểu rủi ro sai lệch thông tin trong quá trình giao tiếp, được gọi là phương pháp **truyền thông đồng bộ (Synchronous)**. Khác với truyền thông đồng bộ, **truyền thông không đồng bộ**

(**Asynchronous**) có những cách chuẩn hóa riêng giữa các thiết bị giao tiếp với nhau và sẽ không cần dùng đến dây tín hiệu phát xung. Ứng với cách này, giả sử ta quy định cho bộ truyền và bộ nhận rằng cứ mỗi 1 micro giây thì bộ truyền sẽ gửi 1 bit dữ liệu, do đó cứ mỗi 1 micro giây bộ nhận sẽ chỉ cần kiểm tra và tiếp nhận thông tin, sau đó tổng hợp để thu được dữ liệu có nghĩa.

Giao tiếp Serial

Trong phần tiếp theo chúng ta sẽ cùng tìm hiểu về chuẩn giao tiếp Serial (Serial protocol) một chuẩn giao tiếp đại diện điển hình nhất của phương pháp truyền thông nối tiếp không đồng bộ (Asynchronous Serial). Nội dung phần này sẽ lần lượt trình bày các khái niệm cơ bản xoay quanh giao tiếp Serial cũng như những ví dụ thực tế từ chuẩn giao tiếp này.

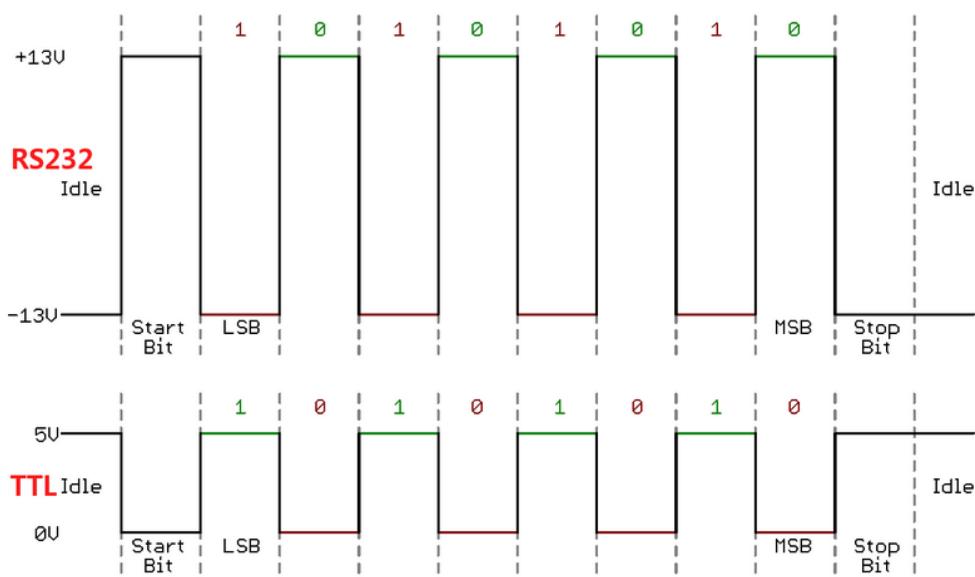
Những khái niệm cơ bản

Chuẩn giao tiếp Serial là gì?

Serial là chuẩn giao tiếp truyền thông nối tiếp không đồng bộ (được trình bày trong phần trước), sử dụng **UART** (Universal Asynchronous Receiver-Transmitter, một bộ phận phần cứng được tích hợp bên trong chip ATmega328 của board UnoX) kết hợp với mức logic TTL (mức điện áp cao tương ứng với 5V hoặc 3.3V và mức điện áp thấp tương ứng với 0V). Trong thực tế, chúng ta vẫn có thể kết hợp UART và các mức logic khác để tạo ra một số chuẩn truyền khác chẳng hạn RS232, nhưng trong khuôn khổ quyển sách này chúng ta chỉ khảo sát mức logic TTL là chính.



UART là tên gọi để chỉ một bộ phận phần cứng dùng để chuyển đổi thông tin từ thanh ghi chứa nội dung cần trao đổi ra bên ngoài (thông thường có 8 bits) thành 2 dây tín hiệu Serial, phục vụ cho chuẩn giao tiếp Serial chứ **không phải là tên gọi của một chuẩn giao tiếp**.



Hình 60. RS232 và TTL

Như đã giới thiệu từ phần trước, khi sử dụng chuẩn giao tiếp **Serial**, để trao đổi thông tin một cách hiệu quả, đòi hỏi các thiết bị giao tiếp với nhau cần phải thống nhất với nhau một số quy cách riêng như tốc độ baud (baudrate), khung truyền (frame), start bit, stop bits,... và sau đây chúng ta sẽ lần

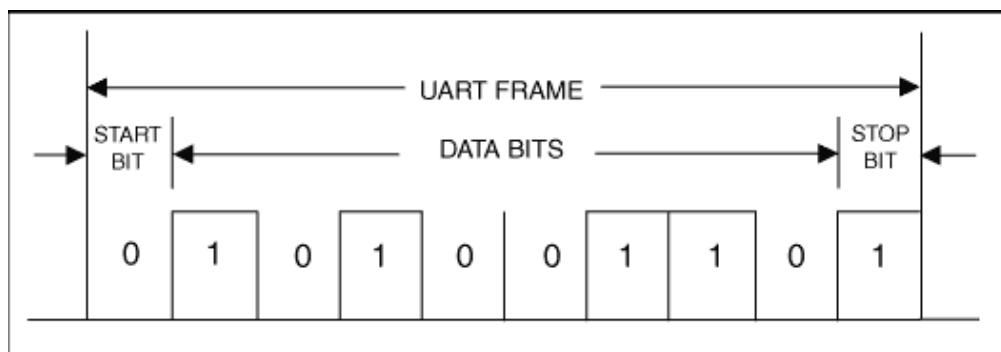
lượt tìm hiểu:

Baudrate

Như trong ví dụ về việc truyền 1 bit data sau mỗi 1 micro giây, ta nhận thấy rằng để việc giao tiếp diễn ra thành công, giữa 2 thiết bị cần có những thống nhất rõ ràng về khoảng thời gian truyền cho mỗi bit dữ liệu, hay nói cách khác tốc độ truyền cần phải được thống nhất với nhau. Tốc độ này được gọi là tốc độ baud (baudrate), được định nghĩa là số bit truyền trong mỗi giây. Ví dụ, nếu tốc độ baud được sử dụng là 9600, có nghĩa thời gian truyền cho 1 bit là $1/9600 = 104.167$ micro giây.

Khung truyền (frame)

Bên cạnh tốc độ baud, khung truyền đối với chuẩn giao tiếp Serial cũng hết sức quan trọng để đảm bảo việc truyền nhận được diễn ra chính xác. Khung truyền quy định về số lượng bit truyền trong mỗi lần truyền, bao gồm start bit, các bit data, stop bits, ngoài ra còn có thể có parity bit (kiểm tra lỗi dữ liệu trong quá trình truyền nhận).



Hình 61. Ví dụ về một khung truyền

Hình ảnh trên là một ví dụ về khung truyền, bao gồm 1 bit start, 8 bits data và cuối cùng là 1 bit stop.

Start bit

Đây là bit đầu tiên được truyền vào khung truyền, có chức năng thông báo cho thiết bị nhận biết đang có một chuỗi dữ liệu sắp truyền đến. Đối với các dòng AVR nói chung và UnoX nói riêng, ở trạng thái chưa có dữ liệu (Idle) đường truyền luôn kéo lên mức cao. Khi có dữ liệu mới, đường truyền được kéo xuống mức thấp, do đó start bit sẽ được quy định là mức 0.

Dữ liệu (data)

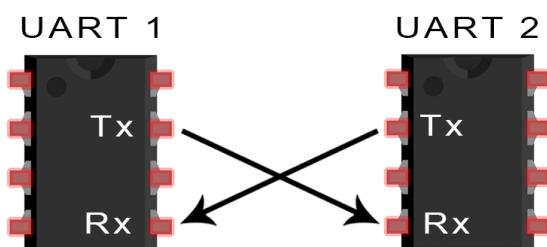
Dữ liệu cần truyền thông thường gồm 8 bits, tuy nhiên chúng ta hoàn toàn có thể tùy chỉnh số lượng bit data cho một gói tin, có thể là 5, 6, 7, 9,... Trong quá trình truyền, bit có trọng số thấp nhất (LSB) sẽ được truyền trước, và cuối cùng sẽ là bit có trọng số cao nhất (MSB).

Stop bits

Stop bits là một hoặc nhiều bit có chức năng thông báo cho thiết bị nhận biết rằng một gói dữ liệu đã được gởi xong. Đây là bit quan trọng, cần phải có trong một khung truyền. Giá trị của các stop bit luôn bằng mức Idle (mức nghỉ) và ngược với giá trị của start bit. Đối với các dòng vi điều khiển AVR các bit kết thúc này luôn là mức cao.

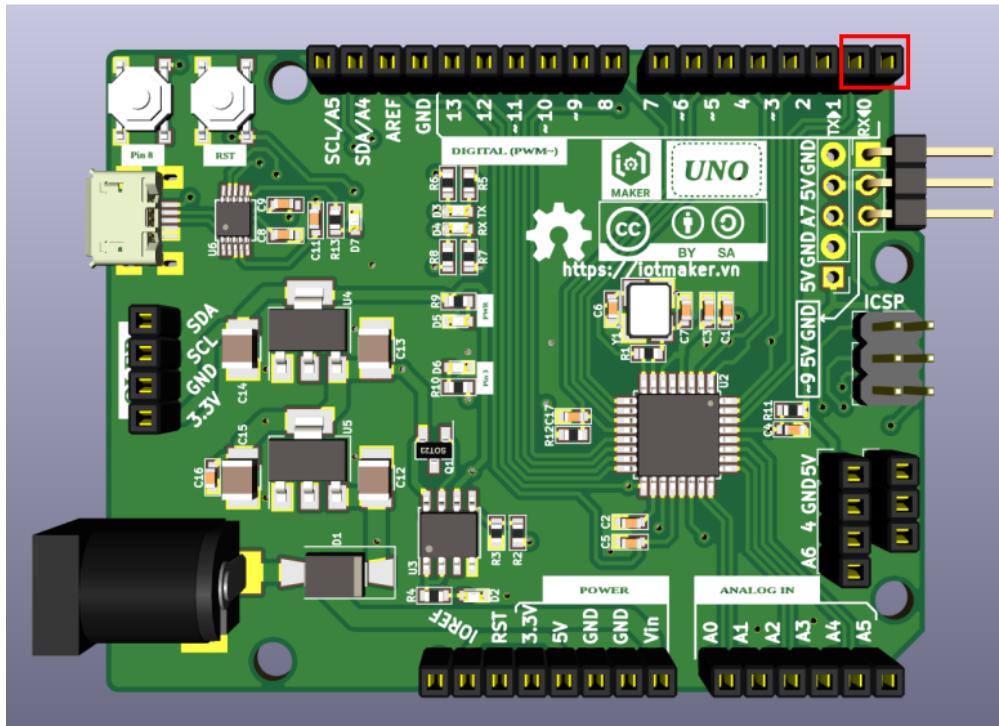
Sử dụng chuẩn giao tiếp Serial với board IoT Maker UnoX

Giao tiếp Serial giữa 2 thiết bị với nhau thông thường sẽ có 2 dây tín hiệu, đó là **TX** (Transmitter) có chức năng truyền tải dữ liệu và **RX** (Receiver) có chức năng thu nhận dữ liệu. 2 dây tín hiệu này được kết nối chéo nhau giữa 2 thiết bị. Cụ thể, TX của thiết bị 1 kết nối với RX của thiết bị 2 và RX của thiết bị 1 kết nối với TX của thiết bị 2. Điều đặc biệt của Serial đó là quá trình truyền và quá trình nhận dữ liệu được diễn ra hoàn toàn độc lập với nhau.



Hình 62. Giao tiếp 2 thiết bị sử dụng Serial UART

Hiện nay, các board Arduino đều được hỗ trợ sẵn 1 hoặc nhiều cổng giao tiếp Serial. Đối với board IoT Maker UnoX, cổng Serial được ra chân tại vị trí chân số 1 và chân số 0 trên board), đây được gọi là **Hardware Serial**, tức là giao tiếp Serial dựa trên phần cứng (có sử dụng UART). Ngoài ra, đối với các vi điều khiển không hỗ trợ UART hoặc trong trường hợp muốn mở rộng nhiều cổng Serial để giao tiếp với nhiều thiết bị có hỗ trợ giao tiếp Serial, chúng ta hoàn toàn có thể giả lập giao tiếp Serial bằng thư viện cho Arduino có tên là SoftwareSerial, chúng ta sẽ tìm hiểu thư viện này ở phần sau.



Hình 63. Vị trí chân giao tiếp UART trên board IoT Maker UnoX

Ứng dụng

Giao tiếp Serial của board IoT Maker UnoX với máy tính cá nhân

Khi kết nối board IoT Maker UnoX với máy tính bằng cable USB, thông qua chip USB-to-Serial CH430E trên board, cổng USB của máy tính sẽ được chuyển đổi thành tín hiệu Serial để kết nối với chân TX, RX của board, lúc này, ta có thể dễ dàng trao đổi dữ liệu giữa board và máy tính. Và đây cũng là cách để nạp chương trình từ máy tính xuống cho board IoT Maker UnoX. Sau đây, ta sẽ xét một số ví dụ đơn giản khi giao tiếp giữa IoT Maker UnoX và máy tính.

1. Lắng nghe dữ liệu từ board IoT Maker UnoX

1.1. Giới thiệu:

Việc máy tính lắng nghe dữ liệu từ board IoT Maker UnoX thông thường được sử dụng khi debug chương trình, qua đó chúng ta có thể quan sát và kiểm tra kết quả trả về từ board trong khi hoạt động thông qua cửa sổ Serial Monitor. Một số hàm cần lưu ý khi sử dụng chức năng này:

- `Serial.begin(baudrate);`
- `Serial.print(message);`
- `Serial.println(message);`

Trong đó:

- **baudrate** như đã giới thiệu đó là tốc độ baud, các tốc độ baud thường dùng: 9600, 57600, 115200, 921600,... Lưu ý khi sử dụng: giữa 2 thiết bị giao tiếp với nhau cần được khai báo tốc độ baud như nhau để trao đổi dữ liệu được hiệu quả.
- **message** ở đây là thông tin mà Arduino muốn truyền qua Serial. Thông tin này có thể là các kiểu dữ liệu như String, int, byte, char, double,...
- **Serial.begin(baudrate)** có chức năng khai báo sử dụng Serial với tốc độ baud là **baudrate**.
- **Serial.print(message)** có chức năng truyền Serial với nội dung là **message** mà không có kí tự kết thúc dòng. Ta sử dụng hàm này khi cần ghi dữ liệu lên Serial Monitor trên cùng 1 dòng.
- **Serial.println(message)** có chức năng truyền Serial với nội dung là **message** có kí tự kết thúc dòng. Ta sử dụng hàm này khi cần ghi cần kết thúc dòng nội dung ghi trên Serial Monitor.

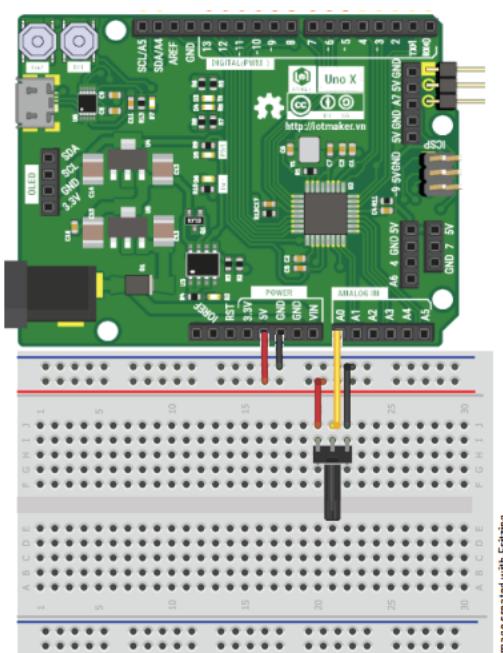
1.2. Ví dụ:

Đọc giá trị từ biến trỏ, xuất kết quả ra Serial Monitor.

Chuẩn bị:

- Board IoT Maker UnoX
- 1 biến trỏ 1KΩ.
- Breadboard
- Dây cắm male-male

Kết nối:



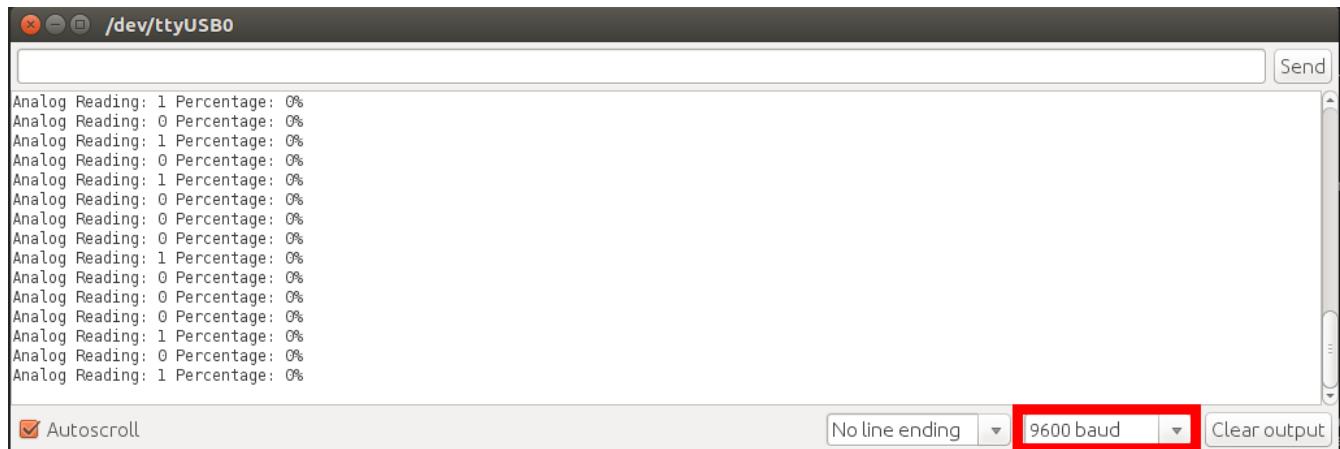
Hình 64. Kết nối Arduino với biến trỏ

Source code:

```
//Chương trình đọc giá trị từ chiết áp và gửi dữ liệu đến máy tính bằng Serial
const int POT=0; //Sử dụng chân A0 trên board để lấy giá trị từ chiết áp
void setup()
{
    Serial.begin(9600); //Khai báo sử dụng Serial với tốc độ baud = 9600
}
void loop()
{
    int val = analogRead(POT); //Tiến hành đọc giá trị chiết áp
    int per = map(val, 0, 1023, 0, 100); //Chuyển đổi sang phần trăm
    Serial.print("Analog Reading: ");
    Serial.print(val); //In giá trị thô đọc được
    Serial.print(" Percentage: ");
    Serial.print(per); //In giá trị đã chuyển đổi sang phần trăm
    Serial.println("%"); //In kí tự '%' và kết thúc dòng thông tin
    delay(1000); //Đợi 1s sau đó tiếp tục lặp
}
```

Xem kết quả:

Chọn **Tools → Serial Monitor** để xem kết quả từ Serial Monitor. Lưu ý: chọn baudrate đúng với giá trị đã khai báo.



Hình 65. Chọn baudrate trên Serial Monitor để xem kết quả

2. Gửi dữ liệu từ máy tính đến board UnoX qua Serial Monitor

2.1. Giới thiệu

Ta thường sử dụng Serial Monitor của Arduino IDE để gửi lệnh đến board IoT Maker UnoX. Một số hàm cần chú ý:

- **Serial.available()**: Trả về số lượng kí tự hiện tại đang lưu trữ trong bộ nhớ mà Arduino đã nhận được. Khi giá trị này khác 0, ta sẽ tiến hành đọc dữ liệu được lưu trữ.
- **Serial.read()**: Đọc kí tự trong bộ lưu trữ và tự động chuyển đến kí tự tiếp theo (nếu có).

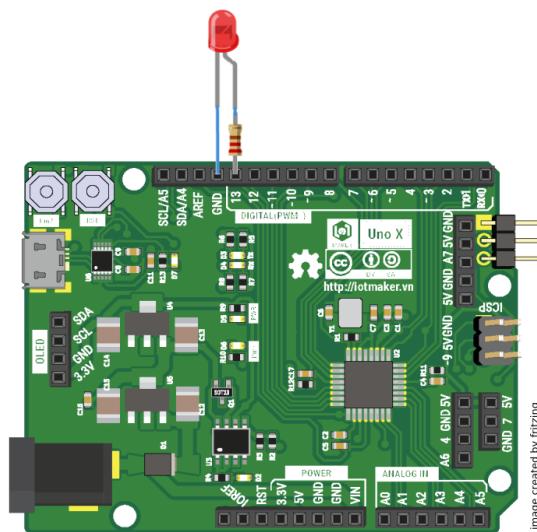
2.2. Ví dụ

Gửi lệnh điều khiển LED từ máy tính xuống board IoT Maker UnoX bằng Serial.

Chuẩn bị:

- Board IoT Maker UnoX
 - 1 LED đơn.
 - 1 điện trở $1\text{K}\Omega$.

Kết nối:



Hình 66. Kết nối LED với board IoT Maker UnoX

Source Code:

```
//Chương trình sử dụng Serial Monitor để gửi lệnh điều khiển đèn LED

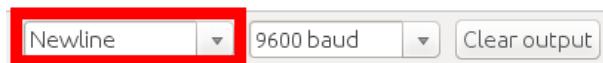
const int LED = 13; //Sử dụng chân 13 trên board để điều khiển đèn LED
String data = ""; //Khai báo biến data lưu lệnh mà người dùng nhập
void setup()
{
    pinMode(LED, OUTPUT);
    digitalWrite(LED, LOW); // Ban đầu cho đèn LED tắt
    Serial.begin(9600); // Khai báo sử dụng Serial với tốc độ baud = 9600
}
void loop()
{
    if (Serial.available() > 0)
    {
        char c = Serial.read(); //Đọc từng kí tự
        // Kiểm tra khi kết thúc câu lệnh
        if (c == '\n'){
            Serial.print("Send to Arduino: ");
            Serial.println(data);
            if (data == "on"){
                digitalWrite(LED, HIGH);
            } else
            if (data == "off"){
                digitalWrite(LED, LOW);
            }
            data = "";
        } else
            data = data + c;
        //Print byte of data
    }
}
```

Giải thích code

Chương trình trên sử dụng hàm `Serial.available()` để kiểm tra lệnh được gửi đến board. Bất cứ khi nào giá trị `Serial.available()` khác 0, ta sẽ dùng biến `c` kiểu `char` đọc từng kí tự được gửi đến, nếu `c` không phải là kí tự kết thúc dòng `\n` ta sẽ ghép từng kí tự `c` này vào biến `data` (`data = data + c`). Và nếu gặp kí tự kết thúc chuỗi (`c == '\n'`), tức là đã kết thúc 1 lệnh mà người dùng nhập, ta sẽ tiến hành kiểm tra xem lệnh đó là gì, nếu là `"on"` thì board IoT Maker UnoX sẽ `on` LED trên board (`digitalWrite(LED,HIGH)`) và nếu lệnh `"off"` thì điều khiển LED tắt (`digitalWrite(LED,LOW)`). Tất nhiên, nếu người dùng gửi lệnh khác `"on"` và `"off"` thì board IoT Maker UnoX sẽ không làm gì cả.



Khi người dùng hoàn thành việc nhập lệnh và click `send` hoặc nhấn phím `Enter` đồng nghĩa với chuỗi lệnh mà người dùng gửi đến có kết thúc là kí tự xuống dòng `\n` chương trình trên sử dụng đặc điểm này để nhận biết khi nào là kết thúc 1 chuỗi rồi tiến hành kiểm tra chuỗi đó có nội dung là gì. Để thực hiện việc này ta cần tùy chỉnh chế độ nhập từ Serial Monitor là **Newline**, ngoài ra cần phải chỉnh tốc độ baud là 9600 như ví dụ trước.



Hình 67. Tùy chỉnh chế độ nhập của Serial Monitor

Kết quả:



Hình 68. Nhập "on" hoặc "off" trên Serial Monitor

3. Giao tiếp Serial giữa 2 board Arduino qua Serial

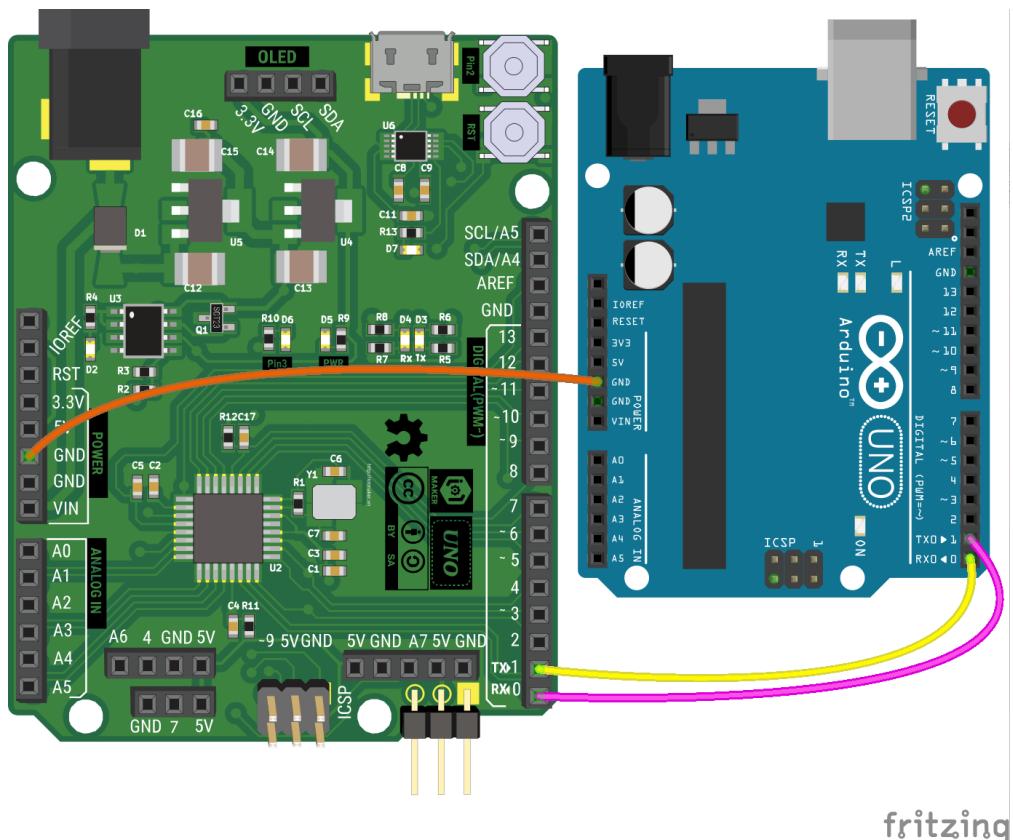
Ví dụ tiếp theo chúng ta sẽ thực hiện việc trao đổi thông tin giữa 2 board IoT Maker UnoX, board thứ nhất làm master và board còn lại là slave. Board master sẽ gửi lệnh điều khiển đèn LED xuống board Slave. Board Slave khi nhận được lệnh thì sẽ điều khiển đèn LED có sẵn trên board của chính nó.

3.1. Chuẩn bị

- 1 [Board IoT Maker UnoX](#) và 1 board Arduino bất kì.
- Một số dây header để kết nối các board.

3.2. Kết nối chân

Số thứ tự	IoT Maker UnoX (Slave)	Arduino khác(Master)
1	TXD	RXD
2	RXD	TXD
3	GND	GND



Hình 69. Hình ảnh kết nối board IoT Maker UnoX với Arduino khác

Lưu ý: Để hoạt động được, 2 board này cần được cấp nguồn, nguồn này có thể chung hoặc riêng nhưng GND của cả hai board cần được kết nối với nhau.

3.3. Source Code và giải thích

Master code:

```
// Giao tiếp 2 board Arduino (Master)
void setup() {
    Serial.begin(9600); // Khai báo sử dụng Serial với tốc độ baud là 9600
}

void loop() {
    //Cứ sau 1 giây ta sẽ on rồi sau đó gửi off đến Slave để điều khiển LED
    Serial.print("on\n"); // Lưu ý: cần kèm kí tự kết thúc dòng '\n' ở cuối lệnh cần gửi
    delay(1000);
    Serial.print("off\n");
    delay(1000);
}
```

Slave code:

```

//Giao tiếp 2 board Arduino (Slave)
const int LED = 3; //Sử dụng chân 3 trên board để điều khiển đèn LED
String data = ""; //Khai báo biến data lưu lệnh mà master gửi đến
void setup()
{
    pinMode(LED, OUTPUT); // Khai báo sử dụng pin số 3 làm OUTPUT
    digitalWrite(LED,LOW); // Ban đầu cho đèn LED tắt
    Serial.begin(9600); // Khai báo sử dụng Serial với tốc độ baud = 9600
}
void loop()
{
    if (Serial.available() > 0) // Nếu có lệnh gửi đến
    {
        char c = Serial.read(); //Đọc từng ký tự
        // Kiểm tra kết thúc câu lệnh
        if (c == '\n'){
            if (data == "on"){
                digitalWrite(LED, HIGH); // Nếu lệnh gửi đến là "on" thì bật đèn LED
            } else
            if (data == "off"){
                digitalWrite(LED, LOW); //Nếu lệnh gửi đến là "off" thì tắt đèn LED
            }
            data = "";
        } else // Nếu chưa phát hiện ký tự kết thúc câu lệnh thì tiếp tục ghi nhận.
        data = data + c;
    }
}

```

3.4. Kết quả

Nếu thành công, ta sẽ thấy đèn LED trên board IoT Maker UnoX chớp tắt theo chu kỳ 2 giây

3.5. Mở rộng từ ví dụ

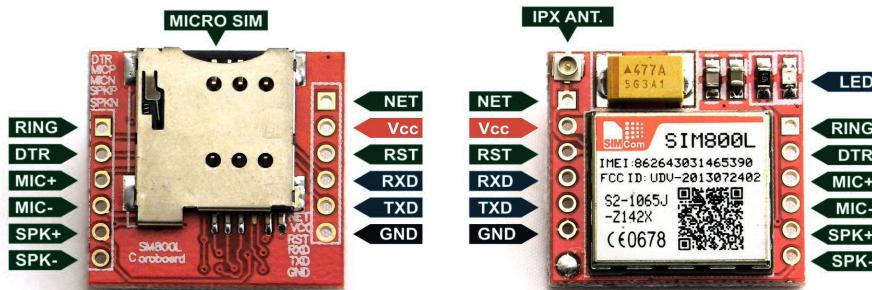
Qua ví dụ này, ta đã biết cách giao tiếp giữa 2 board Arduino bằng Serial, điều này phục vụ cho các dự án lớn khi 1 board Arduino là chưa đủ để quản lý hết tất cả các thiết bị, việc thêm 1 hay nhiều board Arduino để chia sẻ tác vụ là hết sức cần thiết. Có nhiều cách để trao đổi thông tin giữa các Arduino, nhưng dùng Serial là giải pháp giúp tiết kiệm số lượng đường truyền với thời gian truyền đáp ứng tốt.

4. Giao tiếp Serial giữa board IoT Maker UnoX và module Sim800

Ví dụ tiếp theo, chúng ta sẽ sử dụng Arduino cùng với thư viện SoftwareSerial để điều khiển module Sim800L nhắn tin đến điện thoại di động.

4.1. Module Sim800

Module Sim800L là một giải pháp hiệu quả cho các ứng dụng về GSM và GPRS (nhắn tin, gọi điện hoặc sử dụng dịch vụ mạng 2G). Nếu bạn đang tìm kiếm một loại module giúp bạn trong các dự án như phát hiện có đối tượng lạ xâm nhập thì cần gọi điện hoặc nhắn tin đến người chủ, điều khiển hệ thống tưới bằng tin nhắn điện thoại, thu thập dữ liệu gửi lên Server bằng dịch vụ GPRS,... thì Sim800L là giải pháp bạn cần cân nhắc đầu tiên. Sim800L được hỗ trợ giao tiếp Serial TTL do đó rất dễ sử dụng để giao tiếp với các vi điều khiển thông qua tập lệnh AT.



Hình 70. Module Sim800L GSM GPRS

Các thông số kỹ thuật của module Sim800L:

- Điện áp hoạt động : 3.7 - 4.2V
- Dòng khi ở chế độ chờ: 10 mA
- Dòng khi hoạt động: 100 mA đến 1A.
- Khe cắm SIM : MICROSIM
- Hỗ trợ 4 băng tần : GSM850MHz, EGSM900MHz, DSC1800Mhz, PCS1900MHz
- Có thể giao tiếp với vi điều khiển qua Serial TTL
- 1 LED báo tín hiệu.

4.2. Tập lệnh AT:

Một số module như Sim, GPS, ESP8266, bluetooth... được hỗ trợ giao tiếp Serial TTL đi kèm với đó là tập lệnh AT (AT commands), bằng những tập lệnh này thế giới bên ngoài có thể truy cập vào module, đưa ra những câu lệnh về thiết lập, điều khiển hoặc lấy thông tin,... Tương tự ví dụ thứ 2 ta sử dụng dụng máy tính để ra lệnh cho Arduino điều khiển đèn LED thông qua chuẩn giao tiếp Serial, tập lệnh AT cũng có chức năng giống vậy tuy nhiên cấu trúc các lệnh AT được chuẩn hóa theo cách riêng.

Ví dụ một số lệnh AT cơ bản đối với module Sim800L:

- AT: Kiểm tra đáp ứng của Module Sim, nếu trả về OK thì Module hoạt động.
- AT+IPR=[baud rate] : Cài đặt tốc độ giao tiếp dữ liệu với Module Sim, chỉ cài được các tốc độ sau: 0(auto), 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200.
- ATD[Số_điện_thoại]: Lệnh thực hiện cuộc gọi.
- AT+CNMI=2,2: Hiển thị nội dung tin nhắn ngay khi có tin nhắn đến.

Ngoài ra bạn có thể tham khảo thêm các câu lệnh khác tại: [SIM800 Series_AT Command Manual_V1.09.pdf](#) (keywords: AT commands for sim800)

4.3. Thư viện SoftwareSerial

Như đã giới thiệu, đối với một số vi điều khiển không được hỗ trợ Serial hoặc số cổng Serial được hỗ trợ là hạn chế như board IoT Maker UnoX thì việc mở rộng giao tiếp với nhiều thiết bị bằng Serial thật sự khó khăn, khi đó việc sử dụng thư viện giúp giả lập giao tiếp bằng Serial là hết sức cần thiết, và **SoftwareSerial** là thư viện tốt nhất hỗ trợ việc này. Theo đó, ta chỉ cần kết nối bằng 2 chân GPIO bất kỳ của Arduino rồi khai báo sử dụng 2 nút để giao tiếp Serial là ta có thể sử dụng Serial như cách truyền thống.

Khai báo sử dụng Serial ảo: **SoftwareSerial mySerial(RX,TX)**, trong đó TX, RX là các pin sử dụng cho giao tiếp Serial. Kết thúc việc khai báo này, ta có thể sử dụng đối tượng **mySerial** trong lập trình để đọc, ghi dữ liệu bằng Serial (Giống như biến Serial trong các ví dụ trước).

4.4. Kết nối chân

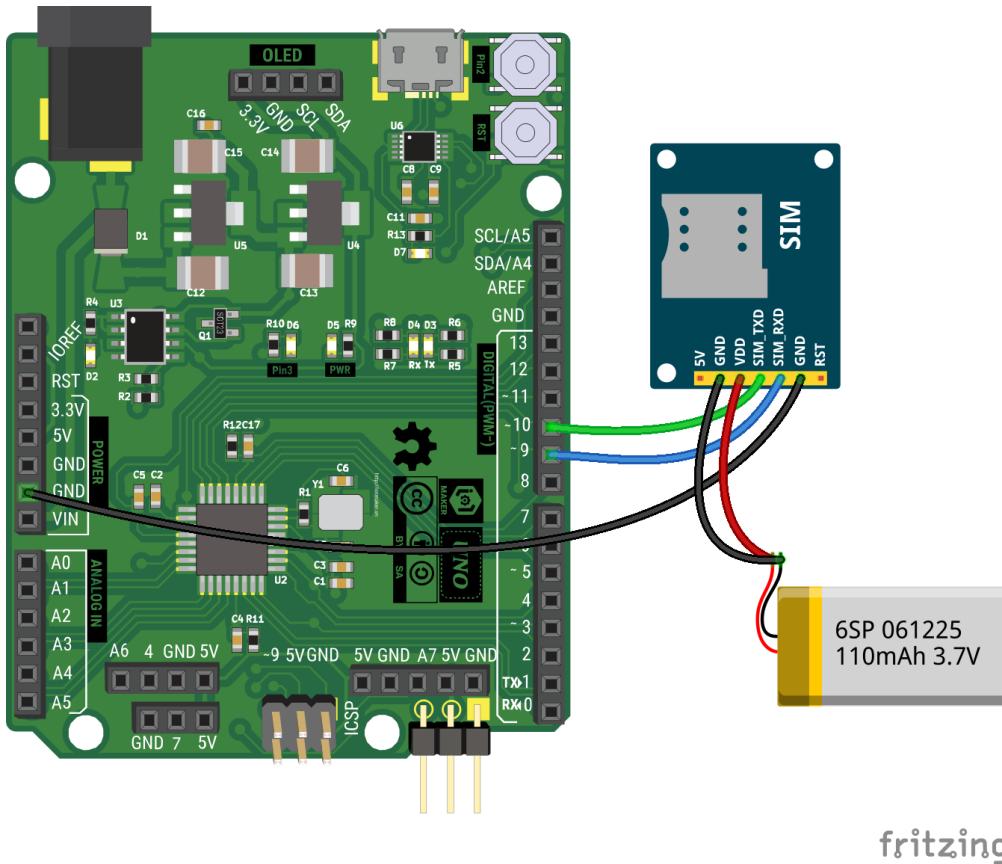
Trong ví dụ lần này, ta sẽ sử dụng chân 9 và 10 của board IoT Maker UnoX làm TX và RX để giao tiếp bằng Serial với module Sim800L. Vì module Sim800L chỉ có thể hoạt động ở mức điện áp từ 3.7V đến 4.2V do đó để cung cấp đúng mức điện áp cho module ta sử dụng pin 3.7V LiPo.



Hình 71. Pin 3.3V LiPo

Bảng 7. Bảng đấu nối

Số thứ tự	IoT Maker UnoX	Module Sim800L
1	GND	GND
2	9	RX
3	10	TX



fritzing

Hình 72. Hình ảnh kết nối board IoT Maker UnoX với Sim800L

Sau khi hoàn tất kết nối, ta cần lắp thẻ microSim vào khay sim của module.

4.5. Source Code và giải thích

```

#include <SoftwareSerial.h>           //Khai báo sử dụng thư viện SoftwareSerial

#define TX_PIN 9                      //Sử dụng chân số 9 làm TX
#define RX_PIN 10                     //Sử dụng chân số 10 làm RX

//Khởi tạo đối tượng serialSIM800 để giao tiếp Serial với sim800L
SoftwareSerial serialSIM800(RX_PIN, TX_PIN);

void setup() {
    //Khai báo sử dụng Serial để giao tiếp với máy tính thông qua Serial Monitor
    Serial.begin(9600);
    //Khai báo sử dụng serialSIM800 để giao tiếp với tốc độ baud là 9600
    serialSIM800.begin(9600);
    delay(1000);
    Serial.println("Setup Complete!");
    Serial.println("Sending SMS...");

    //Thiết lập định dạng tin nhắn là ASCII để có thể gửi tin nhắn dạng text
    serialSIM800.write("AT+CMGF=1\r\n");
    delay(1000);                      //Cứ mỗi lệnh AT ta cần có thời gian delay để Sim800 thực thi

    //Cài đặt số điện thoại muốn gửi tin nhắn đến, và nội dung tin nhắn
    serialSIM800.write("AT+CMGS=\\"0162657XXXX\\r\\n");
    delay(1000);

    //Nội dung tin nhắn muốn gửi
    serialSIM800.write("Hello, this is IoT Maker UnoX");
    delay(1000);

    //Gửi kí tự có mã ASCII là 26 ($1A) để thông báo cho Sim800 là đã hoàn thành việc nhập nội dung tin nhắn
    serialSIM800.write((char)26);
    delay(1000);

    Serial.println("SMS Sent!");
}

void loop() {
}

```



Ta cần khai báo chính xác số điện thoại để Sim800 gửi đúng.

4.6. Kết quả

Kết quả ta sẽ nhận được tin nhắn có nội dung "Hello, this is IoT Maker UnoX" sau vài giây từ khi board IoT Maker UnoX khởi động.

4.7. Mở rộng

Qua ví dụ trên, chúng ta đã biết cách sử dụng thư viện SoftwareSerial để điều khiển module Sim800L gửi tin nhắn đến điện thoại di động, ngoài ra vẫn còn rất nhiều ứng dụng thú vị khác được sử dụng với Sim800L chẳng hạn như thực hiện cuộc gọi hay sử dụng dịch vụ GPRS,... Tuy nhiên đối với các ứng dụng phức tạp, để tiện cho sử dụng ta hay dùng thư viện đã module hóa các tác vụ thành các hàm phục vụ cho nhắn tin, gọi điện hay cập nhật dữ liệu qua GPRS,... nhưng nguyên lý chung vẫn là sử dụng tập lệnh AT để điều khiển Sim800L. Nếu nắm được bản chất này chúng ta hoàn toàn có thể sử dụng các module khác tương tự như GPS, ESP8266, bluetooth,... Chỉ cần tra cứu tập lệnh riêng của từng module là ta có thể áp dụng ngay chúng vào dự án.

Đối với thư viện SoftwareSerial, đây là một công cụ hữu ích giúp chúng ta có thể mở rộng để kết nối nhiều module hỗ trợ Serial với Arduino. Bạn đọc hoàn toàn có thể sử dụng thư viện này để giao tiếp 2 board Arduino như trong ví dụ thứ 3 mà không cần phải sử dụng Serial Hardware.

Tổng kết

Qua chương này, chúng ta đã cùng tìm hiểu về giao thức Serial cũng như những ứng dụng cơ bản xoay quanh chuẩn giao thức này, đó là tiền đề giúp chúng ta có thể xây dựng hệ thống, trong đó có nhiều thiết bị giao tiếp với nhau bằng Serial.

Giao tiếp I2C

Ngoài chuẩn truyền thông nối tiếp Serial, chúng ta còn có một chuẩn giao tiếp khác giữa các vi điều khiển, IC đó là chuẩn I2C.

Ở chương này, chúng ta sẽ tìm hiểu các nội dung sau:

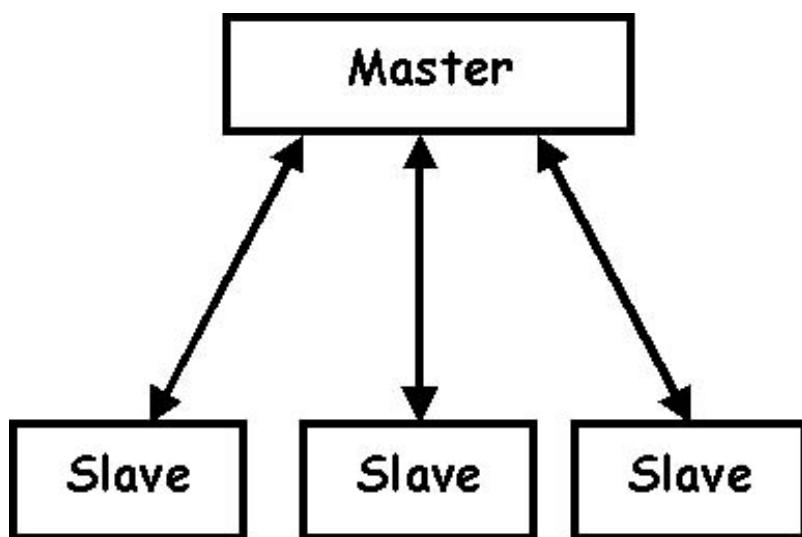
- Các khái niệm liên quan tới giao tiếp I2C như: mô hình master/slave, hoạt động cơ bản - truyền, nhận bit của giao thức I2C.
- Cách xác định địa chỉ của thiết bị trong giao tiếp I2C bằng chương trình, xác định địa chỉ với nhiều thiết bị khác nhau, những vấn đề xảy ra với nhiều thiết bị giống nhau trong truyền, nhận dữ liệu I2C.
- Sử dụng giao tiếp I2C đơn giản với LCD và OLED.
- Giao tiếp giữa 2 board IoT Maker UnoX với nhau thông qua chuẩn I2C.

Mô hình Master/slave

Master/slave là một mô hình giao tiếp mà một thiết bị có thể điều khiển ít nhất một thiết bị khác. Mô hình master/slave được dùng để chỉ các mô hình giao tiếp giữa các thiết bị điện tử, cơ khí và máy tính.

Trong mô hình master/slave, một thiết bị được chọn làm master (thiết bị chủ điều khiển các thiết bị khác), các thiết bị còn lại làm vai trò slave (là các thiết bị chịu sự điều khiển của master).

Ví dụ, trong một mô hình ta có một vi điều khiển muốn điều khiển, hoặc đọc dữ liệu từ các cảm biến, relay,... thì vi điều khiển đóng vai trò là một master và các cảm biến, relay sẽ đóng vai trò là slave.



Hình 73. Sơ đồ khái niệm cơ bản của mô hình master/slave

Giao tiếp I2C

Giới thiệu

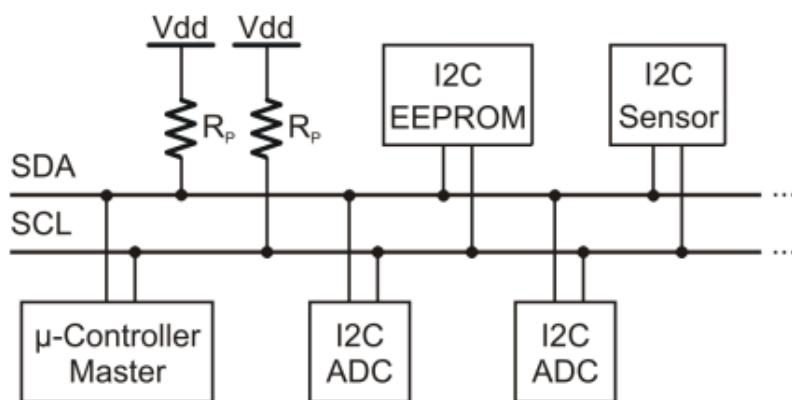
I2C là một chuẩn giao tiếp theo mô hình master/slave được phát triển vào năm 1982 bởi hãng Philips cho việc giao tiếp ngoại vi giữa các vi điều khiển, IC **trong khoảng cách ngắn**. Chuẩn giao tiếp I2C được sử dụng rất phổ biến trong các thiết bị điện tử bởi đặc tính dễ thực hiện với giao tiếp giữa một hoặc nhiều thiết bị làm master với một hoặc nhiều thiết bị làm slave. Các hãng sản xuất IC ngày nay đều hỗ trợ giao tiếp I2C trên các IC có ứng dụng cần giao tiếp với các IC hay các vi điều khiển khác.

Giao tiếp I2C chỉ sử dụng 2 dây cho việc giao tiếp giữa 128 thiết bị với việc định địa chỉ 7 bit và 1024 thiết bị với việc định địa chỉ 10 bit. Khi đó, mỗi thiết bị trong mô hình master/slave sẽ có một địa chỉ cố định và duy nhất và master sẽ lựa chọn thiết bị nào cần giao tiếp.

Hoạt động

I2C sử dụng 2 dây giao tiếp là SCL và SDA. Dây SCL (viết tắt của Serial Clock Data) là dây truyền xung clock phát từ thiết bị master đồng bộ với việc truyền dữ liệu. Dây SDA (Serial Data) là dây truyền dữ liệu.

Mạch vật lý I2C là mạch cực thu hở, do đó để mạng I2C có thể hoạt động được, cần tối thiểu 2 cặp điện trở pull-up như trên hình, thông thường các giá trị điện trở 4k7 hoặc 1k2 được sử dụng, tùy thuộc vào tốc độ truyền và khoảng cách truyền.

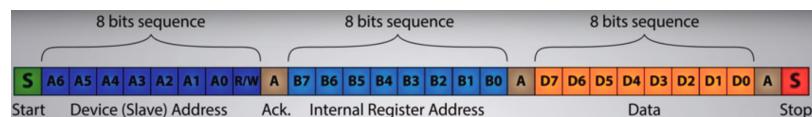


Hình 74. Mô hình mạng I2C

Truyền nhận bit trong I2C

Các dữ liệu được truyền trong I2C dạng các chuỗi 8 bit. Sau khi bit **Start** đầu tiên được truyền, 8 bit tiếp theo chứa thông tin về địa chỉ của thiết bị sẽ được truyền tiếp. Sau đó, một bit **ACK** sẽ được

truyền để xác nhận việc truyền bit thành công. Sau khi truyền bit **ACK**, 8 bit tiếp theo chứa thông tin địa chỉ thanh ghi nội của thiết bị slave sẽ được truyền. Sau khi hoàn tất việc định địa chỉ với 8 bit này, một bit **ACK** nữa sẽ được truyền để xác nhận. Sau đó, 8 bit Data sẽ được truyền tiếp theo. Cuối cùng, quá trình truyền kết thúc với 1 bit **ACK** và 1 bit **Stop** xác nhận việc truyền dữ liệu kết thúc.

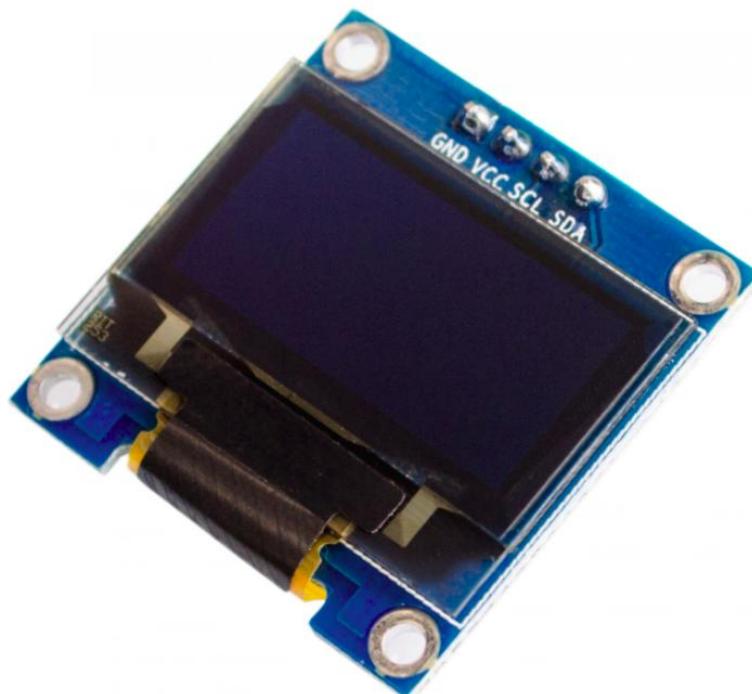


Hình 75. Các bit truyền, nhận trong giao tiếp I2C.

Để hiểu rõ hơn về quá trình truyền nhận dữ liệu và bit trong I2C, tham khảo: [Understanding the I2C Bus, How I2C Communication Works and How To Use It with Arduino](#).

Sử dụng giao thức I2C

Với các vi điều khiển, IC, cảm biến,... có hỗ trợ chuẩn giao tiếp I2C thì sẽ có 2 chân có tên SCL và SDA, tương ứng với 2 dây SCL, SDA cho giao tiếp I2C. Với board IoT Maker UnoX, chân A4 là chân SDA và chân A5 là chân SCL, 2 chân A4, A5 này cũng được nối tới 2 chân SDA, SCL tương ứng ở header sử dụng với OLED của board IoT Maker UnoX. Với các IC và vi điều khiển được sản xuất ngày nay đều có hỗ trợ điện trở nội bộ kéo lên do đó trong việc kết nối dây không cần thêm điện trở nội bộ kéo lên. Với các module hỗ trợ I2C chỉ có 1 tính năng như [OLED SSD1306](#), [LCD 1602](#) để hiển thị, [Cảm biến AM2315](#) để đọc nhiệt độ, độ ẩm của môi trường,... thường hoạt động ở chế độ slave và có 4 chân: SDA, SCL, GND và VCC.



Hình 76. Hình ảnh module sử dụng giao tiếp I2C (OLED-SSD1306)

Viết chương trình cho I2C

Chương trình cho giao tiếp I2C giữa cảm biến và vi điều khiển thường cần thư viện thích hợp cho từng loại vi điều khiển. Các cảm biến, thiết bị,... sẽ đọc dữ liệu từ môi trường và gửi về cho vi điều khiển cũng như vi điều khiển sẽ điều khiển các thiết bị thông qua giao tiếp I2C. Nhiều loại cảm biến, module cần dùng thêm thư viện riêng cho việc đọc, hiển thị dữ liệu,...

Một số chương trình có yêu cầu phải khai báo địa chỉ của thiết bị slave trong giao tiếp I2C thì việc đọc, hiển thị dữ liệu mới thực hiện được. Khi đó, nảy sinh vấn đề phải xác định được địa chỉ của thiết bị trong giao tiếp I2C. Địa chỉ này thường được cung cấp trong datasheet của thiết bị, hoặc có thể xác định thông qua các chương trình đọc địa chỉ.

Để sử dụng thư viện I2C trong Arduino, người dùng cần gọi thư viện `<Wire.h>` tích hợp sẵn trên trình biên dịch Arduino IDE.

Xác định địa chỉ của thiết bị trong giao tiếp I2C

Như đã trình bày ở phần giới thiệu, mỗi thiết bị trong giao tiếp I2C sẽ có một địa chỉ riêng. Các địa chỉ này sẽ được nhà sản xuất cung cấp trong datasheet của thiết bị. Ở phần này, chúng ta sẽ tìm hiểu về chương trình xác định địa chỉ của một thiết bị trong giao tiếp I2C. Một số linh kiện có hỗ trợ I2C nhưng người dùng khó tìm được datasheet chính thức từ nhà sản xuất thì việc dùng một chương trình ngắn để định địa chỉ I2C là một việc làm đơn giản và cần thiết.

Yêu cầu

Xác định địa chỉ các thiết bị trong giao tiếp I2C và phát hiện số thiết bị có giao tiếp I2C trong kết nối hiện tại.

Linh kiện cần dùng

- [Board IoT Maker UnoX](#)
- [Module LCD 20x4](#)

Source code

```

#include <Wire.h>

void setup()
{
    Wire.begin();          // Khởi tạo thư viện Wire
    Serial.begin(9600);
    while (!Serial);      // Đợi Serial Monitor hiển thị
    Serial.println("I2C Scanner");
}

void loop()
{
    byte error, address;
    int nDevices;

    Serial.println("Scanning...");

    nDevices = 0;
    // Định địa chỉ I2C có 7 bit ứng với 128 thiết bị, việc quét sẽ tiến hành
    // từ 1 tới 127 (1 thiết bị làm master nên chỉ còn 127)
    for (address = 1; address < 127; address++) {
        Wire.beginTransmission(address);
        error = Wire.endTransmission();
        if (error == 0) {
            Serial.print("I2C device found at address 0x");
            if (address < 16)
                Serial.print("0");
            Serial.print(address, HEX);
            Serial.println(" !");
            nDevices++;
            Serial.println("Device no. " + String(nDevices));
        } else if (error == 4) {
            Serial.print("Unknown error at address 0x");
            if (address < 16)
                Serial.print("0");
            Serial.println(address, HEX);
        }
    }

    if (nDevices == 0)
        Serial.println("No I2C devices found\n");
    else
        Serial.println("number of devices " + String(nDevices));

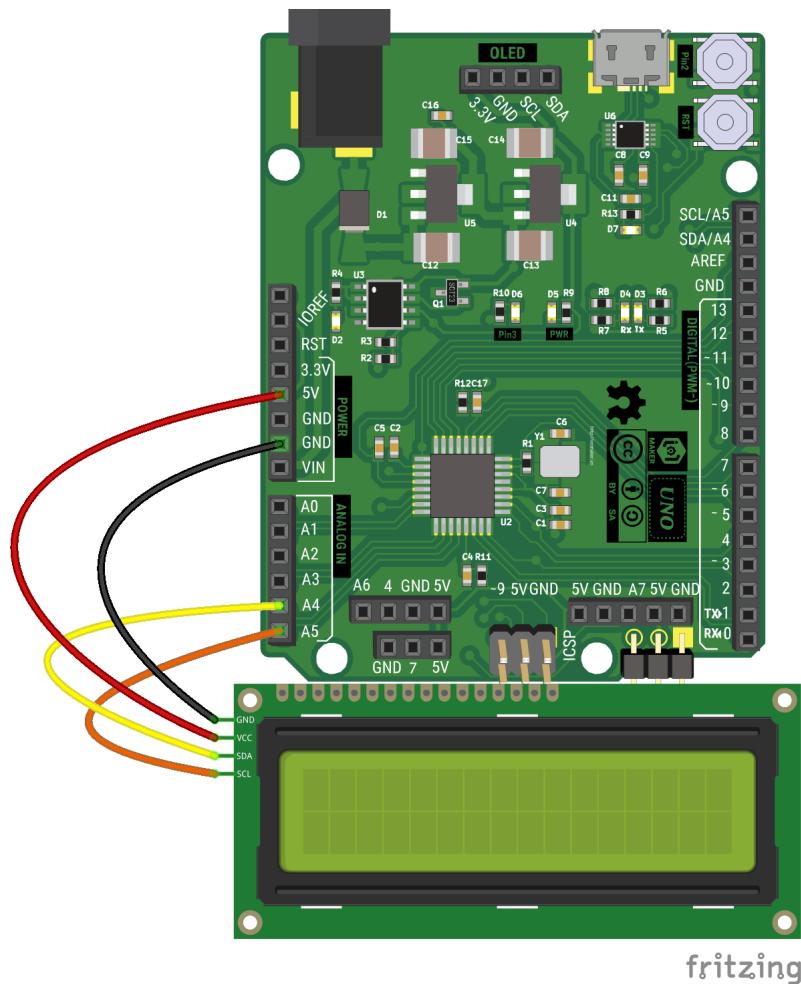
    delay(5000); // Delay 5s cho quá trình scan tiếp theo
}

```

Với chương trình này, đầu tiên ta kiểm tra khi chỉ có 1 master - 1 slave với master là [board IoT Maker UnoX](#) và slave là [Module LCD 20x4](#) [các kiến thức cơ bản về hoạt động của LCD bạn đọc tham khảo ở nội dung phần tiếp theo [Giới thiệu về LCD và OLED](#)].

Bảng 8. Bảng kết nối board IoT Maker UnoX với Module LCD 20x4

Module LCD20x4	IoT Maker UnoX
VCC	5V
GND	GND
SDA	A4
SCL	A5



Hình 77. Hình ảnh kết nối LCD 20X04 với board IoT Maker UnoX

Sau khi kết nối thành công, ta nạp chương trình trên vào board IoT Maker UnoX và kiểm tra kết quả.

```
I2C Scanner
Scanning...
I2C device found at address 0x3F !
Device no. 1
number of devices 1
Scanning...
I2C device found at address 0x3F !
Device no. 1
number of devices 1
```

Hình 78. Kết quả hiển thị địa chỉ I2C của LCD 20X04

Kết quả:

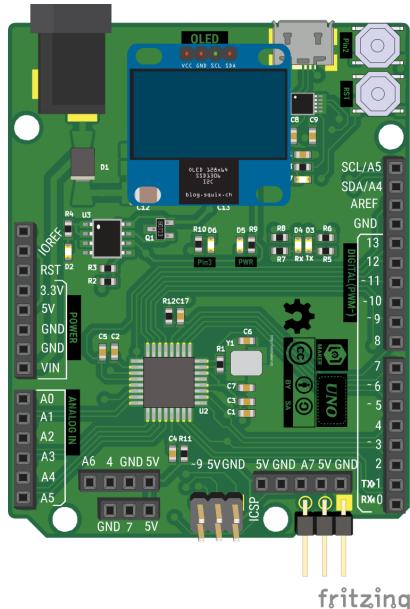
- Số thiết bị kết nối vào là 1 thiết bị.
- Slave có địa chỉ I2C là 0x3F, ta sẽ dùng địa chỉ này để khai báo trong những chương trình giao tiếp I2C với module này.

Tiếp theo, ta xét việc định địa chỉ với chương trình trên khi trong giao tiếp I2C có ít nhất 2 thiết bị làm slave. Kết nối board IoT Maker UnoX với LCD 20x04 và OLED SSD1306.

Bảng 9. Bảng kết nối board IoT Maker UnoX với OLED

OLED SSD1306	IoT Maker UnoX
VCC	5V

OLED SSD1306	IoT Maker UnoX
GND	GND
SDA	A4
SCL	A5



Hình 79. Hình ảnh kết nối OLED với board IoT Maker UnoX

Kết nối song song các chân SCL, SDA, VCC, GND của các thiết bị slave và thiết bị master với nhau. Sau khi kết nối thành công, nạp chương trình trên vào cho board IoT Maker UnoX.

Kết quả

```
I2C. 1
I2C device found at address 0x3F !
Device no. 2
number of devices 2
...
I2C device found at address 0x3C !
Device no. 1
I2C device found at address 0x3F !
Device no. 2
number of devices 2
I2C Scanner
Scanning...
I2C device found at address 0x3C !
Device no. 1
I2C device found at address 0x3F !
Device no. 2
number of devices 2
Scanning...
I2C device found at address 0x3C !
Device no. 1
I2C device found at address 0x3F !
Device no. 2
number of devices 2
```

Hình 80. Kết quả hiển thị địa chỉ I2C của LCD 20X04 và OLED SSD1306

Bạn đọc có thể sẽ đặt câu hỏi rằng: "Vậy việc định địa chỉ sẽ như thế nào khi trong giao tiếp I2C có 2 thiết bị giống nhau?"

Để kiểm chứng việc này, ta sẽ kết nối 2 thiết bị giống nhau trong giao tiếp I2C và nạp chương trình trên. Kết quả nhận được sẽ là chương trình chỉ phát hiện được có 1 thiết bị kết nối vào, và trả về

địa chỉ I2C duy nhất của thiết bị đó.



Một số module có các option để lựa chọn địa chỉ I2C thông qua việc setup phần cứng, một số module cho phép người dùng có thể định lại địa chỉ I2C bằng phần mềm, nếu không chúng ta có thể sử dụng các mạch I2C Multiplexer khi muốn kết nối nhiều module có cùng địa chỉ I2C.

Giới thiệu về LCD và OLED

Giới thiệu về LCD

LCD là chữ viết tắt của Liquid Crystal Display, tiếng Việt có nghĩa là màn hình tinh thể lỏng, đây là loại thiết bị để hiển thị nội dung, cấu tạo bởi các tế bào (cũng là các điểm ảnh) chứa các tinh thể lỏng (liquid crystal) có khả năng thay đổi tính phân cực của ánh sáng và do đó thay đổi cường độ ánh sáng truyền qua khi kết hợp với các kính lọc phân cực. LCD có ưu điểm là phẳng, cho hình ảnh sáng, chân thật và tiết kiệm năng lượng.



Hình 81. Màn hình LCD 20x04

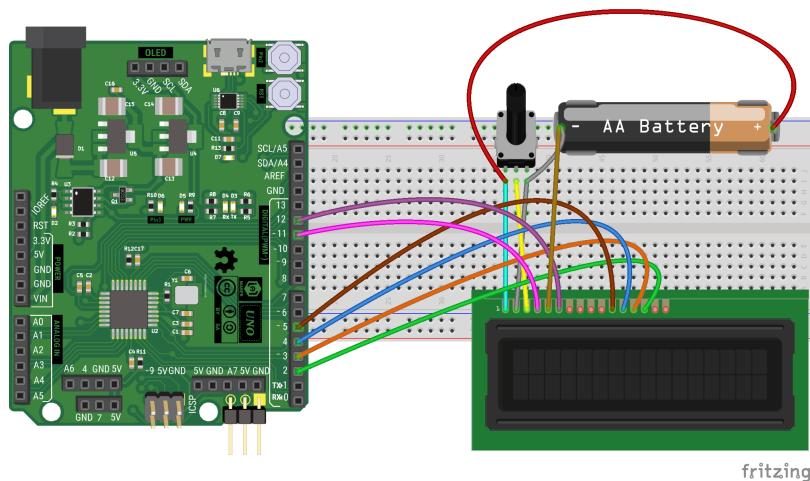
Các hãng sản xuất linh kiện ngày nay sản xuất nhiều module LCD hỗ trợ cho việc tương tác với các vi điều khiển mà phổ biến nhất là 2 module LCD text 16x02 và LCD text 20x04. Các thông số 16x02 và 20x04 là số hàng và số cột tương ứng của các module, ví dụ với 16x02 cho biết module có 16 hàng và 2 cột, 20x04 là 20 hàng và 4 cột.

Như ở hình vẽ, các module có 16 chân để kết nối với chức năng mỗi chân:

- **VSS**: Tương đương với GND - cực âm.
- **VDD**: Tương đương với VCC - cực dương (5V).

- **Contrast Voltage (Vo)**: Điều khiển độ sáng màn hình.
- **Register Select (RS)**: Lựa chọn thanh ghi (RS=0 chọn thanh ghi lệnh, RS=1 chọn thanh ghi dữ liệu).
- **Read/Write (R/W)**: R/W=0 ghi dữ liệu , R/W=1 đọc dữ liệu.
- **Enable pin**: Cho phép ghi vào LCD.
- **D0 - D7**: 8 chân nhận dữ liệu.
- **Backlight (Backlight Anode [+]) và Backlight Cathode [-]**: Tắt bật đèn nền màn hình LCD.

Để sử dụng module LCD, người dùng cần gọi thư viện LCD <LiquidCrystal.h> tích hợp sẵn trên trình biên dịch Arduino IDE. Nếu không dùng thêm module nào, người dùng sẽ phải đấu nối 8 dây tín hiệu và cần dùng thêm một biến trả để điều chỉnh độ sáng màn hình như hình mô phỏng bên dưới.

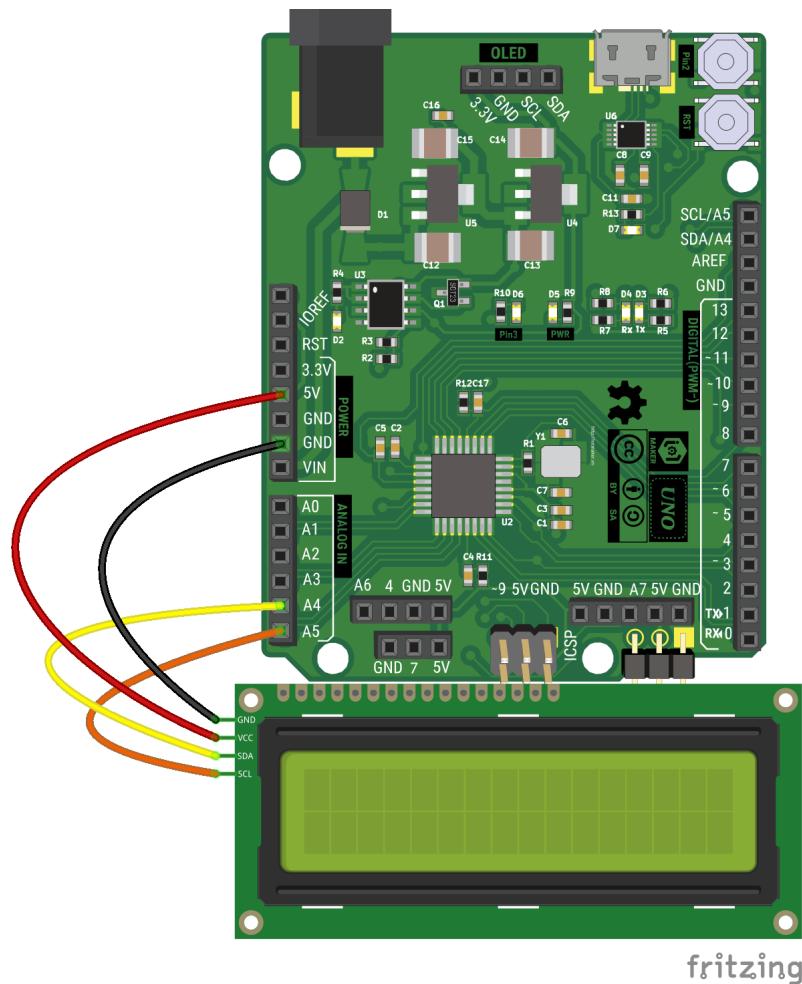


Hình 82. Hình ảnh mô phỏng kết nối dây với LCD

Để khắc phục nhược điểm đấu nối nhiều dây với module LCD thông thường, các nhà sản xuất đã tích hợp thêm IC LCM1602 hỗ trợ giao tiếp I2C vào module LCD như hình vẽ, việc đấu nối cũng như nạp chương trình từ đây sẽ trở nên đơn giản hơn.

Bảng 10. Bảng kết nối IoT Maker UnoX với IC LCM1602

LCM1602	IoT Maker UnoX
VCC	5V
GND	GND
SDA	A4
SCL	A5



fritzing

Hình 83. Hình ảnh kết nối module LCD2004 với board IoT Maker UnoX

Chúng ta sẽ viết chương trình hiển thị ký tự lên màn hình LCD. Với module LCD có module I2C kèm theo, chúng ta cần thêm thư viện `LiquidCrystal_I2C` và `Wire.h`.

Link download thư viện : [Liquid Crystal I2C](#). Add thư viện sau khi download vào chương trình.

Yêu cầu

Chương trình hiển thị dòng chữ "IOT MAKER VIETNAM" và "Hello World" lên LCD trên 2 hàng

Source code

```

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x3F, 20, 4); // Khởi tạo lcd 20x04 với 0x3F là địa chỉ của LCD

void setup()
{
    lcd.begin(); // Khởi tạo LCD
    lcd.backlight(); // Mở đèn nền của LCD
    lcd.setCursor(1,0); // Lệnh setCursor() tương tự như với thư viện LiquidCrystal
    lcd.print("IOT MAKER VIETNAM");
    lcd.setCursor(3,1);
    lcd.print("Hello, world!");
}

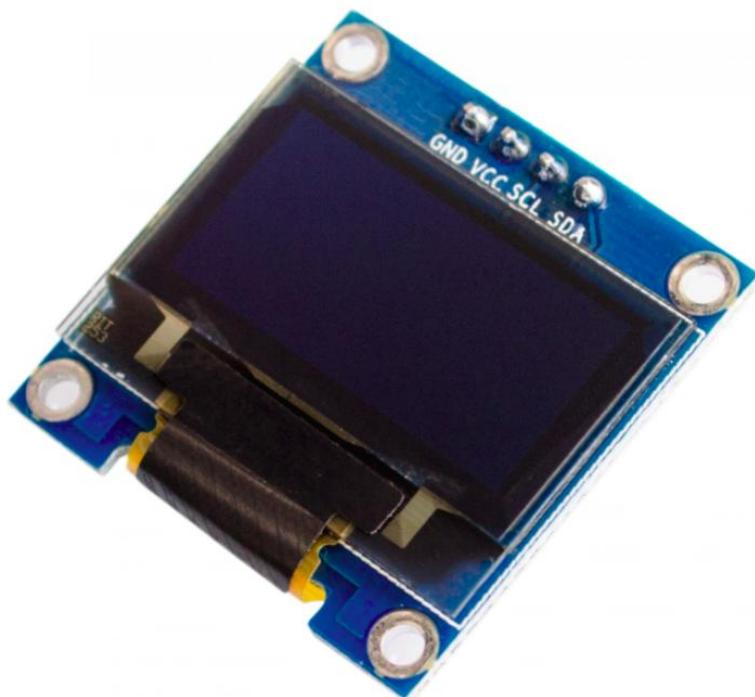
void loop()
{
    // Không làm gì cả
}

```

Giới thiệu về OLED

OLED là chữ viết tắt của Organic Light Emitting Diode), là loại màn hình hiển thị bao gồm một lớp vật liệu hữu cơ với thành phần chính là carbon nằm giữa hai điện cực anode và cathode, nó sẽ tự động phát sáng mỗi khi có dòng điện chạy qua. OLED sử dụng diode phát quang hữu cơ, chính vì thế OLED không cần tới đèn nền chiếu sáng nên có kích thước nhỏ gọn cũng như tiết kiệm điện hơn so với các loại LCD, độ sáng của OLED cũng tương đối tốt ở môi trường sáng tự nhiên.

OLED SSD1306



Hình 84. Hình ảnh màn hình OLED SSD1306

OLED SSD1306 là loại OLED có màn hình loại nhỏ, kích thước tầm 0.96 inch cho tới 1.25 inch. OLED SSD1306 hỗ trợ chuẩn giao tiếp I2C được sử dụng khá rộng rãi trong các sản phẩm điện tử. Tấm nền của OLED được điều khiển bằng chip driver SSD1306. Về cơ bản, để OLED có thể hiển thị được các thông tin mong muốn thì cần có thư viện hỗ trợ, cũng giống như khi làm việc với LCD. Tùy vào mỗi loại vi điều khiển với kiến trúc phần cứng khác nhau mà sẽ có những thư viện OLED SSD1306 khác nhau hỗ trợ, ví dụ như với các board dùng chip ESP8266 có thể sử dụng thư viện: [ESP8266 OLED SSD1306](#), với board IoT Maker UnoX (dùng chip ATmega328), thư viện OLED hỗ trợ được nhiều người sử dụng là [Adafruit SSD1306](#).



Để sử dụng được hoàn chỉnh tất cả các tính năng thư viện này bao gồm cả những tính năng đồ họa, người dùng cần cài đặt thêm thư viện: [Adafruit GFX Library](#) (thư viện hỗ trợ thêm các tính năng đồ họa).

Viết chương trình hiển thị thời gian lên màn hình OLED

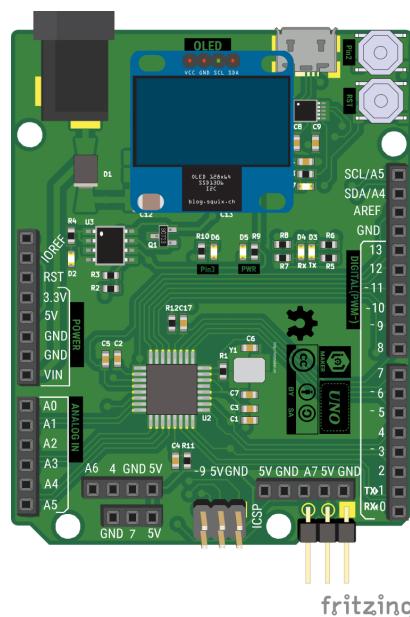
Yêu cầu

Chương trình hiển thị thời gian từ lúc nạp chương trình vào board IoT Maker UnoX.

Đầu nối

Bảng 11. Kết nối IoT Maker UnoX với OLED

OLED	IoT Maker UnoX
VCC	5V
GND	GND
SDA	A4
SCL	A5



Hình 85. Hình ảnh kết nối OLED SSD1306 với board IoT Maker UnoX

Source code

```

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define OLED_RESET 4
#define LOGO16_GLCD_HEIGHT 16
#define LOGO16_GLCD_WIDTH 16

#if (SSD1306_LCDHEIGHT != 32)
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

Adafruit_SSD1306 display(OLED_RESET);
int time_run = 0;

void setup()
{
    Serial.begin(9600);

    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    display.display();
    delay(2000);
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 0);
    display.println("Hello, world!");
    display.display();
    delay(2000);
    display.clearDisplay();
}
void loop()
{
    int hour_run, min_run, sec_run;
    delay(1000);
    time_run++;
    hour_run = time_run / 3600;
    min_run = (time_run % 3600) / 60;
    sec_run = time_run % 60;
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println(String(hour_run) + ":" + String(min_run) + ":" + String(sec_run));
    display.display();
}

```

Ngoài ra, chúng ta cũng có thể tự tìm hiểu thêm các tính năng đồ họa của OLED với thư viện Adafruit SSD1306 với các chương trình có sẵn trong phần Example của thư viện.

Giao tiếp giữa 2 board IoT Maker UnoX

Có nhiều cách để giao tiếp giữa các vi điều khiển với nhau như truyền nhận qua Serial như đã đề cập ở chương 3. Phần này, chúng ta sẽ tìm hiểu cách giao tiếp giữa 2 board IoT Maker UnoX thông qua chuẩn giao tiếp I2C.

Yêu cầu*

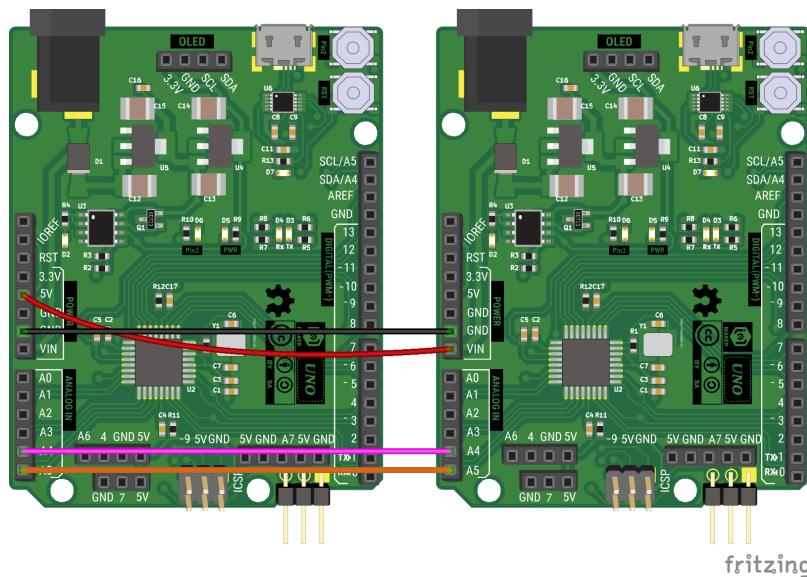
Board master điều khiển bật tắt LED trên thiết bị slave.

Kết nối

Đầu tiên ta thực hiện việc kết nối giữa 2 board IoT Maker UnoX.

Bảng 12. Kết nối giữa Arduino

Master	Slave
5V	Vin
GND	GND
A4	A4
A5	A5



Hình 86. Hình ảnh kết nối giữa 2 board IoT Maker UnoX

Ở đây, để thuận tiện ta có thể lấy ngõ ra 5V trên master để cấp nguồn cho slave, người dùng cũng có thể cấp nguồn riêng cho thiết bị slave trong các ứng dụng thực tế tùy theo yêu cầu sử dụng.

Nạp các chương trình cho master và slave.

Source code cho master

```

// Master
#include <Wire.h>
void setup()
{
    Serial.begin(9600); // Khởi tạo giao tiếp I2C
    Wire.begin(); // Khởi tạo truyền nhận dữ liệu I2C
}

void loop()
{
    // Bắt đầu quá trình truyền dữ liệu trên Serial Monitor
    while (Serial.available()) {
        char c = Serial.read(); // Đọc dữ liệu từ serial nếu có và lưu vào biến c
        if (c == 'H') {
            Wire.beginTransmission(3); // Bắt đầu truyền đến slave với address là 3
            Wire.write('H'); // Truyền chữ H đến slave
            Wire.endTransmission(); // Kết thúc quá trình truyền
        } else if (c == 'L') {
            Wire.beginTransmission(3);
            Wire.write('L');
            Wire.endTransmission();
        }
    }
}

```

Source code cho slave

```

// Slave
#include <Wire.h>
#define pinLed 3 // Định nghĩa chân LED trên board Iotmaker Uno X
void setup()
{
    Wire.begin(3); // Khởi tạo giao tiếp I2C với địa chỉ của slave là 3
    Wire.onReceive(receiveEvent); // Đăng ký hàm receiveEvent sẽ được gọi khi nhận được dữ liệu
    pinMode(pinLed,OUTPUT);
    digitalWrite(pinLed,LOW);
}

void loop()
{
    // Không làm gì cả
}
void receiveEvent()
{
    while(Wire.available()) {
        char c = Wire.read(); // Lưu dữ liệu nhận được từ master vào biến c nếu có
        if(c == 'H') // So sánh dữ liệu nhận được và điều khiển LED
            digitalWrite(pinLed,HIGH);
        else if(c == 'L')
            digitalWrite(pinLed,LOW);
    }
}

```

Giải thích source code

I2C sử dụng việc định địa chỉ 7 bit tương ứng với 128 thiết bị kết nối. Trong Arduino, ta có thể định địa chỉ cho slave bởi hàm `Wire.begin(địa chỉ slave)`, khi đó, slave được khởi tạo một địa chỉ với địa chỉ nằm trong khoảng từ 1 tới 127. Master sẽ truyền dữ liệu tới slave qua câu lệnh `Wire.beginTransmission(địa chỉ slave)`. Master thì không cần truyền địa chỉ.

Kết quả

Sau khi nạp chương trình cho master và slave thành công, ta mở cửa sổ Serial Monitor ở board IoT Maker UnoX master lên và gõ vào dòng **Send** kí tự **H** hoặc **L** tương ứng với các lệnh để bật và tắt LED trên board IoT Maker UnoX slave.



Hình 87. Hình ảnh gởi dữ liệu trên Serial Monitor

Tổng kết

Qua phần này, chúng ta đã tìm hiểu được những khái niệm cơ bản về giao tiếp I2C, các kết nối giữa vi điều và các IC sử dụng giao thức này, hiểu được cách xác định địa chỉ và tìm hiểu một số chương trình cơ bản dùng giao thức này. Bạn đọc có thể tham khảo các ứng dụng mở rộng của giao thức này với các module cảm biến giá tốc, thời gian thực, các module phối hợp chuẩn I2C và SPI,... Tùy vào những ứng dụng cụ thể mà người dùng sẽ phải lựa chọn những IC thích hợp dùng giao tiếp I2C.

Chuẩn giao tiếp truyền nhận dữ liệu SPI

Chương này chúng ta sẽ tìm hiểu về một chuẩn giao tiếp khá thông dụng trong truyền nhận dữ liệu, đó là chuẩn giao tiếp truyền nhận SPI. Điểm qua 1 số nội dung sẽ tìm hiểu ở chương này:

- Giới thiệu về chuẩn SPI, lịch sử hình thành và nguyên lý hoạt động.
- Một số ví dụ sử dụng SPI trong truyền, nhận dữ liệu như điều khiển LED matrix và đọc giá trị nhiệt độ, áp suất, độ cao bằng cảm biến BMP280 hiển thị giá trị lên màn hình OLED.

Giao thức SPI

Giới thiệu

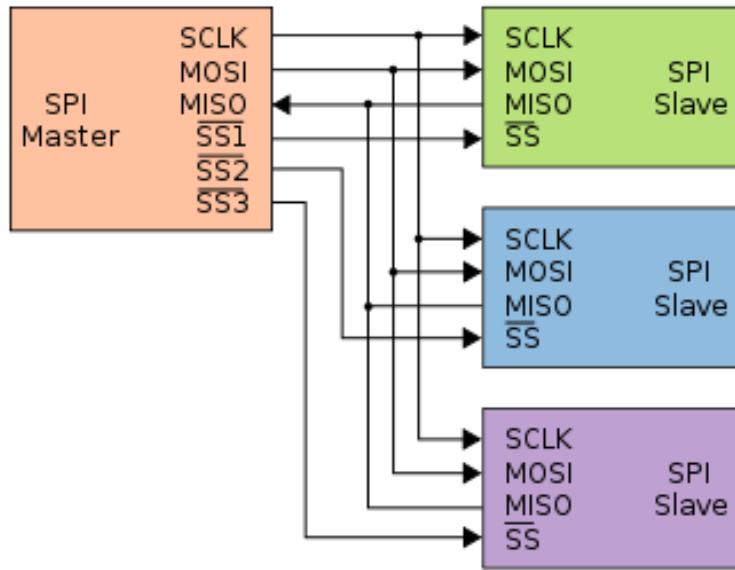
Với tốc độ phát triển của công nghệ ngày nay thì việc truyền dữ liệu qua các chuẩn truyền I2C, UART chưa đáp ứng được đối với các dự án cần truyền dữ liệu với tốc độ cao, để đáp ứng điều đó hãng Motorola đã đề xuất ra chuẩn truyền SPI.

SPI là chữ viết tắt của Serial Peripheral Interface, chuẩn giao tiếp nối tiếp đồng bộ tốc độ cao do hãng Motorola đề xuất được sử dụng cho truyền thông khoảng cách ngắn, chủ yếu là trong các hệ thống nhúng. Giao diện được Motorola phát triển vào giữa những năm 1980 và đã trở thành tiêu chuẩn trong thực tế. Các ứng dụng điển hình như Secure Digital cards (các loại thẻ nhớ SD ví dụ: miniSD, microSD cards) và liquid crystal displays (màn hình tinh thể lỏng).

Đôi khi SPI còn được gọi là chuẩn truyền thông "4 dây" vì nó có 4 đường giao tiếp là SCK (Serial Clock), MISO (Master Input/Slave Output), MOSI (Master Output/Slave Input) và SS(Slave Select).

- **SCK (Serial Clock):** Là đường xung giữ nhịp cho chuẩn SPI, là chân output từ master, do chuẩn SPI là giao tiếp đồng bộ nên phải cần dùng một đường giữ nhịp. Đây là sự khác biệt giữa truyền thông đồng bộ và truyền thông không đồng bộ như chuẩn giao tiếp UART. SCK giúp chuẩn SPI có tốc độ truyền/nhận dữ liệu cao và ít xảy ra lỗi trong quá trình truyền/nhận dữ liệu.
- **MISO (Master Input/Slave Output):** Với master thì MISO là chân input và với slave là chân output, 2 chân MISO của master và slave nối trực tiếp với nhau.
- **MOSI (Master Output/Slave Input):** Với chip master thì MOSI là chân output và với chip slave là chân input, 2 đường MOSI của master và slave nối trực tiếp với nhau.
- **SS (Slave Select):** Là chân chọn thiết bị slave cần giao tiếp, trên thiết bị slave sẽ có một chân slave kết nối với chân SS của master và trên thiết bị master sẽ có nhiều chân SS điều khiển thiết bị slave. Chân SS trên các chip slave sẽ ở mức cao khi không giao tiếp, nếu chip master kéo đường SS của một slave nào đó xuống mức thấp thì master sẽ giao tiếp với slave đó.

Chuẩn truyền SPI sử dụng kiểu truyền thông master-slave, với một master có thể điều khiển nhiều slave thông qua việc lựa chọn các đường SS (Slave Select), muốn điều khiển slave nào thì chỉ cần chọn SS của slave đó. Các thiết bị sử dụng chuẩn truyền SPI sẽ truyền dữ liệu song công (duplex communication) là truyền và nhận dữ liệu cùng lúc, master có thể gửi dữ liệu đến slave và nhận dữ liệu từ slave cùng một thời điểm.



Hình 88. Hình ảnh cách kết nối các thiết bị trong giao thức SPI (Nguồn en.wikipedia.org)

SPI, ưu và nhược điểm

Ưu điểm

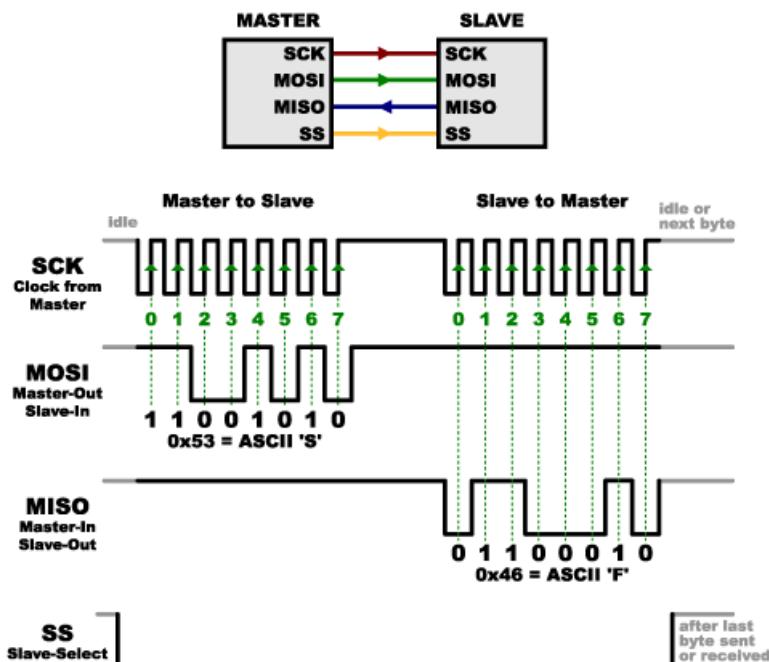
- Chuẩn truyền thông nối tiếp SPI có tốc độ truyền dữ liệu và ít lỗi phát sinh trong quá trình truyền/nhận dữ liệu hơn các chuẩn truyền nối tiếp khác.
- Hỗ trợ truyền thông song công (duplex communication) là dữ liệu có thể truyền và nhận cùng một thời điểm.
- Có giao diện phần cứng khá đơn giản.
- Không giới hạn tốc độ xung clock, cho phép truyền dữ liệu với tốc độ cao.
- Hỗ trợ điều khiển nhiều slave.

Nhược điểm:

- Tốn năng lượng.
- Chỉ hỗ trợ một master.
- Không có giao thức kiểm tra lỗi.
- SPI đòi hỏi các slave có một đường SS (slave Select) riêng biệt, vì thế nếu cần nhiều slave thì sẽ cần nhiều đường SS sẽ làm tốn chân của chip master và nhiều dây sẽ gây rối.

Nguyên lý hoạt động

Thiết bị master và slave mỗi thiết bị có thanh ghi dữ liệu 8 bit. Khi đường SCK của master tạo ra một xung nhịp thì một bit trong thanh ghi dữ liệu của master truyền qua slave trên đường MOSI, và ngược lại một bit dữ liệu từ slave sẽ truyền qua master trên đường MISO, do 2 dữ liệu được truyền cùng một lúc trên một nhịp xung nên quá trình truyền dữ liệu này gọi là truyền dữ liệu "song công".



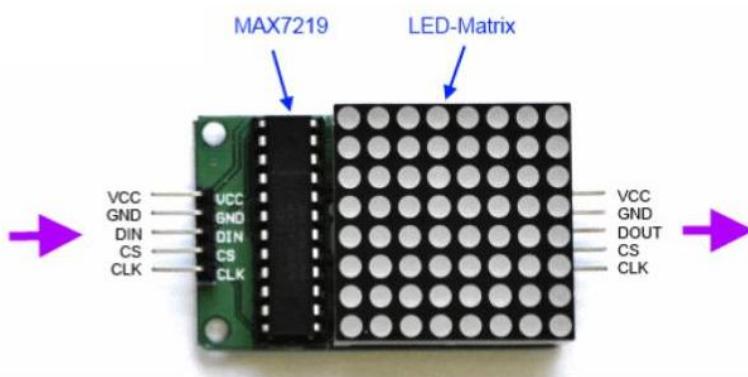
Hình 89. Quá trình truyền nhận dữ liệu trong giao thức SPI (Nguồn learn.sparkfun.com)

SPI, các ví dụ mẫu

Hiển thị chữ trên LED matrix

Yêu cầu

Đây là một ví dụ cơ bản của chuẩn giao tiếp nối tiếp SPI. Vì điều khiển giao tiếp với module LED matrix để hiển thị chữ.



Hình 90. Hình ảnh module LED ledmatrix

LED matrix 8x8 MAX7219 dùng IC 7219 để điều LED matrix 1 cách dễ dàng và đơn giản hơn, dùng 3 dây dữ liệu để truyền dữ liệu và 2 dây nguồn. Module 8x8 LED matrix sử dụng khá đơn giản, có thể điều chỉnh độ sáng của LED ngay trên phần mềm.

Linh kiện cần dùng

- [Board IoT Maker UnoX](#)
- [Module LED matrix MAX7219](#)
- [Dây cắm breadboard male-female](#)

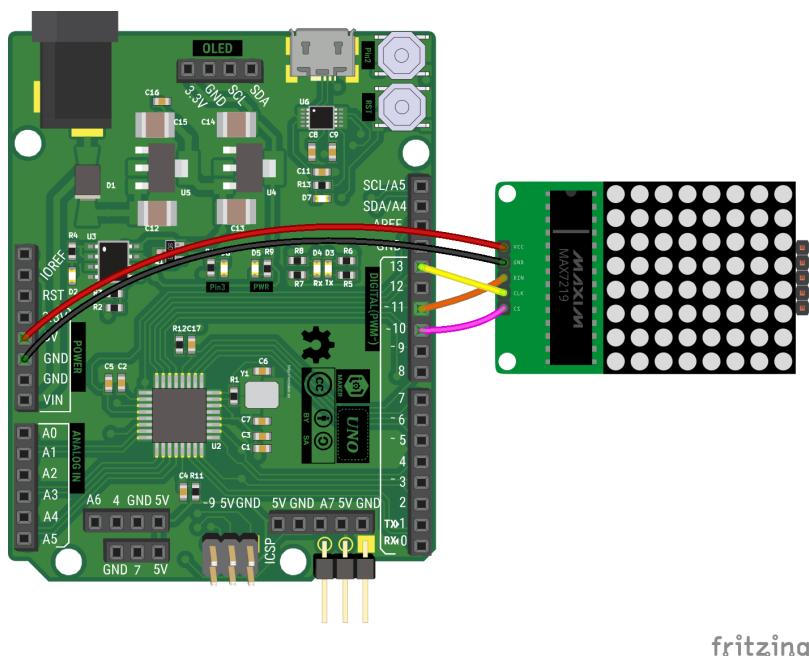
Kết nối IoT Maker UnoX với LED matrix

Board IoT Maker UnoX sẽ là master và LED matrix sẽ là slave.

- Master sẽ gửi dữ liệu ra từ chân D11 (MOSI) và slave sẽ nhận dữ liệu bằng chân DIN.
- Chân D13 (SCK) của master sẽ tạo xung clock qua chân CLK của slave mỗi nhịp sẽ gửi 1bit dữ liệu qua slave.
- Chân D10 (SS) của master nối với chân CS của slave khi muốn giao tiếp với slave thì chân D10 (SS) của master sẽ kéo chân CS của slave xuống mức thấp.

Bảng 13. Bảng kết nối LED matrix và board IoT Maker UnoX

LED matrix	Board IoT Maker UnoX
VCC	5V
GND	GND
DIN	D11 (MOSI)
CLK	D13 (SCK)
CS	D10 (SS)



Hình 91. Hình ảnh kết nối module LED matrix với board IoT Maker UnoX

Thư viện cần dùng:

Với những thư viện không có sẵn trong trình biên dịch Arduino IDE thì cần phải clone (dùng với [git](#)) hoặc Download về máy và add vào chương trình. Các thư viện cần dùng cho ứng dụng được liệt kê bên dưới:

- Thư viện "SPI.h" là thư viện đã có sẵn trong trình biên dịch Arduino IDE.
- Thư viện [bitBangedSPI](#).
- Thư viện [MAX7219_Dot_Matrix](#).

Source code

```
#include <SPI.h>
#include <bitBangedSPI.h>
#include <MAX7219_Dot_Matrix.h>

MAX7219_Dot_Matrix display (chips, 10); // Chân D10 là chân SS của board Iotmaker Uno X

const byte chips = 1; // Số chip MAX7219 được sử dụng
const char message [] = "IOT MAKER VN"; // Nội dung được hiển thị
unsigned long lastMoved = 0;
unsigned long MOVE_INTERVAL = 40; // Thời gian chạy chữ đơn vị (ms)
int messageOffset;

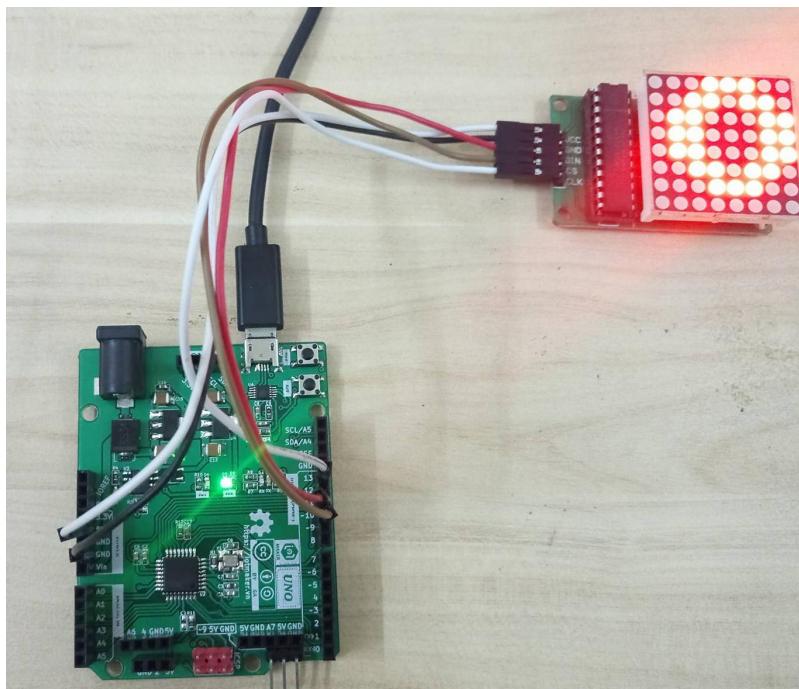
void updateDisplay ()
{
    // Hiển thị chữ của mảng message, bắt đầu từ pixel mesageOffset
    display.sendSmooth (message, messageOffset);

    // Mỗi thời gian hiển thị một pixel từ phải qua trái
    if (messageOffset++ >= (int) (strlen (message) * 8))
        messageOffset = -chips * 8;
}

void setup ()
{
    display.begin (); // Khởi tạo hiển thị
}

void loop ()
{
    // Nội dung được hiển thị lại sau khi chạy
    if (millis() - lastMoved >= MOVE_INTERVAL) {
        updateDisplay ();
        lastMoved = millis();
    }
}
```

Kết quả

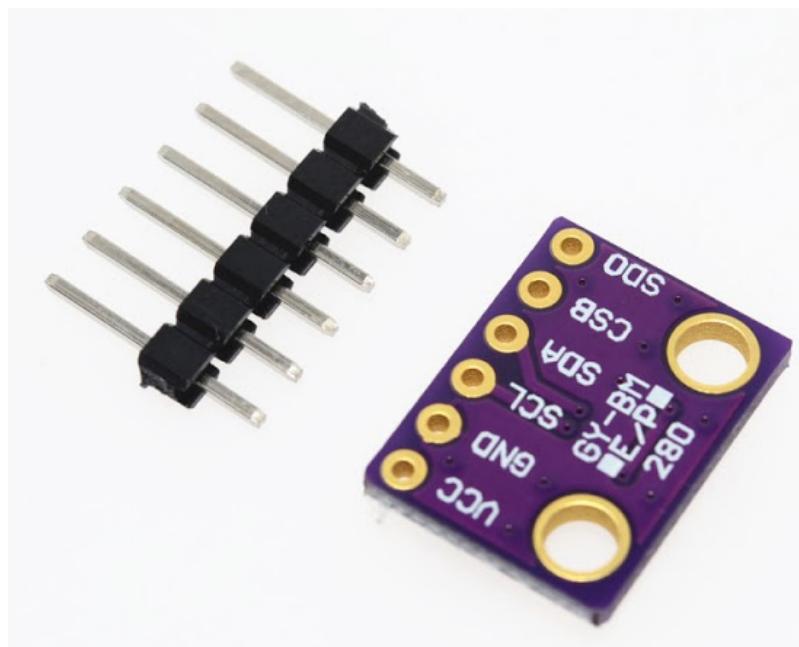


Đọc dữ liệu từ cảm biến BMP280, hiển thị trên OLED

Yêu cầu

Ứng dụng này giúp chúng ta đọc dữ liệu từ cảm biến áp suất và hiển thị giá trị lên màn hình OLED.

Module cảm biến áp suất BMP280



Hình 92. Hình ảnh module cảm biến áp suất BMP280

[BMP280](#) là cảm biến nâng cấp thế hệ tiếp theo cho BMP085/BMP180/BMP183. Với chi phí thấp, độ chính xác cao. Module có chức năng đo áp suất khí quyển và nhiệt độ. Chúng ta cũng có thể sử dụng nó như một module đo độ cao (sai số $\pm 1\text{m}$) bởi mối liên hệ giữa áp suất và độ cao.

Linh kiện cần dùng

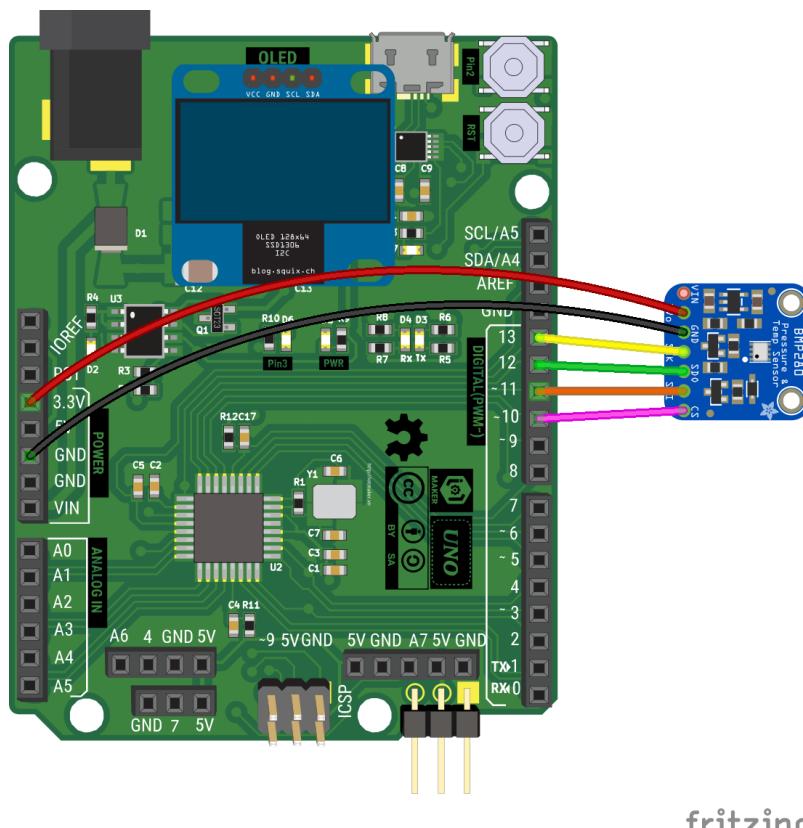
- [Board IoT Maker UnoX](#)
- [Module cảm biến áp suất BMP280](#)
- [Màn hình OLED](#)
- [Dây cắm breadboard male-female](#)

Kết nối :

- Arduino sẽ là master và cảm biến BMP280 sẽ là slave, slave sẽ gửi dữ liệu qua đường SDI.
- Mỗi nhịp xung clock từ chân SCK (D13) của master tạo ra sẽ ứng với 1bit dữ liệu được truyền.
- Chân D10 (SS) của master nối với chân CS của slave khi muốn giao tiếp với slave thì chân D10 (SS) của master sẽ kéo chân CS của slave xuống mức thấp.

Bảng 14. Bảng đấu nối cảm biến BMP280 và board IoT Maker UnoX

BMP280	IoT Maker UnoX
VCC	3.3V
GND	GND
SDI	D11 (MOSI)
SCK	D13 (SCK)
CSE	D10 (SS)
SDO	D12(MISO)



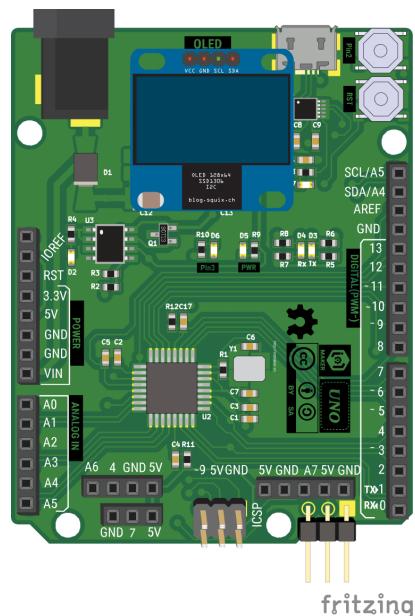
Hình 93. Hình ảnh kết nối module cảm biến áp suất BMP280 với board IoT Maker UnoX

Kết nối board IoT Maker UnoX với OLED

Xem chương I2C để hiểu hơn về giao tiếp Arduino giao tiếp với OLED, chúng ta có thể cắm trực tiếp OLED vào header đã được thiết kế sẵn trên board IoT Maker UnoX hoặc kết nối theo hướng dẫn bên dưới.

Bảng 15. Bảng đấu nối OLED với board IoT Maker UnoX

OLED	IoT Maker UnoX
VCC	3.3V
GND	GND
SDA	SDA(hoặc A4)
SCL	SCL(hoặc A5)



Hình 94. Hình ảnh kết nối OLED SSD1306 với board IoT Maker UnoX

Thư viện cần dùng:

- Thư viện "SPI.h" và "Wire.h" là hai thư viện có sẵn trong Arduino IDE.
- Thư viện Adafruit_SSD1306.h.
- Thư viện Adafruit_GFX.h.
- Thư viện Adafruit_Sensor.h.
- Thư viện Adafruit_BMP280.h.

Source code

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>

#define BMP_SCK    13
#define BMP_MISO   12
#define BMP_MOSI   11
#define BMP_CS     10
#define OLED_RESET 4

Adafruit_SSD1306 display(OLED_RESET);
// Chọn giao tiếp SPI (BMP280 có 2 chuẩn giao tiếp SPI và I2C)
Adafruit_BMP280 bmp(BMP_CS, BMP_MOSI, BMP_MISO, BMP_SCK);

void setup()
{
  Serial.begin(9600);
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C); // Lấy địa chỉ I2C của oled
  display.clearDisplay(); // Lệnh xóa màn hình hiển thị
  display.setTextColor(WHITE); // Chọn màu chữ
  display.setTextSize(1); // Chọn kích thước chữ
  display.setCursor(15, 15); // Chọn vị trí của chữ
}
```

```
display.print(" IOT MAKER VN ");
display.display();
delay(2000);
Serial.println(F("BMP280 test"));

if (!bmp.begin()) { // In ra thông báo nếu kết nối thất bại
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring!"));
    while (1);
} else
    Serial.println("BMP280 OK"); // Hiển thị "BMP280 OK" khi kết nối thành công
}

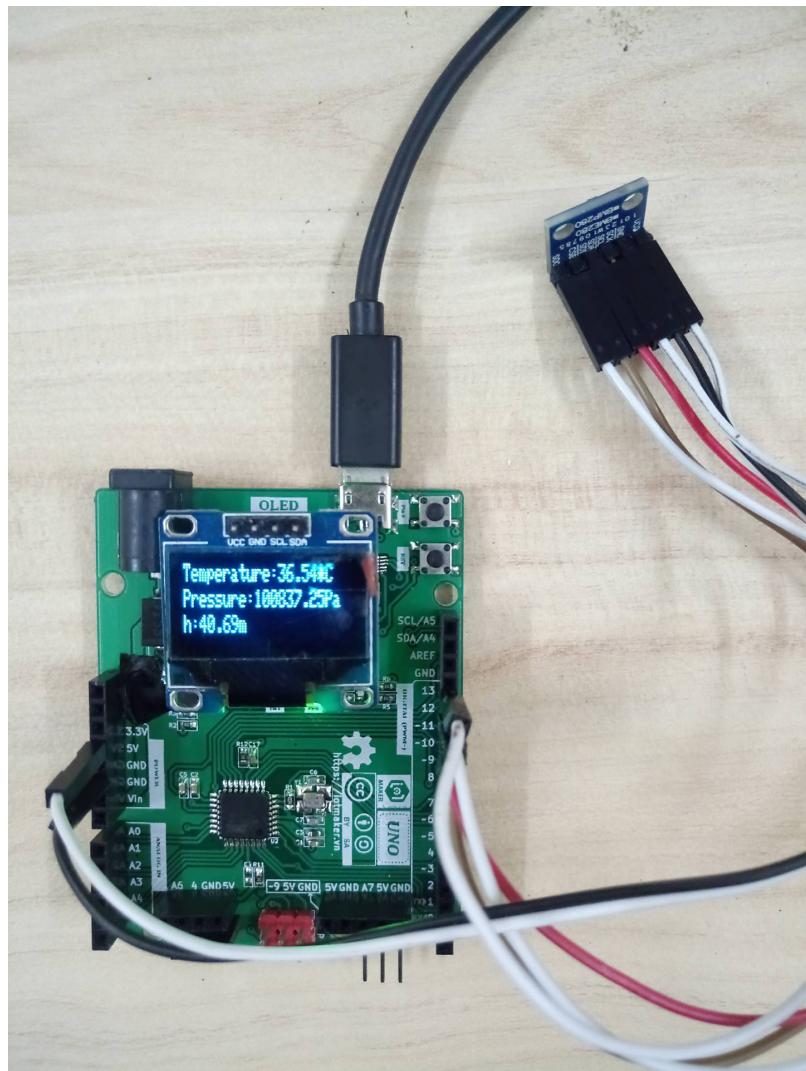
void loop()
{
    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.print("Temperature:");
    display.print(bmp.readTemperature()); // Lấy giá trị nhiệt độ
    display.print("*C");
    // Hiển thị giá trị nhiệt độ trên serial monitor
    Serial.print("Temperature:");
    Serial.print(bmp.readTemperature()); // Lấy giá trị nhiệt độ
    Serial.println("*C");

    display.setCursor(0, 10);
    display.print("Pressure:");
    display.print(bmp.readPressure()); // Lấy giá trị áp suất
    display.print("Pa");
    // Hiển thị giá trị áp suất trên serial monitor
    Serial.print("Pressure:");
    Serial.print(bmp.readPressure()); // Lấy giá trị áp suất
    Serial.println("Pa");

    display.setCursor(0, 20);
    display.print("h:");
    display.print(bmp.readAltitude()); // Lấy giá trị độ cao
    display.print("m");
    // Hiển thị giá trị độ cao trên serial monitor
    Serial.print("h:");
    Serial.print(bmp.readAltitude()); // Lấy giá trị độ cao
    Serial.println("m");

    display.display();
    delay(2000);
}
```

Kết quả



Hình 95. Hình ảnh kết quả hiển thị trên OLED

Tổng kết

Qua phần này, bạn đã tìm hiểu được những khái niệm quan trọng trong giao tiếp SPI, các chân giao tiếp theo chuẩn SPI là SCK, MISO, MOSI, SS. Cách chọn slave để giao tiếp của master và SPI hoạt động theo kiểu truyền song công là như thế nào. Từ đó, chúng ta có thể dễ dàng xây dựng các ứng dụng có hỗ trợ giao tiếp SPI.

Chuẩn giao tiếp 1-Wire

Trong chương này chúng ta sẽ cùng tìm hiểu thêm về 1 chuẩn giao tiếp truyền nhận dữ liệu nữa đó là chuẩn giao tiếp 1-Wire, các ví dụ thực hành đọc cảm biến bằng chuẩn giao tiếp 1-Wire, kết hợp chuẩn giao tiếp 1-Wire với các chuẩn giao tiếp khác như I2C hay SPI.

1-Wire

1-Wire là gì?

Chuẩn giao tiếp 1-Wire được nghiên cứu và phát triển bởi Dallas Semiconductor (Maxim). Không như các chuẩn giao tiếp đã được đề cập trước đó như I2C cần 2 dây hoặc SPI cần 4 dây để truyền nhận dữ liệu. 1-Wire chỉ cần một dây để có thể truyền và nhận dữ liệu.

Chuẩn giao tiếp 1-Wire là chuẩn giao tiếp không đồng bộ và bán song công (half-duplex: tại một thời điểm, tín hiệu chỉ có thể chạy theo một hướng). Vì 1-Wire chỉ sử dụng một dây để nối nguồn và truyền dữ liệu, nên khi ở trạng thái rãnh (không có dữ liệu trên đường truyền) thì nó cần phải ở mức cao, do đó cần kết nối dây này với nguồn thông qua một điện trở. Điện trở này thường được gọi là điện trở kéo lên (pull-up resistor). Tùy theo các thiết bị mà giá trị điện trở này có thể thay đổi, cần tham khảo datasheet của thiết bị để biết rõ.

1-Wire thường sử dụng các cổng logic CMOS/TTL tương ứng với mức logic 0 thì điện áp đỉnh ở mức 0.8V và điện áp tối thiểu cho mức logic 1 là 2.2V. Các thiết bị 1-Wire có nguồn cấp trong khoảng 2.8V đến 6V.

1-Wire có hai chế độ làm việc là **standard** và **overdrive**. Khi làm việc ở chế độ **standard** thì tốc độ truyền dữ liệu là **15.4kbps**, với chế độ **overdrive** là **125kbps**.

Chuẩn giao tiếp 1-Wire tuân theo mô hình master-slave. Trên một đường truyền dữ liệu có thể gắn nhiều thiết bị slave, nhưng chỉ có duy nhất một thiết bị là master. Mỗi một thiết bị slave sẽ có duy nhất 64-bit địa chỉ lưu trữ trong bộ nhớ ROM của mình. Nhờ thế, khi kết nối vào chung một bus dữ liệu thì thiết bị master sẽ có thể nhận biết được giữa các slave. Trong **8 byte** (64 bit) này sẽ được chia ra làm 3 phần chính:

- Bắt đầu là LSB (least significant bit), byte đầu tiên bao gồm 8 bit được gửi đi là mã họ thiết bị (family codes) giúp xác định đó là loại thiết bị nào. 6 byte tiếp theo lưu trữ một địa chỉ riêng của từng thiết bị với 48 bit tùy biến. Byte cuối cùng MSB (most significant bit) là byte kiểm tra tính toàn vẹn của dữ liệu - cyclic redundancy check (CRC), đây là byte giúp kiểm tra xem tín hiệu gửi đi có bị lỗi hay không.



Với số lượng 2^{48} bit địa chỉ được tạo ra thì vẫn cần để về địa chỉ không phải là vấn đề chính trong chuẩn giao tiếp này.

Khi có nhiều thiết bị 1-Wire trên một mạch thì nó thường được gọi là mạng 1-Wire hoặc MicroLAN. Trong mạng này, yêu cầu chỉ có duy nhất một **master** và phải có ít nhất là một **slave**. Có thể kết nối bao nhiêu slave tùy thích, nhưng khi vượt quá con số **20** thì việc truyền nhận dữ liệu có thể sẽ xảy ra lỗi.

Đường truyền giữa master và slave có thể lên tới 200 mét. Khi sử dụng dây ở khoảng cách lớn thì cần

được bảo quản tốt hơn, như tránh băng qua các đường dây điện để tránh tín hiệu sẽ bị nhiễu. Khi sử dụng ở khoảng cách xa, loại dây rẻ nhất và dễ sử dụng nhất là dây CAT5 được sử dụng trong mạng LAN (dây cable internet).

1-Wire hoạt động như thế nào?

Chuẩn giao tiếp 1-Wire sử dụng khái niệm time slot (khe thời gian). Một time slot là một khoảng thời gian trong đó mức logic 1 hoặc 0 sẽ được ghi hoặc đọc. Time slot có khoảng thời gian là 60μs khi hoạt động ở chế độ standard, và 8μs với chế độ overdrive.

Có 3 thao tác hoạt động cơ bản của 1 Wire là **Reset/Present**, **Read**, **Write**.

1. Reset/Present.

Master sẽ kéo tín hiệu truyền xuống mức thấp trong khoảng 480μs đến 640μs. Khoảng thời gian này được hiểu là khoảng thời gian **reset**. Sau khoảng thời gian này, nếu có slave sẽ gửi trả tín hiệu **present**. Tức là slave sẽ kéo tín hiệu xuống mức thấp trong khoảng 60μs đến 240μs. Nếu không có tín hiệu **present** trả về thì master sẽ hiểu là không có slave nào được kết nối vào mạng, và các quá trình tiếp theo sẽ không được diễn ra.

2. Write.

Đối với bit 1 master kéo đường truyền xuống mức thấp trong khoảng 1 đến 15μs, sau đó sẽ giải phóng đường truyền về mức cao.

Đối với bit 0 master sẽ kéo đường truyền xuống mức thấp trong khoảng 60μs đến 120μs, sau đó sẽ giải phóng đường truyền về mức cao.



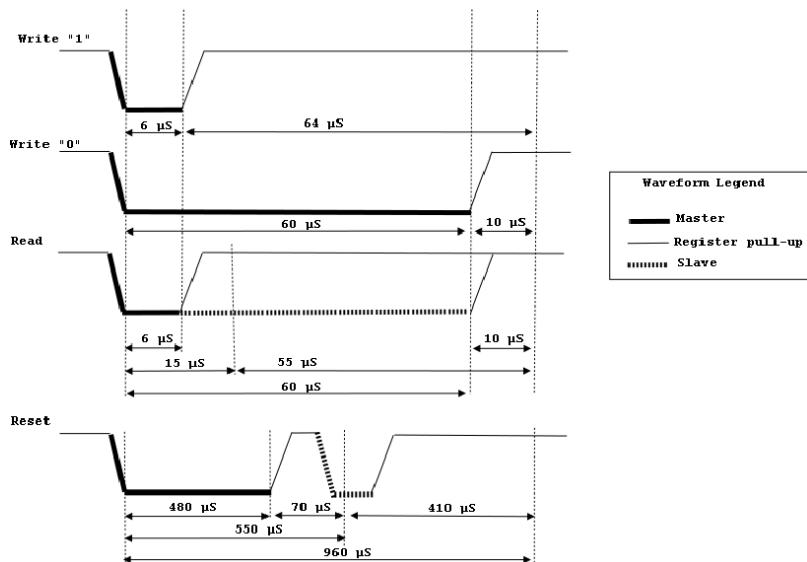
Giữa các lần gửi bit (0 hoặc 1), phải có khoảng thời gian nghỉ (recovery time) tối thiểu 1μs.

3. Read.

Master sẽ kéo tín hiệu truyền xuống mức thấp trong khoảng 0-15μs. Nếu slave muốn gửi bit 1 sẽ giải phóng đường truyền trở về mức cao, nếu muốn gửi bit 0 slave sẽ giữ đường truyền ở mức thấp trong khoảng thời gian 15μs đến 60μs.

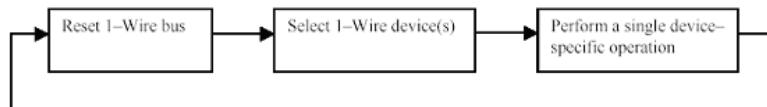


Các khoảng thời gian trong các hoạt động trên được xác định ở chế độ **standard**



Hình 96. Tổng hợp

Tiến trình hoạt động (Workflow)



Hình 97. Workflow

- Trước khi bắt đầu xác định một slave để làm việc, master cần đưa ra lệnh **reset** để xác định là có slave nào đó có nằm trên đường truyền bằng cách phản hồi lại tín hiệu **present**.
- Sau khi đã xác định có thiết bị slave được kết nối, master sẽ chọn tất cả hay chỉ 1 slave (dựa trên địa chỉ của thiết bị) để làm việc. Hoặc sẽ xác định slave tiếp theo bằng thuật toán tìm kiếm nhị phân.
- Sau khi đã xác định được slave làm việc thì tất cả các slave khác sẽ được bỏ qua, và tất cả các tín hiệu truyền đi sẽ chỉ được nhận bởi thiết bị master và slave đã được chọn.
- Nếu muốn giao tiếp với một slave khác, master sẽ bắt đầu lại quá trình từ **bước 1**.



Khi một thiết bị được chọn, master có thể đưa ra các lệnh cụ thể cho thiết bị, gửi dữ liệu đến nó hoặc đọc dữ liệu từ nó. Bởi vì mỗi loại thiết bị thực hiện các chức năng khác nhau và phục vụ cho mục đích khác nhau, mỗi máy có một giao thức duy nhất khi nó đã được chọn. Mặc dù mỗi loại thiết bị có thể có các giao thức và tính năng khác nhau nhưng tất cả chúng đều có quy trình lựa chọn giống nhau và tuân theo tiến trình như trên.

Trên đây là phần giới thiệu về chuẩn giao tiếp 1-Wire, tiếp theo chúng ta sẽ cùng thực hành một số ví dụ để có thể hiểu rõ cách thức hoạt động của chuẩn giao tiếp này.

Ví dụ chuẩn giao tiếp 1-Wire

Ở phần này chúng ta sẽ tìm hiểu về ví dụ đọc dữ liệu từ cảm biến đo nhiệt độ DS18B20. Sẽ có hai ví dụ với cảm biến này là:

- Một master và một slave.
- Một master và nhiều slave.

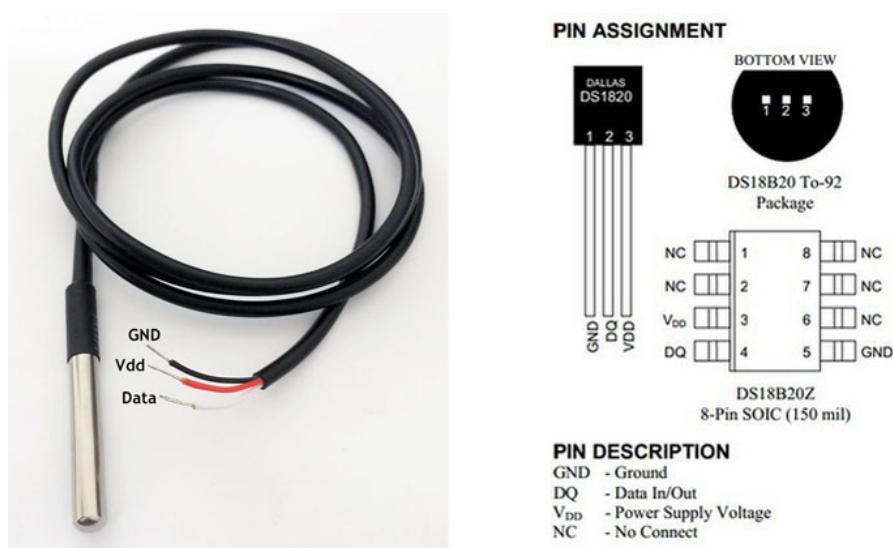
Một master và một slave

Linh kiện cần thiết:

- Board IoT Maker UnoX
- Cảm biến DS18B20 (hoặc DS18S20, hoặc DS1822)
- Màn hình OLED 0.96"
- Điện trở 4k Ω

Giới thiệu DS18B20

Đầu dò nhiệt độ sử dụng DS18B20, đầu ra digital, được tích hợp trong ống thép không rỉ, độ nhạy cao, đo chính xác nhiệt độ trong môi trường ẩm ướt, với chuẩn giao tiếp 1-Wire. DS18B20 cung cấp 9-12 bit cấu hình đọc nhiệt độ theo chuẩn giao tiếp 1-Wire, do đó chúng ta chỉ cần 1 dây để kết nối với một vi xử lý.



Hình 98. Hình ảnh cảm biến nhiệt độ DS18B20

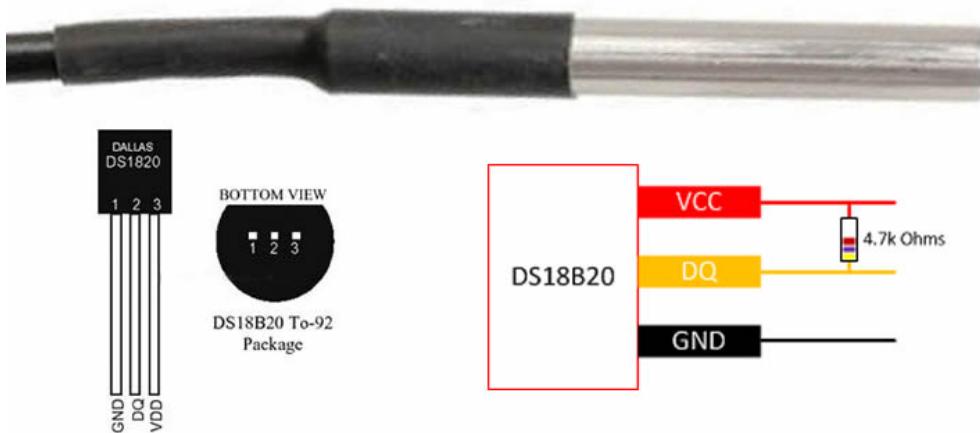
Thông tin cảm biến

- Điện áp đầu vào: 3.0-5.5V.
- Không thấm nước, không rỉ.
- Khoảng nhiệt độ đo: -55 ° C đến +125 ° C.
- Độ chính xác từ ± 0.5 ° C trong khoảng -10 ° C đến +85 ° C.
- Đầu ra : VCC (Red), DATA (Blue), GND (Black).

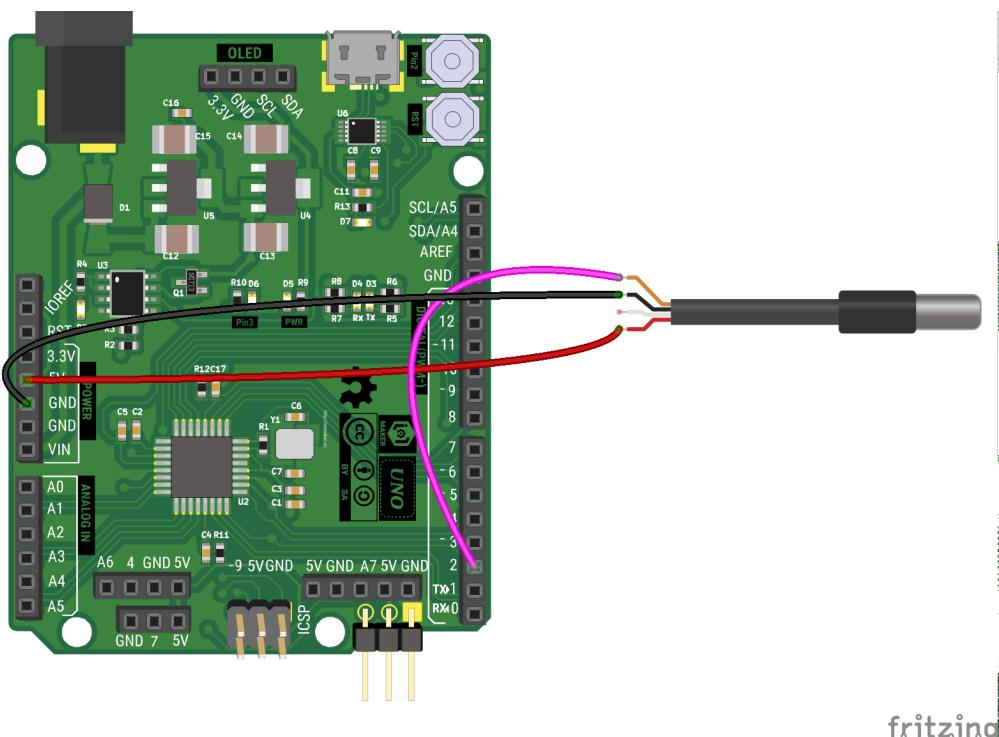
Kết nối với board IoT Maker UnoX

Bảng 16. Bảng sơ đồ kết nối cảm biến DS18B20 và board IoT Maker UnoX

DS18B20	IoT Maker UnoX
Đỏ	5V
Đen	GND
Vàng	GPIO2



Hình 99. Hình ảnh kết nối với board lotmaker Uno X



Hình 100. Hình ảnh kết nối của DSB18B20 với IoT Maker UnoX

Chú ý

- Dây data của DS18B20 cần phải được gắn với một điện trở kéo $4k7\Omega$ và được nối với nguồn 5V.
- Người dùng có thể kết nối dây vàng của cảm biến đến bất cứ chân digital nào của vi điều khiển.

Thư viện sử dụng

[OneWire](#)

[Adafruit-GFX](#)

[Adafruit_SSD1306](#)

Giải thích một số hàm trong thư viện OneWire

- OneWire myWire(pin):** Tạo một đối tượng myWire thuộc thư viện OneWire bằng một pin cụ thể. Mặc dù có thể kết nối với nhiều thiết bị trên cùng một dây, nhưng nếu có quá nhiều thì bạn có thể chia nhỏ ra và kết nối vào các pin riêng và tạo ra các đối tượng 1-Wire với từng pin.
- myWire.(addrArray):** Tìm thiết bị được kết nối tiếp. addArray là một mảng gồm 8 bit. Nếu tìm thấy một thiết bị, addArray sẽ chứa địa chỉ của thiết bị đó và sẽ trả về giá trị là **true**. Nếu không tìm thấy thiết bị nào sẽ trả về **false**.
- myWire.reset_search():** Tìm kiếm lại từ đầu, thiết bị tìm thấy sẽ là thiết bị đầu tiên.
- myWire.reset():** Reset 1-wire bus. Thường sử dụng hàm này trước khi bắt đầu giao tiếp với một thiết bị bất kỳ.

- **myWire.select(addrArray)**: Chọn một thiết bị để làm việc dựa trên địa chỉ của nó. Sử dụng lệnh này sau khi đã reset.
- **myWire.skip()**: Bỏ qua bước lựa chọn thiết bị. Cách này chỉ dùng được khi chỉ có một thiết bị slave.
- **myWire.write(num, power)**: Gửi 1 byte, và nếu sau khi bạn cần sử dụng nguồn sau khi gửi thì set power = 1, nếu không thì bỏ trống hoặc set power = 0 (đối với cảm biến DS18B20 có thể làm việc trong parasite mode – dây đỏ và đen nối chung xuống GND).
- **myWire.read()**: Đọc 1 byte.
- **myWire.crc8(dataArray, length)**: Tính bit CRC trên dữ liệu được nhận.

Mã nguồn

```
// Khai báo thư viện sử dụng
#include <OneWire.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);
#if (SSD1306_LCDHEIGHT != 32)
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif
OneWire ds(2); // Khai báo chân 2 lấy dữ liệu (cần có điện trở 4k7 để kéo lên nguồn)
// Thiết lập các giá trị ban đầu
void setup()
{
    Serial.begin(9600);
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(18, 14);
    display.println("1-Wire Example");
    display.display();
    delay(2000);
    display.clearDisplay();
}

void loop(void)
{
    byte i;
    byte present = 0;
    byte type_s; // Biến lưu giá trị xác định loại cảm biến
    byte data[12]; // Biến lưu giá trị nhiệt độ (HEX)
    byte addr[8]; // Biến lưu địa chỉ cảm biến (gồm 8 bit)
    float celsius, fahrenheit;

    if (!ds.search(addr)) {
        Serial.println("No more addresses.");
        display.clearDisplay();
        display.setCursor(0, 0);
        display.println("No more addresses.");
        display.display();
        Serial.println();
        ds.reset_search();
    }
}
```

```

delay(250);
return;
}
// In địa chỉ ROM ra Serial và OLED
Serial.print("ROM =");
display.clearDisplay();
display.setCursor(0, 0);
display.print("ROM =");
for( i = 0; i < 8; i++) {
    Serial.write(' ');
    Serial.print(addr[i], HEX);
    display.print(addr[i], HEX);
}
if (OneWire::crc8(addr, 7) != addr[7]) {
    Serial.println("CRC is not valid!");
    return;
}
Serial.println();
switch (addr[0]) { // Dùng byte đầu tiên trong ROM để xác định loại cảm biến
    case 0x10:
        Serial.println(" Chip = DS18S20");
        display.setCursor(0,8);
        display.println("Chip = DS18S20");
        type_s = 1;
        break;
    case 0x28:
        Serial.println(" Chip = DS18B20");
        display.setCursor(0,8);
        display.println("Chip = DS18B20");
        type_s = 0;
        break;
    case 0x22:
        Serial.println(" Chip = DS1822");
        display.setCursor(0,8);
        display.println("Chip = DS1822");
        type_s = 0;
        break;
    default:
        Serial.println("Device is not a DS18x20 family device.");
        return;
}

ds.reset();
ds.select(addr);
ds.write(0x44, 1);      // Gửi lệnh bắt đầu giao tiếp (đối với cảm biến DS18B20)

delay(1000);           // Cần một khoảng thời gian chờ, ít nhất là 750ms
present = ds.reset();
ds.select(addr);
ds.write(0xBE);        // Gửi lệnh bắt đầu đọc dữ liệu (đối với cảm biến DS18B20)

Serial.print(" Data =");
Serial.print(present, HEX);
Serial.print(" ");
for ( i = 0; i < 9; i++) {
    data[i] = ds.read();
    Serial.print(data[i], HEX);
    Serial.print(" ");
}
Serial.print(" CRC=");
Serial.print(OneWire::crc8(data, 8), HEX);
Serial.println();

int16_t raw = (data[1] << 8) | data[0]; // Chuyển đổi giá trị nhiệt độ từ 16 bit
if (type_s) {
    raw = raw << 3;                      // Độ phân giải mặc định 9 bit
    if (data[7] == 0x10) {
        raw = (raw & 0xFFFF) + 12 - data[6];
}

```

```

    }
} else {
    byte cfg = (data[4] & 0x60);
    // Tại độ phân giải thấp hơn, các bit thấp đều không xác định được do đó, chúng ta cho nó bằng
    if (cfg == 0x00) raw = raw & ~7;           // Độ phân giải 9 bit, 93.75 ms
    else if (cfg == 0x20) raw = raw & ~3;      // Độ phân giải 10 bit, 187.5 ms
    else if (cfg == 0x40) raw = raw & ~1;      // Độ phân giải 11 bit, 375 ms
    // Độ phân giải mặc định 12 bit, thời gian chuyển đổi 750 ms
}
celsius = (float)raw / 16.0;
fahrenheit = celsius * 1.8 + 32.0;

// In các giá trị nhiệt độ ra Serial và OLED
Serial.print(" Temperature = ");
Serial.print(celsius);
Serial.print(" Celsius, ");
Serial.print(fahrenheit);
Serial.println(" Fahrenheit");
display.setCursor(0,16);
display.print("Temp = ");
display.print(String(celsius));
display.print((char)247);display.print("C");
display.setCursor(42, 24);
display.print(String(fahrenheit));
display.print((char)247);display.print("F");
display.display();
delay(1500);
}

```

Output

Kết quả trả về trên cổng Serial trên máy tính sẽ có dạng như sau:

```

ROM = 28 FF 3C D6 63 16 4 83
Chip = DS18B20
Data = 1 DB 1 4B 46 7F FF C 10 95 CRC=95
Temperature = 29.69 Celsius, 85.44 Farenheit
No more addresses.

```

Kết quả xuất hiện trên Oled sẽ có dạng như sau:

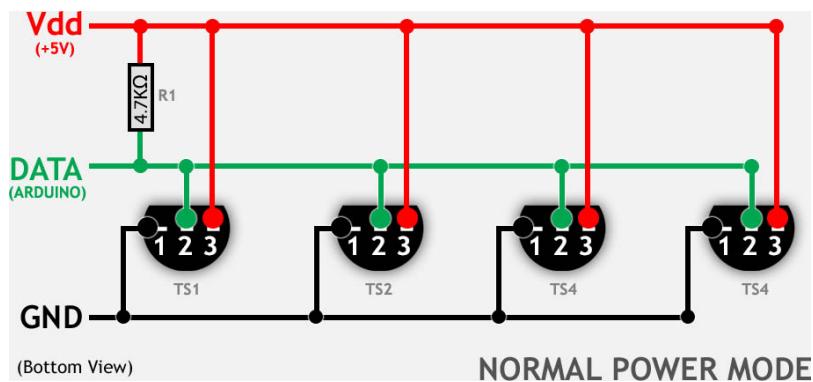
```

ROM = 28 FF 3C D6 63 16 4 83
Chip = DS18B20
Temperature = 29.69 °C
85.44 °F

```

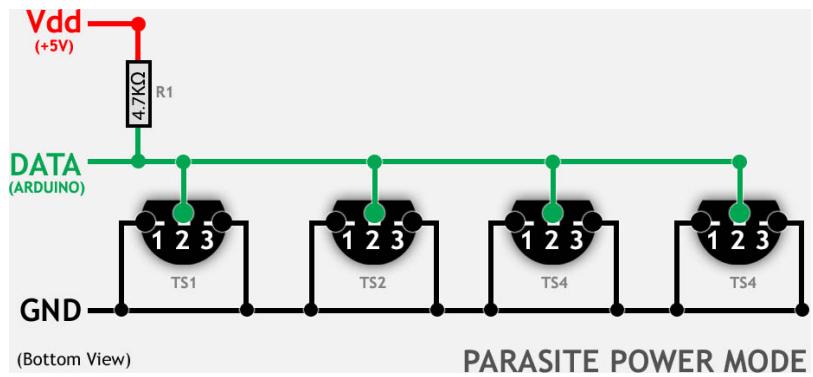
Một master và nhiều slave

Đối với ví dụ này sẽ có hai cách kết nối cảm biến như sau:



Hình 101. Normal Power Mode (Nguồn Tweeking4All.com)

Các cảm biến có thể nối song song với nhau và các dây được nối như khi sử dụng một cảm biến.



Hình 102. Parasite Power Mode (Nguồn Tweeking4All.com)

Parasite mode về cơ bản là sẽ nối dây đỏ và dây đen của cảm biến xuống GND, và chỉ có một dây vàng là được nối lên nguồn.

Ví dụ này sẽ sử dụng hai cảm biến DS18B20 kết nối theo kiểu **Normal Power** như hình.

Mã nguồn



Ở ví dụ này ta sẽ sử dụng chung một mã nguồn trên để thiết lập việc lấy dữ liệu.

Kết quả

Vì ở đây chỉ kết nối hai cảm biến nên trên OLED và Serial sẽ hiện kết quả lấy được của cả hai cảm biến.

Serial

```
ROM = 28 49 BA 3 0 0 80 B
Chip = DS18B20
Data = 1 E5 1 FF FF 7F FF FF FF D3  CRC=D3
Temperature = 30.31 Celsius, 86.56 Fahrenheit
ROM = 28 FF 3C D6 63 16 4 83
Chip = DS18B20
Data = 1 DE 1 4B 46 7F FF C 10 C3  CRC=C3
Temperature = 29.87 Celsius, 85.77 Fahrenheit
No more addresses.
```

OLED sẽ lần lượt xuất hiện.

```
ROM = 28 49 BA 3 0 0 80 B
Chip = DS18B20
Temperature = 30.31 Celsius
86.56 Fahrenheit
```

```
ROM = 28 FF 3C D6 63 16 4 83
Chip = DS18B20
Temperature = 29.87 Celsius
85.77 Fahrenheit
```

Tổng kết

Qua phần này chúng ta đã tìm hiểu được về một chuẩn giao tiếp mới là 1-Wire, và các cách lập trình sử dụng chuẩn giao tiếp này. Dưới đây là một số trang web để bạn có thể tham khảo thêm nếu muốn tìm hiểu sâu về chuẩn giao tiếp này:

[1-Wire Turorial](#)

[Overview of 1-Wire Technology and Its Use](#)

[1-Wire® Devices](#)

Timer - Interrupt

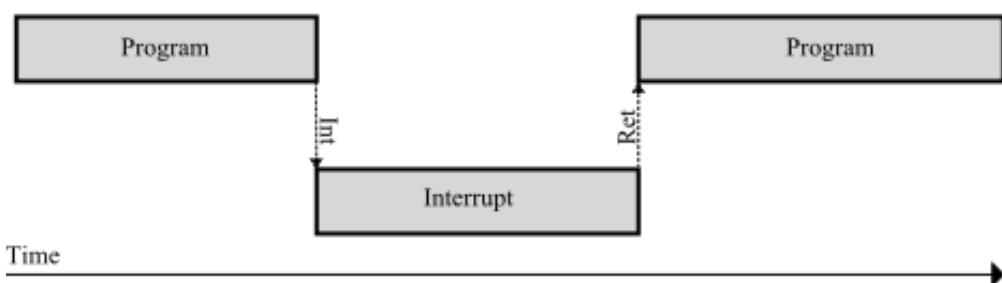
Trong chương này chúng ta sẽ tìm hiểu:

- Các khái niệm về Interrupt, Timer/Counter.
- Tại sao và khi nào cần sử dụng Interrupt, Timer/Counter.
- Cách sử dụng Interrupt, Timer/Counter.

Interrupt

Giới thiệu

Trong quá trình làm việc, bỗng nhiên bạn nhận được một yêu cầu gấp, cần phải thực hiện ngay lập tức. Lúc này, bạn buộc phải dừng công việc hiện tại để thực hiện yêu cầu đó và chỉ có thể tiếp tục công việc đang dở dang khi hoàn thành song yêu cầu. Tương tự vậy, trong lập trình, chúng ta có một khái niệm, đó chính là **ngắt (Interrupt)**.



Hình 103. Ngắt

Ngắt là khi một tín hiệu khẩn cấp được gửi tới bộ xử lý, yêu cầu bộ xử lý tạm dừng tức khắc các hoạt động hiện tại để nhảy đến một nơi khác thực hiện một nhiệm vụ khẩn cấp nào đó, nhiệm vụ này được gọi là trình phục vụ ngắt - ISR (Interrupt Service Routine). Sau khi kết thúc trình phục vụ ngắt - ISR, bộ đếm chương trình sẽ trả về giá trị trước đó để bộ xử lý quay về thực hiện chương trình đang dang dở.

Vì sao cần sử dụng ngắt

Quay trở lại ví dụ về **button** đã đề cập ở phần Hello world, chúng ta phải liên tục đọc trạng thái của nút nhấn bằng hàm **digitalRead()** trong chương trình **loop**.

```
void loop() {
    // Đọc trạng thái của nút nhấn
    buttonState = digitalRead(buttonPin);

    // Kiểm tra nếu nút được nhấn, tức buttonState ở trạng thái LOW:
    if (buttonState == LOW) {
        // Bật LED
        digitalWrite(ledPin, HIGH);
    } else {
        // Tắt LED
        digitalWrite(ledPin, LOW);
    }
}
```

Điều này sẽ cực kì khó khăn khi số lượng các câu lệnh trong chương trình **loop** tăng lên, lúc này tốc độ đáp ứng của chương trình khi chúng ta nhấn nút sẽ không còn nhanh nữa bởi vì chương trình **loop** phải thực hiện song song tất cả các lệnh rồi mới quay trở lại đọc trạng thái nút nhấn. **Ngắt** sẽ giải quyết những vấn đề này cho bạn. Lúc này, chương trình vẫn chạy liên tục những lệnh có trong chương trình **loop** và chỉ khi có ngắt xảy ra khi ta ấn nút nhấn thì chương trình mới thực hiện các lệnh trong

chương trình phục vụ ngắn và sau đó quay trở lại thực hiện tiếp chương trình trong [loop](#).

Điều này sẽ giúp tốc độ đáp ứng của chương trình đối với các thao tác của bạn được nhanh hơn và việc quản lý chương trình của bạn được dễ dàng và hiệu quả hơn.

Vector ngắn

Trong thực tế trên mỗi bộ xử lý đều có rất nhiều ngắn khác nhau, vậy điều gì sẽ xảy ra khi cùng một lúc có hai ngắn xuất hiện? Cũng như ví dụ đầu bài, nhưng bây giờ bạn nhận đến 2 yêu cầu cần thực hiện gấp, vậy bạn sẽ thực hiện yêu cầu nào? Tất nhiên là sẽ ưu tiên yêu cầu cấp thiết nhất, sau khi hoàn thành thì mới thực hiện yêu cầu tiếp theo và cuối cùng là trở lại công việc đang làm dở. Cũng như trong lập trình, mỗi ngắn đều được quy định một mức ưu tiên khác nhau, được gọi là vector ngắn (vector ngắn có giá trị càng nhỏ thì độ ưu tiên càng cao), Reset là ngắn có mức ưu tiên cao nhất.

Ngắn trong Arduino

Trong khuôn khổ của Arduino và mục tiêu của sách là đơn giản hóa mọi vấn đề để các bạn mới bắt đầu dễ dàng tiếp cận thì chúng ta sẽ không đi sâu vào vấn đề này.

Trong Arduino, chúng ta được hỗ trợ 2 loại ngắn như sau:

- **Ngắn số 0** được nối với chân số 2.
- **Ngắn số 1** được nối với chân số 3.

Để sử dụng ngắn chúng ta cần phải kết nối nút nhấn hoặc cảm biến vào hai chân này để tạo tín hiệu ngắn cho bộ xử lý.

Tùy thuộc vào vi điều khiển trên board mà mỗi dòng Arduino có số lượng các ngắn khác nhau. Bạn có thể tham khảo bảng sau:

Bảng 17. Số lượng ngắn trên các Board Arduino

Board	Int.0	Int.1	Int.2	Int.3	Int.4	Int.5
Uno. Ehternet	2	3
Mega2560	2	3	21	20	19	18
Leonardo	2	3	0	1	7	.

Arduino hỗ trợ 2 lệnh giúp ta khai báo và hủy một ngắn bất kì một cách dễ dàng đó là [attachInterrupt\(\)](#) và [detachInterrupt\(\)](#).

Lệnh attachInterrupt()

- Mục đích: Giúp khai báo một ngắn.
- Cú pháp lệnh: [attachInterrupt\[interrupt, ISR, mode\]](#).
- Các đối số:

- **interrupt:** Lựa chọn ngắt mà bạn muốn dùng, với IoT Maker UnoX có 2 lựa chọn là:
 - 0: Ứng với ngắt số 0 trong Arduino (chân số 2).
 - 1: Ứng với ngắt số 1 trong Arduino (chân số 3).
- **ISR:** Chương trình phục vụ ngắt. Chương trình này sẽ được thực hiện khi có ngắt xảy ra.
- **mode:** Kiểu kích hoạt ngắt:
 - **LOW:** Ngắt sẽ được kích hoạt khi trạng thái chân ở mức thấp.
 - **HIGH:** Ngắt sẽ được kích hoạt khi trạng thái chân ở mức cao.
 - **CHANGE:** Ngắt khi có sự thay đổi trạng thái trên chân ngắt (trạng thái thay đổi từ mức điện áp thấp lên mức điện áp cao hoặc ngược lại, từ mức điện áp cao xuống mức điện áp thấp).
 - **RISING:** Ngắt sẽ được kích hoạt khi trạng thái của chân digital **chuyển** từ mức điện áp thấp sang mức điện áp cao.
 - **FALLING:** Ngắt sẽ được kích hoạt khi trạng thái của chân digital **chuyển** từ mức điện áp cao sang mức điện áp thấp.



Trong mode LOW và HIGH, chương trình ngắt sẽ được gọi liên tục khi chân digital vẫn còn giữ ở mức điện áp thấp hoặc cao.

- Giá trị trả về: Không có giá trị trả về

Lệnh detachInterrupt()

- Mục đích: Tắt ngắt hiện tại.
- Cú pháp lệnh: **detachInterrupt(pin)**.
- Các đối số:
 - **pin:** Lựa chọn ngắt mà bạn muốn tắt, dùng board Arduino Uno có 2 lựa chọn là:
 - 0: Tắt ngắt số 0 trong Arduino (chân số 2).
 - 1: Tắt ngắt số 1 trong Arduino (chân số 3).
 - Giá trị trả về : Không có giá trị trả về.

Ví dụ

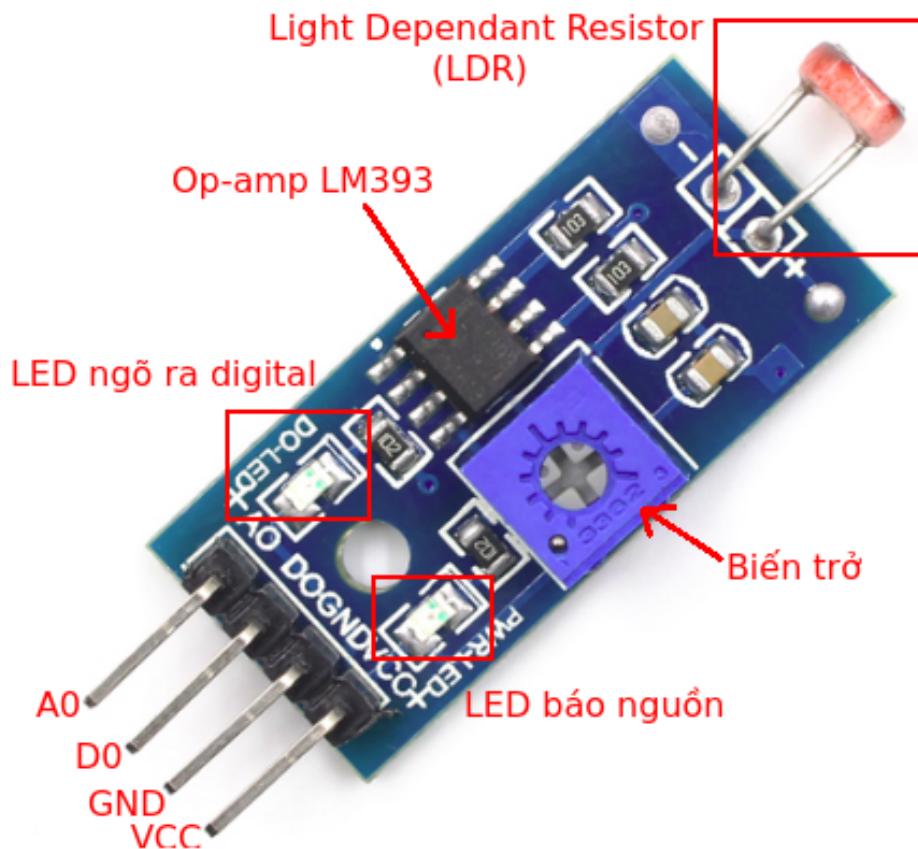
Yêu cầu

Thiết kế hệ thống tự động bật sáng đèn khi trời tối.

Linh kiện cần dùng

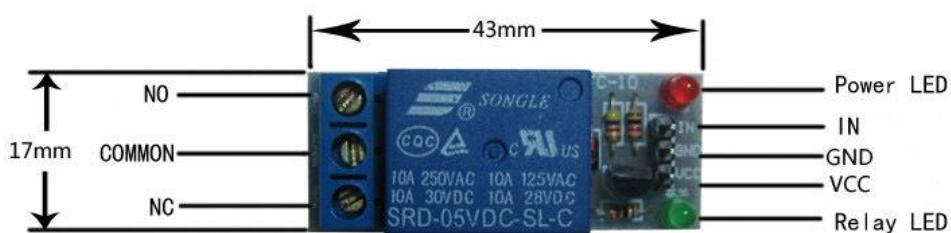
- Board IoT Maker UnoX.
- Cảm biến ánh sáng quang trở.
- Module Relay 5V 1 kênh.

Cảm biến ánh sáng quang trở



Hình 104. Cảm biến ánh sáng quang trở

Relay



Hình 105. Module Relay 5v 1 kênh.

Relay là công tắc chuyển mạch (có thể đóng cắt điện áp DC và AC) được điều khiển bằng tín hiệu DC. Đây là linh kiện thụ động rất thường gặp trong các mạch điện tử.

Relay có hai dạng phổ biến: Relay đóng mức thấp và đóng mức cao. Để hiểu thêm nguyên lý hoạt động của Relay, [Module relay và cách sử dụng](#).

Phân tích chương trình:

- Chúng ta sẽ dùng chân D0 (Digital Output) của quang trở kết nối với ngắt số 0 (chân số 2). Trong điều kiện có ánh sáng, chân D0 sẽ xuất ra mức 0 và ngược lại khi không có ánh sáng sẽ xuất ra mức 1.
- Chúng ta sẽ khai báo ngắt số 0 (chân số 2) với kiểu kích hoạt là LOW (ngắt khi trạng thái chân số 2 ở mức thấp) và chương trình phục vụ ngắt là `turnOffRelay` (để tắt Relay).
- Chương trình chính sẽ thực hiện việc bật Relay.
- Khi môi trường có ánh sáng, chân D0 của cảm biến sẽ xuất mức LOW, lúc này ngắt xảy ra và Relay sẽ tắt (đèn tắt).

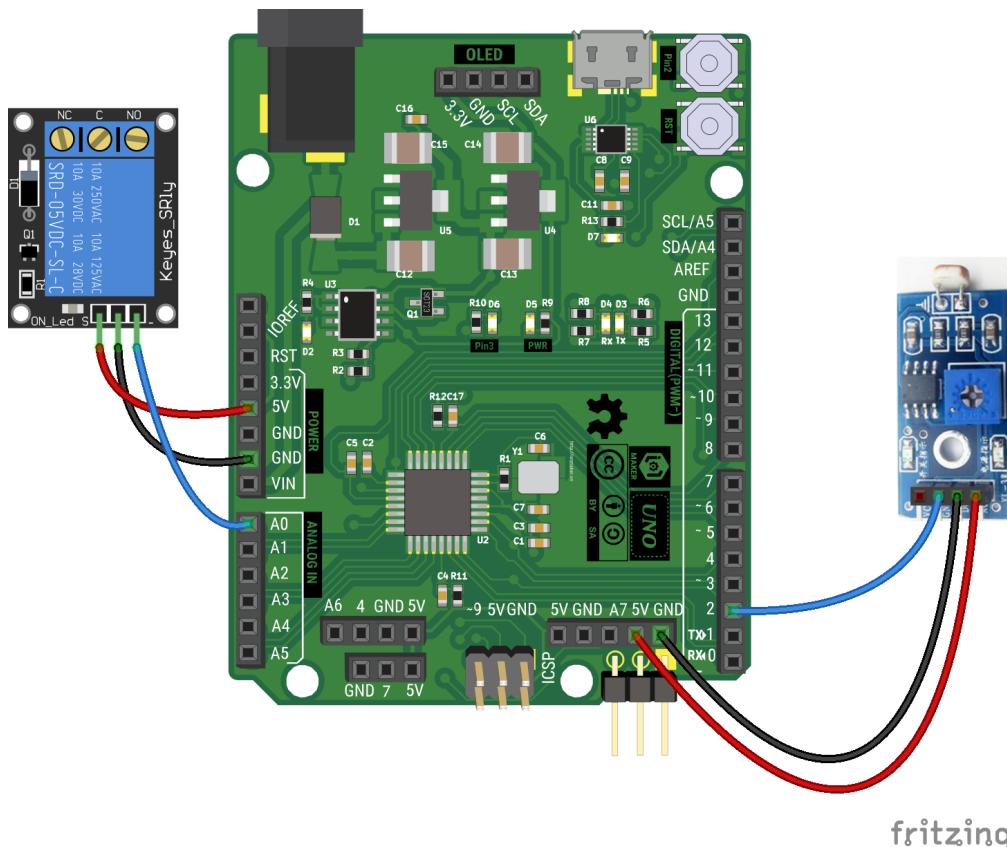
Kết nối

Bảng 18. Bảng đấu nối board IoT Maker UnoX và Relay

IoT Maker UnoX	Relay
5V	5V
GND	GND
A0	IN

Bảng 19. Bảng đấu nối board IoT Maker UnoX và Cảm biến ánh sáng

IoT Maker Uno X	Cảm biến ánh sáng
5V	5V
GND	GND
2	DO



Hình 106. Hình ảnh kết nối cảm biến ánh sáng quang trở và Relay với board IoT Maker UnoX

Source code

```

int relay = A0; // khai báo chân relay là chân A0

void turnOffRelay()
{
    digitalWrite(relay, LOW); // Tắt relay khi xảy ra ngắt.
}

void setup()
{
    pinMode(relay, OUTPUT);
    pinMode(2, INPUT_PULLUP); // khai báo chân số 2 là ngõ vào sử dụng điện trở kéo lên.
    attachInterrupt(0, turnOffRelay, LOW);
    // khai báo ngắt số 0, xảy ra khi chân số 2 ở mức thấp
    // chương trình turnOffRelay được gọi khi ngắt xảy ra.
}

void loop()
{
    digitalWrite(relay, HIGH); // bật relay
}

```



Trong ví dụ sử dụng Relay đóng mức cao. Khi sử dụng Relay đóng mức thấp thì cần phải đảo tín hiệu điều khiển.

Kết nối board IoT Maker UnoX với máy tính bằng cổng micro USB, nạp code và kiểm tra kết quả.

Timer/Counter

Giới thiệu

Timer/Counter là một trong những ngoại vi hoạt động độc lập và không thể thiếu trong bất kì vi điều khiển nào. Thực chất, Timer/Counter chỉ là một bộ đếm xung clock (có thể là xung nhịp nội bộ trong vi điều khiển hoặc xung clock bên ngoài). Nó tương tự như việc đếm giờ, thay vì ngồi cạnh đồng hồ và đếm từng giây thì chúng ta có thể đặt báo thức và làm những công việc khác, việc đếm thời gian thì giao cho đồng hồ xử lý.

Quá trình hoạt động của Timer/Counter được quản lý bởi thanh ghi, chúng gồm thanh ghi chứa giá trị Timer/Counter đếm được và thanh ghi điều khiển các hoạt động đếm của nó.

Trong khuôn khổ của sách này, chúng ta sẽ không tìm hiểu sâu về thanh ghi, nếu bạn có nhu cầu tìm hiểu rõ hơn có thể truy cập vào đường dẫn: arduino.vn/bai-viet/411-timercounter-tren-avrarduino.

Timer/Counter được sử dụng rất nhiều trong các ứng dụng định thời, đếm sự kiện, tạo xung PWM,...

Timer/Counter trong Arduino

Một số định nghĩa mà các bạn cần lưu ý trong khi sử dụng Timer/Counter:

- BOTTOM: Giá trị nhỏ nhất mà Timer/Counter có thể đạt được. Mặc định giá trị này bằng 0.
- MAX: Giá trị lớn nhất mà thanh ghi giá trị của Timer/Counter có thể chứa được. Tùy thuộc vào độ rộng của thanh ghi giá trị mà chúng ta có được giá trị MAX khác nhau. Ví dụ như Timer/Counter 0 là bộ timer 8 bit tức là giá trị tối đa (MAX) của nó là $2^8 - 1 = 255$ và đối với timer 16 bit là $2^{16} - 1 = 65535$.
- TOP: Giá trị mà khi Timer/Counter đạt đến nó sẽ thay đổi trạng thái, giá trị này phải bé hơn hoặc bằng 255 (đối với timer 8 bit) và 65535 (đối với timer 16 bit). Ví dụ: Trong Timer/Counter 0 chúng ta sẽ có giá trị MAX là 255, nếu không thiết lập giá trị TOP thì timer sẽ đếm từ 0 đến 255 và sau đó quay trở lại 0, nếu chúng ta thiết lập giá trị TOP thì timer sẽ đếm từ 0 đến giá trị TOP và quay về 0.
- Ngắt Timer: Bất cứ khi nào Timer đếm đến giá trị TOP hoặc Timer bị tràn (đếm đến giá trị MAX) thì điều xảy ra ngắt.

Chip ATMega328p (vi điều khiển của board IoT Maker UnoX) bao gồm các Timer/Counter sau:

- Timer/Counter 0: Là bộ timer 8 bit được sử dụng nhiều trong các hàm `delay()`, `millis()`, `micros()`. Chúng ta nên hạn chế sử dụng bộ timer này vì rất dễ gây ảnh hưởng đến những hàm trên.
- Timer/Counter 1: Là bộ timer 16 bit được sử dụng trong thư viện `servo`.
- Timer/Counter 2: Là một bộ timer 8 bit tương tự như Timer/Counter 0 và được sử dụng trong hàm `tone()`.



Khi sử dụng bộ Timer/Counter nào thì chúng ta cần lưu ý đến những hàm hoặc thư viện sử dụng bộ Timer/Counter đó nhằm tránh việc chương trình bị xung đột hoặc chạy không chính xác.

Thư viện TimerOne

Trên nền tảng Arduino và mục tiêu của eBook là đơn giản hóa cho những bạn mới tiếp cận, vì vậy chúng ta sẽ không điều khiển trực tiếp Timer bằng thanh ghi mà thay vào đó sẽ sử dụng thư viện [TimerOne](#) để tiếp cận với Timer/Counter.

Đúng như tên gọi, thư viện sẽ sử dụng bộ Timer/Counter 1 (16 bit), ngoài ra nếu bạn muốn tìm hiểu rõ hơn về cách sử dụng cũng như quá trình hoạt động của tất cả Timer trong board IoT Maker UnoX, bạn có thể tham khảo bài viết sau: arduino.vn/bai-viet/411-timercounter-tren-avrarduino.

Các hàm cần chú ý trong thư viện:

- **initialize[microseconds]**: Khởi động ngắt Timer với một chu kì xác định.
 - Các đối số:
 - **microseconds**: chu kỳ ngắt của Timer (tính bằng micro giây). Mặc định giá trị **microseconds = 1 000 000** tương ứng với 1s.
- **start()**: Khởi động lại Timer sau quá trình thay đổi, chỉnh sửa.
- **startBottom()**: Cho Timer đếm lại từ 0 (giá trị BOTTOM).
- **read()**: Đọc giá trị Timer đếm được ở thời điểm hiện tại.
- **stop()**: Dừng Timer.
- **attachInterrupt()**: Thêm chương trình phục vụ ngắt khi xảy ra sự kiện ngắt Timer.
- **detachInterrupt()**: Hủy hàm ngắt Timer.
- **pwm(pin, duty)**: Xuất xung PWM ra một chân xác định.
 - Các đối số:
 - **pin**: Lựa chọn chân xuất PWM.
 - **duty**: Chu kì xung PWM.
- **disablePwm(char pin)**: Hủy băm xung PWM.
 - Các đối số:
 - **pin**: Lựa chọn chân để hủy PWM.

Một số ví dụ

Điều khiển LED

Yêu cầu

Sử dụng Timer để điều khiển LED trên board nháy sau mỗi 0,15s và hiển thị số lần LED sáng lên OLED.

Linh kiện cần dùng

- Board IoT Maker UnoX.
- OLED SSD1306.

Phân tích:

- Chúng ta sẽ khai báo một Timer để định thời gian là 0,15s. Cứ sau 0,15s Timer sẽ ngắt, thực hiện chương trình `blinkLED` và đếm lại từ đầu.
- Chương trình `blinkLED` sẽ thực hiện việc đảo trạng thái của đèn LED từ sáng thành tắt và ngược lại, đồng thời đếm số lần đèn LED sáng.
- Vòng lặp `loop()` sẽ thực hiện việc hiển thị số lần LED sáng lên màn hình OLED SSD1306.

Thư viện

- Đầu tiên, chúng ta cần download thư viện TimerOne của PaulStoffregen cho Arduino. [Link download](#).

Sau khi tải thư viện về, bạn mở phần mềm arduino, chọn Sketch → Import Library... → Add Library.... Sau đó chọn file .zip mà bạn vừa tải về để có thể sử dụng thư viện.

Source code

```
#include <TimerOne.h> // thư viện cho Timer1
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

const int led = LED_BUILTIN; // LED BUILTIN chính là LED mặc định trên BOARD

void setup(void)
{
    pinMode(led, OUTPUT);
    Timer1.initialize(15000); // Khởi động ngắt Timer1 tràn mỗi 0.15s
    Timer1.attachInterrupt(blinkLED); // blinkLED to run every 0.15 seconds
    setUpOLED(); // thiết lập OLED
}

void setUpOLED()
{
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
}

int ledState = LOW;
volatile unsigned long blinkCount = 0; // biến đếm số lần Led sáng

void blinkLED(void)
{
    if (ledState == LOW) {
        ledState = HIGH;
        blinkCount = blinkCount + 1; // Tăng biến đếm mỗi khi LED sáng
    } else {
        ledState = LOW;
    }
    digitalWrite(led, ledState);
}

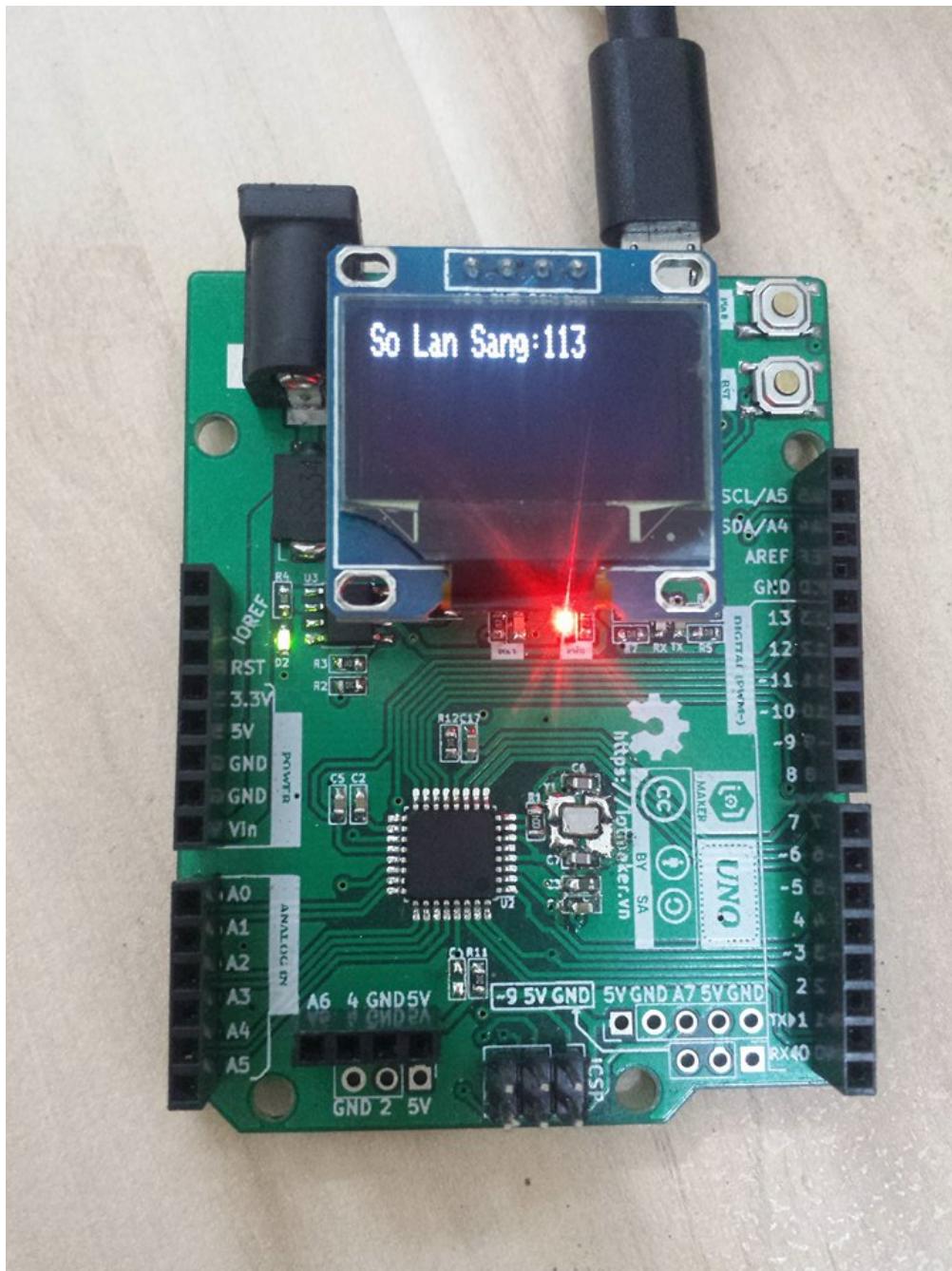
void loop(void)
{
    unsigned long blinkCopy; // biến linkCopy được dùng để copy số lần LED từ biến blinkCount
    // tắt ngắt nhằm tránh trường hợp biến blinkCount thay đổi (do ngắt timer xảy ra) trong quá trình copy.
    noInterrupts();
    blinkCopy = blinkCount;
    interrupts();

    display.clearDisplay();
    display.setCursor(0, 0);
    display.println("So Lan Sang:" + String(blinkCopy));
    display.display();
    delay(1000);
}
```

Kết nối board với máy tính bằng cổng COM, nạp chương trình và xem kết quả.



Từ khóa **volatile** được dùng cho những biến sử dụng trong chương trình phục vụ ngắt và các chương trình khác.



Hình 107. Kết quả hiển thị trên OLED

Điều khiển tốc độ quạt

Yêu cầu

Điều khiển tốc độ quạt sử dụng băm xung PWM.

Thực tế chúng ta vẫn có thể điều khiển tốc độ quạt bằng hàm `analogWrite()`, nhưng vấn đề đặt ra là hàm `analogWrite()` chỉ xuất ra xung PWM với tần số mặc định khoảng 490Hz đến 3920Hz trong khi một số quạt hiện nay yêu cầu xung PWM có tần số trong khoảng từ 21kHz đến 28kHz. Vì vậy chúng ta sẽ phải sử dụng Timer/Counter để điều chế xung PWM.

Linh kiện sử dụng

- [Board IoT Maker UnoX.](#)
- [OLED SSD1306.](#)
- Quạt mini (Fan).



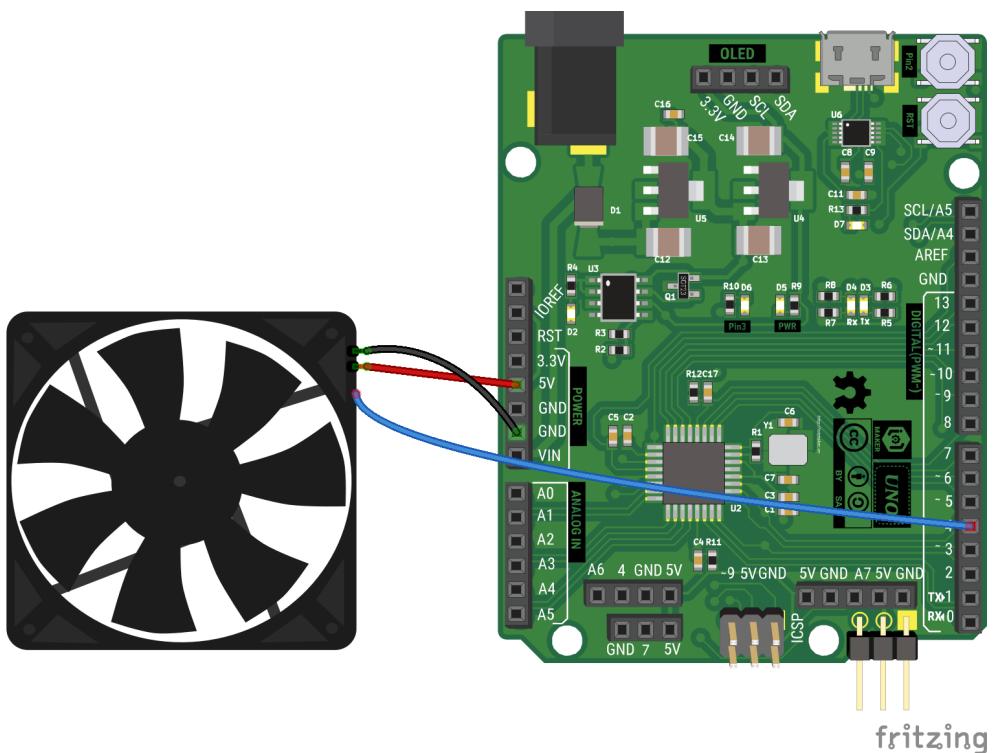
Hình 108. FAN

FAN nói chung hoặc quạt tản nhiệt nói riêng thông thường sẽ có 3 chân, chân (+) màu đỏ, chân GND màu đen và chân còn lại là chân điều khiển tốc độ.

Kết nối

Bảng 20. Bảng đấu nối board IoT Maker UnoX và quạt mini

IoT Maker UnoX	FAN
5V	5V
GND	GND
4	Xanh dương



Hình 109. Hình ảnh kết nối quạt mini với board IoT Maker UnoX

[Source code](#)

```
#include <TimerOne.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

const int fanPin = 4; //khai báo chân điều khiển tốc độ quạt

#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

void setup(void)
{
    Timer1.initialize(40); // thiết lập timer1 với chu kỳ 40us tương ứng 25kHz.
    setUpOLED(); // thiết lập OLED
}

void setUpOLED()
{
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
}

void loop(void)
{
    for (float dutyCycle = 30.0; dutyCycle < 100.0; dutyCycle++)
    {
        Timer1.pwm(fanPin, (dutyCycle / 100) * 1023);
        display.clearDisplay();
        display.setCursor(32, 0);
        display.println("PWM Fan");
        display.setCursor(0, 15);
        display.println("Duty Cycle = " + String(dutyCycle));
        display.display();
        delay(500);
    }
}
```

Kết nối board với máy tính bằng cổng COM, nạp chương trình và kiểm tra kết quả.



Hình 110. Kết quả hiển thị trên OLED

Giải thích source code:

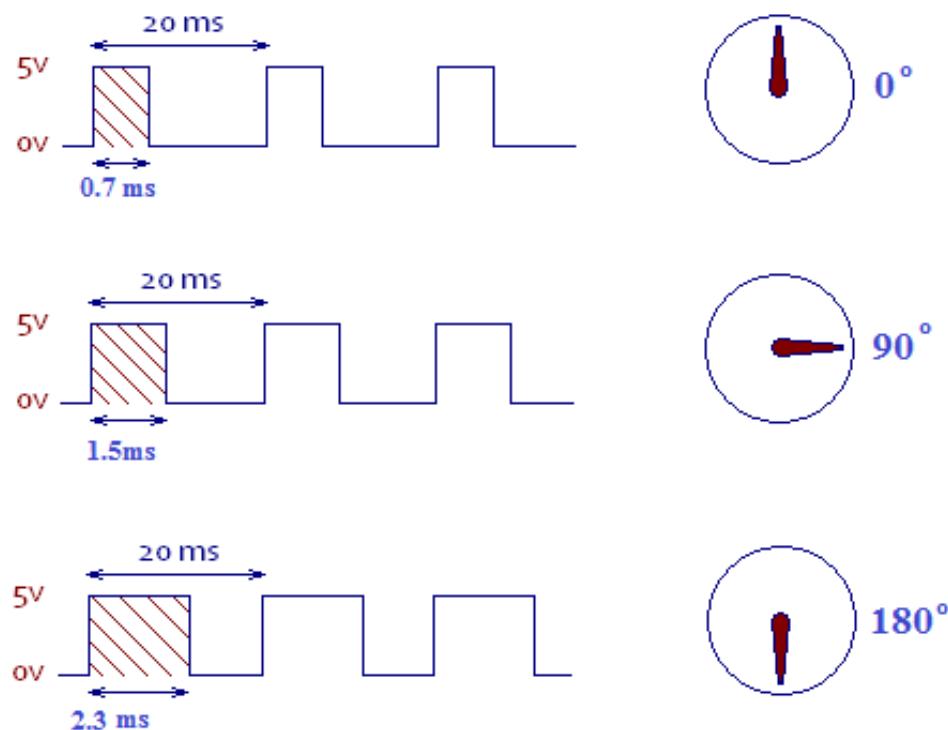
- Đầu tiên khai báo Timer để định thời gian là 40 micro giây (tương ứng với tần số PWM là $1/40 \mu\text{s} = 25\text{kHz}$) bằng lệnh `Timer1.initialize(40)`.
- Trong chương trình `loop()`, chúng ta dùng một vòng lặp for để tăng tốc độ của quạt bằng cách tăng đối số `duty` của hàm `pwm` (`duty` tăng dần theo vòng lặp for).

Điều khiển động cơ Servo bằng biến trớ

Servo là gì?

Khác với những dạng động cơ thông thường chỉ quay liên tục khi cấp nguồn điện, Servo là dạng động cơ điều khiển được góc quay bằng xung PWM. Trên thị trường có rất nhiều loại động cơ Servo với kích thước, khối lượng, cấu tạo khác nhau, từ những loại Servo kích thước nhỏ gọn (dùng cho máy bay mô hình) đến những loại sỡ hữu moment xoắn cực lớn (vài chục N/m).

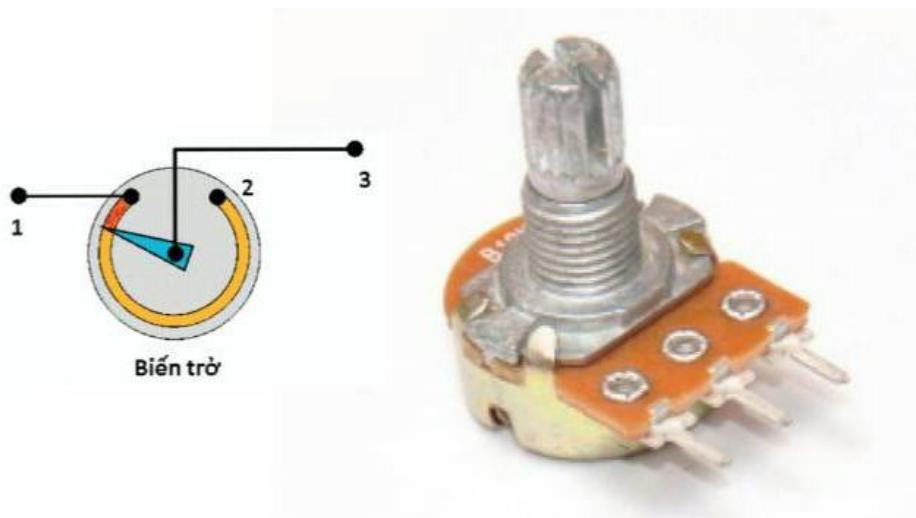
Để điều khiển được góc quay Servo (trong khoảng từ 0° - 180°) chúng ta sẽ dùng xung PWM như sau:



Hình 111. Hình điều khiển Servo

Biến trở

Biến trở là linh kiện điện tử mà giá trị điện trở có thể thay đổi được. Chúng có thể được sử dụng trong các mạch điện để điều chỉnh hoạt động của mạch điện.



Hình 112. Biến trở

Yêu cầu

- Chúng ta sẽ đọc giá trị analog từ biến trở (sử dụng biến trở để tăng giảm mức điện áp từ 0V đến 5V tương ứng với góc quay từ 0° đến 180°).
- Sau đó, chúng ta xuất ra xung PWM tương ứng ra chân số 9.

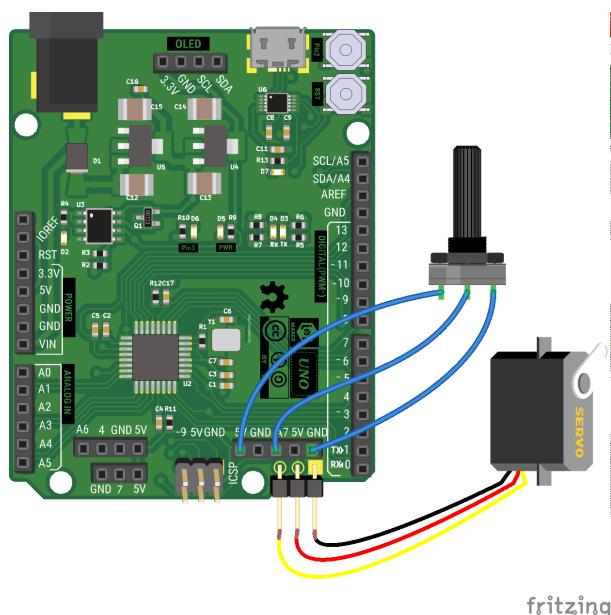
Sơ đồ kết nối

Bảng 21. Bảng đấu nối board IoT Maker UnoX với động cơ Servo

IoT Maker UnoX	Servo Motor
5V	Màu Đỏ
GND	Màu nâu
9	Màu Vàng

Bảng 22. Bảng đấu nối board IoT Maker UnoX với Biến trở

IoT Maker UnoX	Biến trở
5V	1
GND	3
A7	2



Hình 113. Hình ảnh kết nối biến trở và động cơ Servo với IoT Maker UnoX

Thư viện

Trong ứng dụng này, chúng ta sẽ sử dụng thư viện [servo.h](#)

Thư viện [servo.h](#) sử dụng bộ Timer/Counter 1, đóng vai trò quan trọng nếu bạn muốn làm về dự án robot. Nó cung cấp cho bạn một phương thức cực kì đơn giản để điều khiển động cơ Servo. Đồng thời, thư viện đã được tích hợp sẵn trong arduino IDE nên bạn không cần phải tải thêm thư viện khi sử dụng.

Một số hàm cần chú ý như:

- `myservo.attach(pin)`: Khai báo ngõ ra điều khiển Servo.
 - `pin`: lựa chọn chân ngõ ra điều khiển Servo.

- `myservo.write(deg)`: Điều khiển góc quay Servo.
 - `deg`: giá trị góc mà bạn muốn quay (0° đến 180°).

Source code

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Servo.h>      // Thư viện điều khiển servo

#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

// Khai báo đối tượng myservo dùng để điều khiển servo
Servo myservo;

int bientro = A7;          // Khai báo chân analog đọc biến trở điều khiển servo
int servoPin = 9;          // Khai báo chân điều khiển servo

void setup ()
{
    // Cài đặt chức năng điều khiển servo cho chân servoPin
    myservo.attach(servoPin);
    setUpOLED();
}

void setUpOLED()
{
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
}

void loop ()
{
    int value = analogRead(bientro); // Đọc giá trị biến trở

    // Chuyển giá trị analog (0-1023) đọc được từ biến trở sang số đo độ (0-180độ)
    // dùng để điều khiển góc quay cho servo
    int servoPos = map(value, 0, 1023, 0, 180);

    // Cho servo quay một góc là servoPos độ
    myservo.write(servoPos);
    display.clearDisplay();
    display.setCursor(20, 0);
    display.println("Control Servo");
    display.setCursor(0, 15);
    display.println("Goc quay = " + String(servoPos));
    display.display();
    delay(50);
}
```

Kết nối board với máy tính bằng cổng COM, nạp chương trình và xem kết quả.



Hình 114. Kết quả hiển thị trên OLED

Summary

Qua chương này, chúng ta đã tìm hiểu được những khái niệm cơ bản về **ngắt (Interrupt)** và **Timer/counter**, lý do tại sao chúng ta cần sử dụng nó trong các ứng dụng. Đồng thời chúng ta biết được thêm thư viện **TimerOne** giúp ta dễ dàng tiếp cận và làm chủ bộ Timer/Counter trên board IoT Maker UnoX.

Đối với những ví dụ đơn giản, khó có thể nhận thấy sự khác biệt khi sử dụng ngắt (Interrupt) và Timer/Counter. Nhưng khi thực hiện những ví dụ phức tạp, cần khai thác tối ta khả năng xử lý của vi điều khiển thì đây sẽ là trợ thủ đắc lực cho bạn.

Một số dự án tham khảo

Chương này giới thiệu những ứng dụng thực tế trong cuộc sống, những ứng dụng này hầu hết sử dụng các kiến thức điện tử cơ bản và các chuẩn giao tiếp chúng ta đã học được ở các chương trước đồng thời có thêm kiến thức về các quá trình thi công, lắp mạch trong các dự án thực tế.

Những ví dụ mẫu về ứng dụng thực tế khi sử dụng Arduino:

- + Dự án điều khiển xe mô hình thông qua chuẩn giao tiếp Bluetooth.
- + Dự án bật tắt thiết bị trong nhà thông qua chuẩn giao tiếp WiFi.

Điều khiển xe tự động bằng module Bluetooth

Trong các phần trước, chúng ta đã tìm hiểu được các giao thức truyền nhận dữ liệu giữa Arduino với các thiết bị truyền nhận thông qua các chuẩn giao tiếp có dây. Ở phần này, chúng ta sẽ tìm hiểu những kiến thức cơ bản về chuẩn giao tiếp không dây Bluetooth thông qua ứng dụng điều khiển xe tự động. Đồng thời, cũng sẽ làm quen với các kiến thức cơ bản về điều khiển bằng xung PWM, điều khiển động cơ DC.

Cơ bản về ứng dụng điều khiển xe tự động

Bạn đọc chắc hẳn khá quen thuộc với các xe đồ chơi điều khiển từ xa. Phần lớn các xe đồ chơi điều khiển từ xa giá rẻ được bán trên thị trường ngày nay đều được điều khiển bằng sóng vô tuyến thông qua các module RF. Bên cạnh sóng vô tuyến, việc điều khiển từ xa nói chung còn có thể được thực hiện thông qua các chuẩn giao tiếp không dây phổ biến khác là WiFi, Bluetooth và Zigbee.

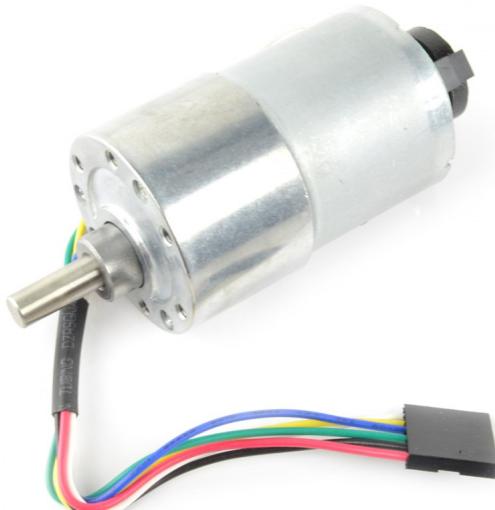
Cũng như các hệ thống điều khiển thông minh khác, các xe điều khiển từ xa có một vi điều khiển làm nhiệm vụ điều khiển chuyển động của động cơ DC, các động cơ này sẽ được nối với các bánh xe được để tạo ra chuyển động cho xe. Phần lớn các xe điều khiển ngày nay hoặc các kit phát triển đều có thêm những module điều khiển động cơ giúp cho việc cấu hình, lắp đặt sẽ đơn giản hơn cho người sử dụng. Do đó, nhiệm vụ của chúng ta là điều khiển từ xa được 4 động cơ tương ứng với 4 bánh xe.



Hình 115. Hình ảnh 1 dạng xe mô hình thường được sử dụng (Nguồn Amazon.ca)

Mở đầu về điều khiển động cơ DC

Động cơ DC hoạt động dựa trên định luật Ampere về tương tác lực từ. Cấu tạo và nguyên lý hoạt động của động cơ DC có thể tham khảo ở video: [DC motor, how it works](#)



Hình 116. Hình ảnh động cơ DC

Chiều quay của động cơ phụ thuộc vào chiều dòng điện chạy trong các khung dây dẫn. Việc đảo chiều quay của động cơ có thể thực hiện bằng cách đảo chiều dòng điện cấp vào cho động cơ. Chiều quay động cơ cũng có thể được đảo lại bằng cách thay đổi chiều của từ trường qua động cơ, tuy nhiên cách làm này không phổ biến.

Tùy từng loại động cơ DC mà người dùng lựa chọn điện áp DC cấp vào để động cơ hoạt động. Việc điều khiển động cơ khá phức tạp do việc điều khiển động cơ không phải chỉ đơn giản là cấp nguồn cho động cơ hoạt động mà phải điều khiển động cơ thông qua tín hiệu ngõ ra từ vi điều khiển. Do dòng ngõ ra của các loại vi điều khiển thường khá nhỏ để đóng/cắt các transistor trong sơ đồ động lực làm quay motor, do đó vấn đề đặt ra là phải thiết kế một mạch driver phù hợp để tín hiệu ngõ ra từ vi điều khiển có thể điều khiển động cơ.

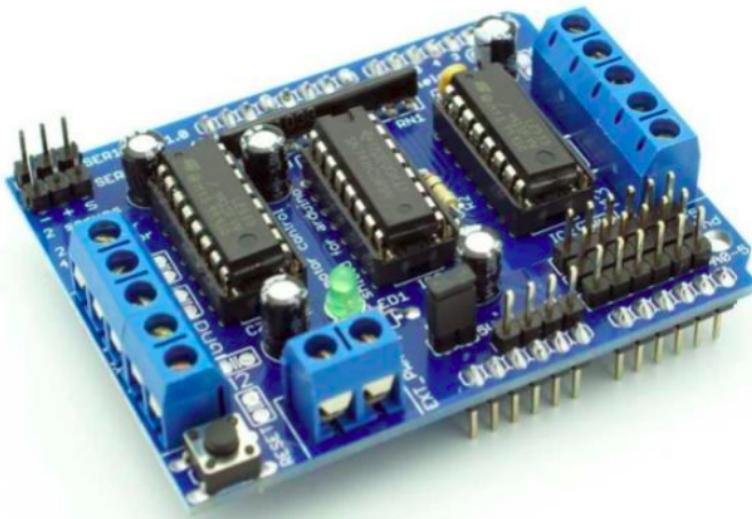
Ngày nay, nhiều module driver cho các loại động cơ như DC, Servo, động cơ bước đã được sản xuất hàng loạt và phân phối rộng rãi trên thị trường, do đó tiết kiệm nhiều thời gian cho người dùng trong việc lựa chọn, thiết kế các mạch driver, buffer.



Nếu muốn tìm hiểu chi tiết về các mạch driver, buffer trong điều khiển động cơ có thể tham khảo các kiến thức liên quan về mạch cầu H, khuếch đại tín hiệu đầu vào với transistor, các linh kiện bán dẫn liên quan như IGBT, MOSFET,...

Ở phạm vi nghiên cứu cơ bản, chúng ta chỉ xét đến việc điều khiển động cơ DC bằng xung PWM thông qua 2 module driver là L293D và L298N.

Điều khiển động cơ với module L293D



Hình 117. Driver Motor Shield L293D

Driver Motor Shield L293D là một module mở rộng chuyên dụng cho các ứng dụng điều khiển động cơ DC, động cơ Servo, động cơ bước. Module sử dụng IC L293D, phù hợp với việc cắm trực tiếp vào các shield Arduino. Thông tin kỹ thuật của module có thể tham khảo tại: iotmaker.vn/driver-motor-shield-293d.html.

Driver Motor Shield L293D có kích thước vừa với module IoT Maker UnoX. Các tín hiệu điều khiển từ board IoT Maker UnoX có thể truyền trực tiếp đến shield L293D và có thể lấy tín hiệu ngõ ra từ shield để điều khiển động cơ.

Yêu cầu

Chương trình điều khiển động cơ DC quay thuận, quay nghịch và tắt tự động cơ theo chu kỳ.

Đấu nối

Bảng đấu nối board IoT Maker UnoX và shield L293D

Để điều khiển được động cơ với shield L293D, cần cài đặt thêm thư viện [AFMotor](#) của Adafruit. Với động cơ DC, thư viện hỗ trợ các tính năng điều khiển mặc định là: FORWARD, BACKWARD và RELEASE tương ứng với các thao tác tiến (quay thuận), lùi (quay ngược) và dừng động cơ.

```
#include <AFMotor.h>

AF_DCMotor motor(1, MOTOR12_64KHZ); // Khởi tạo động cơ 1 với pwm 64KHz

void setup()
{
    Serial.begin(9600);
    Serial.println("Control DC motor");

    motor.setSpeed(255);
}

void loop()
{
    Serial.println("Run forward"); // Chạy tiến
    motor.run(FORWARD);

    delay(1000);

    Serial.println("Run backward");// Chạy lùi
    motor.run(BACKWARD);

    delay(1000);

    Serial.println("Stop"); // Dừng
    motor.run(RELEASE);

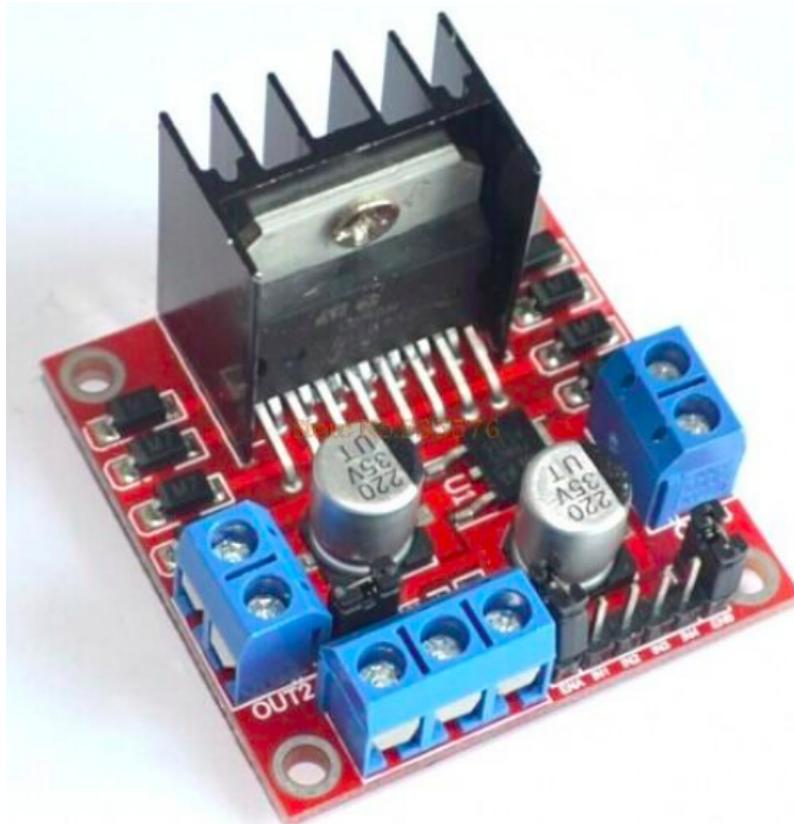
    delay(1000);
}
```

Giải thích

Chương trình trên sẽ cho motor quay thuận 1s, sau đó quay nghịch 1s và dừng 1s. Sau đó, chương trình sẽ lặp lại liên tục.

- Lệnh AF_DCMotor motor(1, MOTOR12_64KHZ): Khởi tạo đối tượng "motor" là ngõ ra motor 1 trên shield L293D dùng để điều khiển động cơ với tần số PWM là 64KHz.
- Lệnh motor.setSpeed(255): Đặt tốc độ tối đa cho động cơ. Cũng giống như hàm analogWrite(<pin>, 255), khi đó chân <pin> sẽ có duty cycle là $255/255 = 100\%$, tức là toàn bộ xung là điện áp mức 1.
- Chiều quay động cơ trong thư viện AFMotor tương ứng với 3 thuộc tính mặc định là FORWARD (tiến/quay thuận), BACKWARD(lùi/quay nghịch), RELEASE (dừng).
- Hàm motor.run() sẽ có 3 đối số truyền vào là FORWARD, BACKWARD và RELEASE tương ứng với việc cho motor quay thuận, quay nghịch và dừng.

Điều khiển động cơ với module L298N

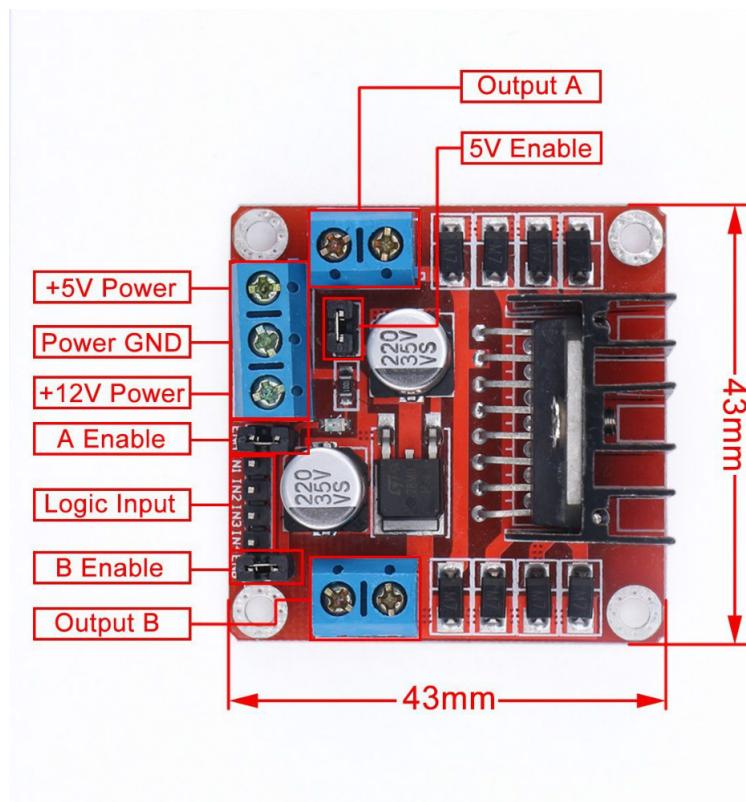


Hình 118. L298N Dual H-Bridge Module

L298N Dual H-Bridge Module là một module giúp người dùng điều khiển động cơ DC dễ dàng, ngoài ra module cũng có thể điều khiển được một động cơ bước lưỡng cực. Module có cấu tạo gồm một driver L298N tích hợp 2 mạch cầu H. Thông tin chi tiết về module L298N Dual H-Bridge có thể tham khảo tại: [Module cầu H L298N](#).

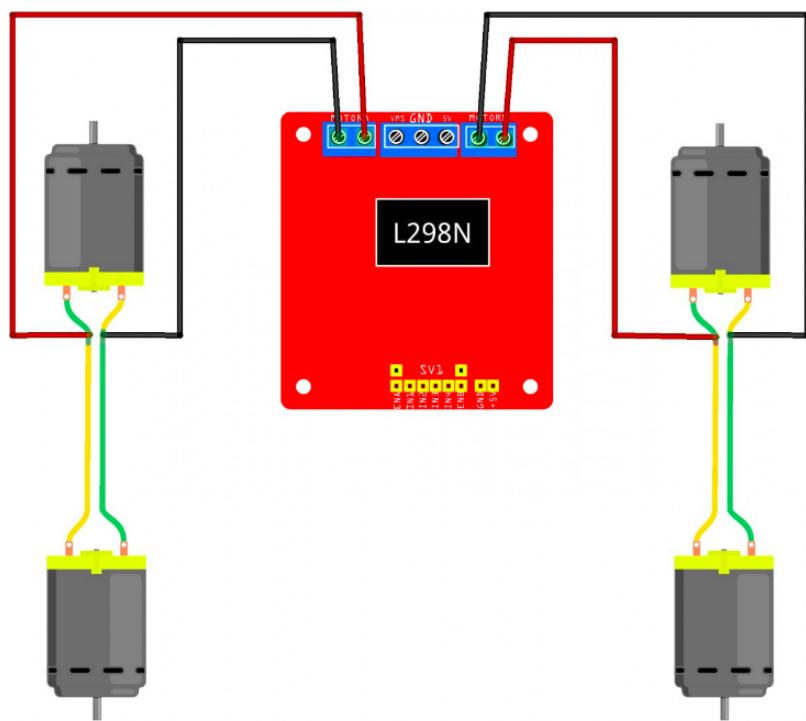
Để sử dụng module, người dùng không cần dùng thêm thư viện hỗ trợ nào do các hàm mặc định của Arduino đã đủ để điều khiển. Người dùng chỉ cần kết nối đúng các chân điều khiển và nạp chương trình thì đã có thể điều khiển được motor. Với module L298N, để motor quay thuận, ta chỉ cần ghi điện áp mức 1 vào một cực của motor và mức 0 vào cực còn lại, để đảo chiều, ta chỉ cần đảo thứ tự ghi mức điện áp.

Các chân của L298N



Hình 119. Hình ảnh các chân của L298 (nguồn robotics.org.za)

- +12V Power và +5V Power là 2 chân cắm nguồn cho motor và nguồn từ pin sẽ được cấp vào 2 chân này.
- Có thể cấp từ 9-12VDC cho chân +12V Power và khi cấp 9-12VDC vào chân +12V chân +5V Power sẽ là chân output ra 5VDC cấp cho Arduino.
- GND là chân GND của nguồn cấp cho motor.
- Nếu sử dụng nguồn 5VDC của chân +5V Power cấp cho Arduino thì GND của Arduino sẽ được nối với chân này.
- 2 Jump A enable và B enable là 2 chân giúp OUTPUT A và OUTPUT B hoạt động, nếu bạn không muốn motor nào hoạt động thì rút jump của OUTPUT motor đó ra.
- 4 chân Input IN1, IN2, IN3, IN4 là các chân tín hiệu nhận từ arduino để điều khiển tốc độ quay, quay thuận hoặc quay nghịch.
- OUTPUT A nối với động cơ A, OUTPUT B nối với động cơ B, nếu bạn nối ngược chiều thì động cơ sẽ chạy ngược, thấy motor chạy ngược thì bạn chỉ cần nối lại thôi.



Hình 120. Hình ảnh kết nối module L298N để điều khiển 4 động cơ DC (Nguồn custom-build-robots.com)

Chương trình cho động cơ DC quay thuận, quay nghịch và tắt tự động theo chu kỳ

```

#define IN1 8 // Chân kết trên board Iotmaker Uno X kết nối với module L298
#define IN2 9 // Chân kết trên board Iotmaker Uno X kết nối với module L298

#define MAX_SPEED 255
#define MIN_SPEED 0

void setup()
{
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
}

void stop()
{
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
}

// Động cơ quay thuận (đi tới)
void forward(int speed) {
    speed = constrain(speed, MIN_SPEED, MAX_SPEED); // Cố định khoảng giá trị của speed trong khoảng từ MIN_SPEED tới MAX_SPEED
    digitalWrite(IN1, HIGH); // digitalWrite() thường dùng cho chân không có PWM
    analogWrite(IN2, 255 - speed); // Dùng cho chân có PWM, ở đây là chân 9
}

// Động cơ quay nghịch (đi lùi). Để quay nghịch, ta chỉ cần đảo chiều dòng điện cấp vào động cơ DC
void backward(int speed)
{
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN1, LOW);
    analogWrite(IN2, speed);
}

void loop()
{
    forward(MAX_SPEED);
    delay(5000);
    backward(MAX_SPEED);
    delay(5000);
    stop();
    delay(5000);
}

```

Xe điều khiển từ xa với 4 động cơ DC

Từ kiến thức về điều khiển 1 động cơ DC cơ bản đã tìm hiểu ở các phần trên. Tiếp theo, chúng ta sẽ tìm hiểu việc vận hành đơn giản với một xe điều khiển từ xa dùng 4 động cơ DC tương ứng với 4 bánh. Một ứng dụng điều khiển xe hoàn chỉnh với các tính năng chạy tiến, lùi, rẽ trái, rẽ phải thì mỗi động cơ DC ứng với mỗi bánh phải có một ngõ ra tương ứng trên shield driver. Chúng ta sẽ dùng 2 module L298N để điều khiển cả 4 động cơ DC của xe mô hình với thư viện "AFMotor.h" như đã trình bày ở trên. Ta chỉ cần kết nối 4 ngõ ra của 2 module đến 4 động cơ DC tương ứng, chú ý thứ tự IN-OUT tương ứng để điều khiển đúng chiều quay động cơ. Một số khung xe mô hình thường được sử dụng có thể tham khảo tại: [Các sản phẩm robot và drone](#).



Để xe tiến hoặc lùi, cách đơn giản nhất là cho 4 động cơ cùng quay một chiều thuận hoặc nghịch; để dừng, ta cho cả 4 động cơ dừng chạy.



Lưu ý: chương trình dưới đây tương ứng với cách kết nối dây trong khung xe dùng cho chương trình này. Bạn đọc tham khảo chương trình này và chỉnh sửa lại cho phù hợp với kết nối dây trong khung xe của mình.

Chương trình cho xe chạy tiễn 5s, chạy lùi 5s, sau đó dừng 1s rồi tiếp tục lặp lại quá trình

```
// Khai báo các chân kết nối với board Iotmaker Uno X
#define IN1 7
#define IN2 6
#define IN3 5
#define IN4 4
#define IN5 11
#define IN6 10
#define IN7 9
#define IN8 8
// Khai báo tốc độ maximum và minimum để cài đặt trong chương trình
#define MAX_SPEED 255
#define MIN_SPEED 0

void setup()
{
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
    pinMode(IN5, OUTPUT);
    pinMode(IN6, OUTPUT);
    pinMode(IN7, OUTPUT);
}
// Các hàm dùng
void motor_stop()
{
    // Motor 1
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
    // Motor 2
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    // Motor 3
    digitalWrite(IN5, LOW);
    digitalWrite(IN6, LOW);
    // Motor 4
    digitalWrite(IN7, LOW);
    digitalWrite(IN8, LOW);
}

void motor_forward(int speed)
{
    // Motor 1
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN1, HIGH);
    analogWrite(IN2, 255 - speed);
    // Motor 2
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN4, HIGH);
    analogWrite(IN3, 255 - speed);
    // Motor 3
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN5, HIGH);
    analogWrite(IN6, 255 - speed);
    // Motor 4
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN8, HIGH);
    analogWrite(IN7, 255 - speed);
}
```

```

}

void motor_backward(int speed)
{
    // Motor 1
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN1, LOW);
    analogWrite(IN2, speed);
    // Motor 2
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN4, LOW);
    analogWrite(IN3, speed);
    // Motor 3
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN5, LOW);
    analogWrite(IN6, speed);
    // Motor 4
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN8, LOW);
    analogWrite(IN7, speed);
}

void loop()
{
    motor_forward(MAX_SPEED);
    delay(5000);

    motor_backward(MAX_SPEED);
    delay(5000);

    motor_stop();
    delay(1000);
}

```



Một vấn đề có thể gặp phải khi chạy chương trình trên là sẽ có ít nhất một bánh xe quay ngược chiều với các bánh còn lại. Để quay đồng bộ chỉ cần đổi thứ tự dây cắm ngõ vào IN tương ứng với motor đó.

Điều khiển xe từ xa bằng Bluetooth

Bluetooth

Bluetooth là một chuẩn kết nối không dây tầm ngắn sử dụng sóng ở tần số 2.4 GHz để kết nối các thiết bị như điện thoại, laptop, máy tính bảng,... với nhau. Khi 2 thiết bị được kết nối với nhau bằng Bluetooth, người dùng có thể chia sẻ dữ liệu của 2 thiết bị đó với nhau.

Bluetooth Low Energy (BLE)

Bluetooth Low Energy (viết tắt là BLE) là một công nghệ Bluetooth tiết kiệm năng lượng hay Bluetooth năng lượng thấp. BLE còn được gọi là "Smart Bluetooth" vì BLE có thể kết nối giao tiếp được với các smartphone được hỗ trợ với các hệ điều hành khác nhau như iOS, Android, Windows Phone và BlackBerry, cũng như macOS, Linux, Windows 8 và 10. So với Bluetooth Classic thì BLE giảm đáng kể về năng lượng tiêu thụ. Tuy nhiên, BLE không thể thay thế hoàn toàn các chuẩn truyền không dây khác

nhiều WiFi, Zigbee, LoRa,... và cả Bluetooth Classic.

BLE hoạt động ổn định trong phạm vi 10m với dữ liệu truyền tải không lớn.

Pairing

Từ định nghĩa Bluetooth ở trên, ta thấy rằng nếu có nhiều hơn 2 thiết bị, ví dụ như nhiều điện thoại thông minh (smartphone) và nhiều thiết bị khác cùng kết nối Bluetooth thì tất cả sẽ không nhận biết nhau được. Do đó, một việc rất quan trọng mà chúng ta cần thực hiện đầu tiên là đảm bảo cho 2 thiết bị kết nối được với nhau. Việc này gọi là Pairing (ghép đôi).

Giới thiệu các module Bluetooth

Một số module Bluetooth được sử dụng rộng rãi trong việc điều khiển thiết bị là [Module Bluetooth HC-06](#) và [Module Bluetooth BT16 4.2](#) bởi sự ổn định khi truyền nhận dữ liệu và giá thành hợp lý.

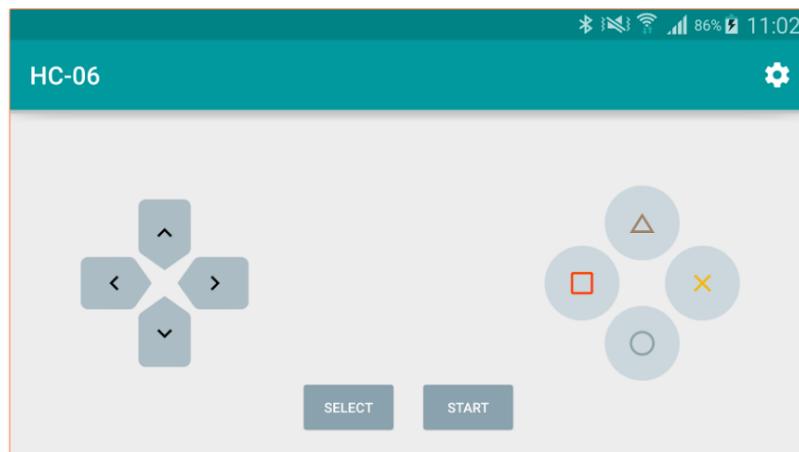
Module Bluetooth HC-06



Hình 121. Module Bluetooth HC-06

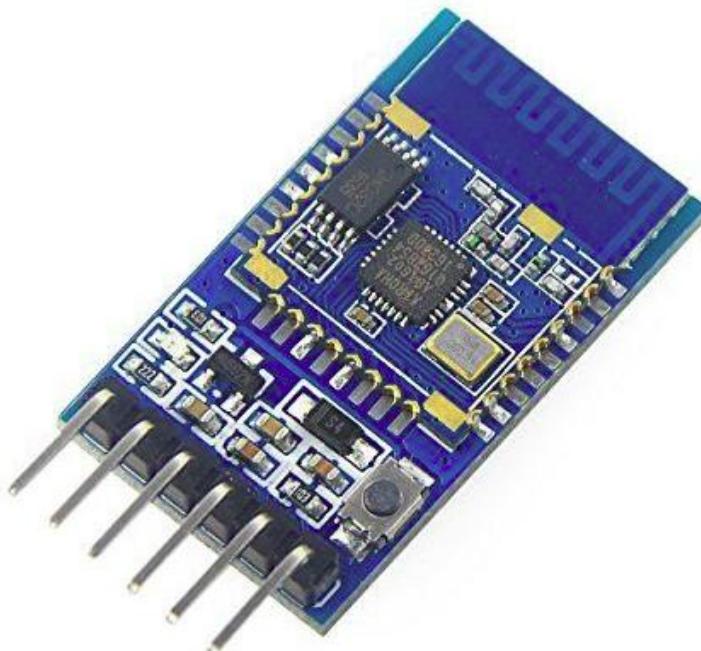
Module Bluetooth HC-06 là một module thuộc kiểu truyền thống (Bluetooth Classic) hỗ trợ giao tiếp với các vi điều khiển ở tần số 2.4Ghz. Sau khi kết nối module đến vi điều khiển, module này sẽ tự động phát Bluetooth, các thiết bị như smartphone, laptop,... sẽ dò được tên module là HC-06, tuy nhiên các thiết bị không ghép đôi được với module. Chỉ có một số ứng dụng Bluetooth mới ghép đôi và truyền nhận dữ liệu được với module. Người dùng có thể tìm được nhiều ứng dụng điều khiển khác nhau trên Google Play, tài liệu này sử dụng ứng dụng: [Arduino bluetooth controller](#) trên Google Play.

Ứng dụng này có hỗ trợ tính năng phím điều khiển, do đó tiện lợi cho các tính năng điều khiển xe tiến, lùi, rẽ trái, rẽ phải.



Hình 122. Hình ảnh giao diện của ứng dụng Arduino Bluetooth Controller

Module Bluetooth BT16 4.2



Hình 123. Module Bluetooth BT16 4.2

[Module Bluetooth BT16 4.2](#) là một module Bluetooth tiết kiệm năng lượng (BLE) sử dụng tần số 2.4GHz. Việc sử dụng module cũng hoàn toàn tương tự như module HC-06. Tuy nhiên, để tìm kiếm 1 ứng dụng trên điện thoại để kết nối và truyền nhận dữ liệu như với module HC-06 là khá khó khăn do cộng đồng BLE vẫn còn khá ít và chưa hỗ trợ nhiều về các ứng dụng này. Do đó, các ứng dụng điện thoại để truyền nhận dữ liệu tới module hiện nay đều sử dụng việc truyền dữ liệu thủ công - tức là người dùng phải nhập vào ký tự cần truyền và nhấn gửi để gửi đến module Bluetooth BT16.

Một ứng dụng kết nối và truyền nhận dữ liệu với module BT16 được sử dụng khá phổ biến là [nRF Connect](#). Đây là một công cụ để phát hiện các thiết bị BLE và thực hiện kết nối tới các thiết bị đó. Sau

khi kết nối thành công, người dùng có thể gửi dữ liệu đến module BLE đã kết nối.

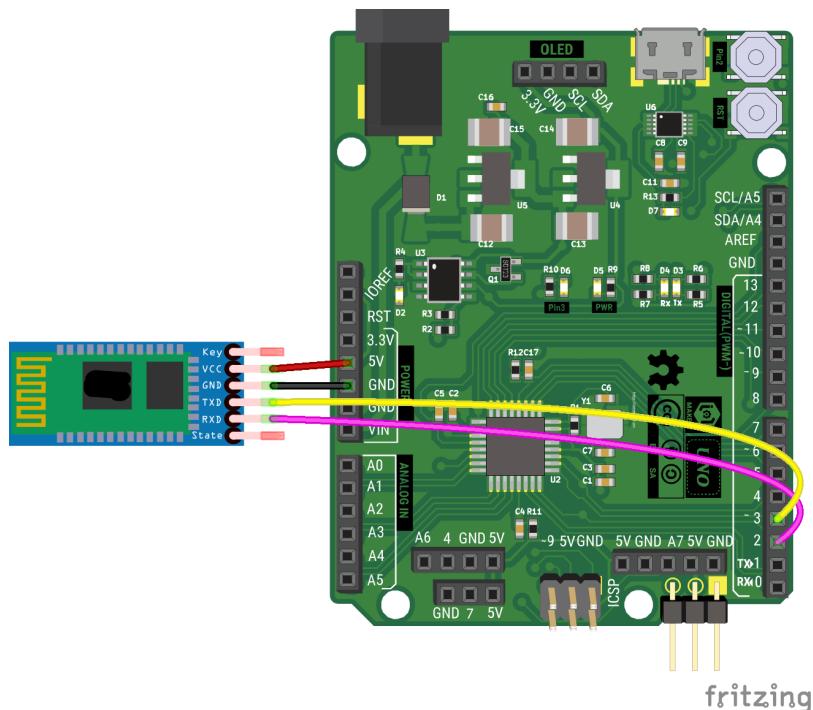
Điều khiển xe từ xa với Bluetooth

Với module BT16 4.2

Kết nối

Bảng 23. Kết nối IoT Maker UnoX với module BT16 4.2

IoT Maker UnoX	BT16 4.2
5V	VCC
GND	GND
3	TX
2	RX



Hình 124. Hình ảnh module Bluetooth BT16 4.2 với IoT Maker UnoX

Từ chương trình xe điều khiển chạy tự động ở phần trước, ta thêm một số dòng lệnh với thư viện SoftwareSerial để điều khiển motor.

Source code

```
#include <SoftwareSerial.h>
// Khai báo các chân kết nối với board Iotmaker Uno X
#define IN1 7
#define IN2 6
#define IN3 5
#define IN4 4
#define IN5 11
#define IN6 10
```

```

#define IN7 9
#define IN8 8
// Khai báo tốc độ maximum và minimum để cài đặt trong chương trình
#define MAX_SPEED 255
#define MIN_SPEED 0

SoftwareSerial mySerial(2, 3); // 2 chân 2,3 tương ứng với RX,TX
int state; // Biến lưu dữ liệu đọc từ mySerial

void setup()
{
    Serial.begin(9600);
    mySerial.begin(9600); // Khởi tạo Serial đã kết nối với chân 2 và 3 nhằm đọc dữ liệu
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
    pinMode(IN5, OUTPUT);
    pinMode(IN6, OUTPUT);
    pinMode(IN7, OUTPUT);

    pinMode(12, OUTPUT);
    pinMode(13, OUTPUT);

    digitalWrite(12, HIGH);
    digitalWrite(13, HIGH);
}

void motor_stop()
{
    // Motor 1
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
    // Motor 2
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    // Motor 3
    digitalWrite(IN5, LOW);
    digitalWrite(IN6, LOW);
    // Motor 4
    digitalWrite(IN7, LOW);
    digitalWrite(IN8, LOW);
}

void motor_forward(int speed)
{
    // Motor 1
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN1, HIGH);
    analogWrite(IN2, 255 - speed);
    // Motor 2
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN4, HIGH);
    analogWrite(IN3, 255 - speed);
    // Motor 3
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN5, HIGH);
    analogWrite(IN6, 255 - speed);
    // Motor 4
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN8, HIGH);
    analogWrite(IN7, 255 - speed);
}

void motor_backward(int speed)
{
    // Motor 1
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);

```

```

digitalWrite(IN1, LOW);
analogWrite(IN2, speed);
// Motor 2
speed = constrain(speed, MIN_SPEED, MAX_SPEED);
digitalWrite(IN4, LOW);
analogWrite(IN3, speed);
// Motor 3
speed = constrain(speed, MIN_SPEED, MAX_SPEED);
digitalWrite(IN5, LOW);
analogWrite(IN6, speed);
// Motor 4
speed = constrain(speed, MIN_SPEED, MAX_SPEED);
digitalWrite(IN8, LOW);
analogWrite(IN7, speed);
}

void loop()
{
    if (mySerial.available() > 0) { // Nếu có dữ liệu nhận được từ mySerial
        state = mySerial.read(); // Đọc dữ liệu nhận được và lưu vào biến state
        Serial.println(state); // In ra màn hình serial dữ liệu nhận được
    }
    if (state == '1') // Nếu nhận được số 1, cho xe chạy tới
        motor_forward(MAX_SPEED);
    else if (state == 2) // Nếu nhận được số 2, cho xe chạy lùi
        motor_backward(MAX_SPEED);
    else if (state == 3) // Nếu nhận được số 3, cho xe dừng
        motor_stop();
    delay(1000);
}

```

Giải thích code

Chương trình trên sử dụng thư viện SoftwareSerial nhằm sử dụng chân số 2 và chân số 3 tương ứng với RX, TX để truyền, nhận dữ liệu với module Bluetooth. Với chương trình trên, ta sử dụng app [nRF Connect](#) để gởi các giá trị "1", "2", "3" đến module Bluetooth, module sẽ truyền giá trị nhận được cho board IoT Maker UnoX thông qua UART, từ đó board IoT Maker UnoX sẽ căn cứ vào giá trị nhận được để điều khiển các motor.

- Dữ liệu nhận được là 1 thì 4 motor quay thuận (xe chạy thẳng).
- Dữ liệu nhận được là 2 thì 4 motor quay nghịch (xe chạy lùi).
- Dữ liệu nhận được là 3 thì 4 motor dừng (xe dừng).

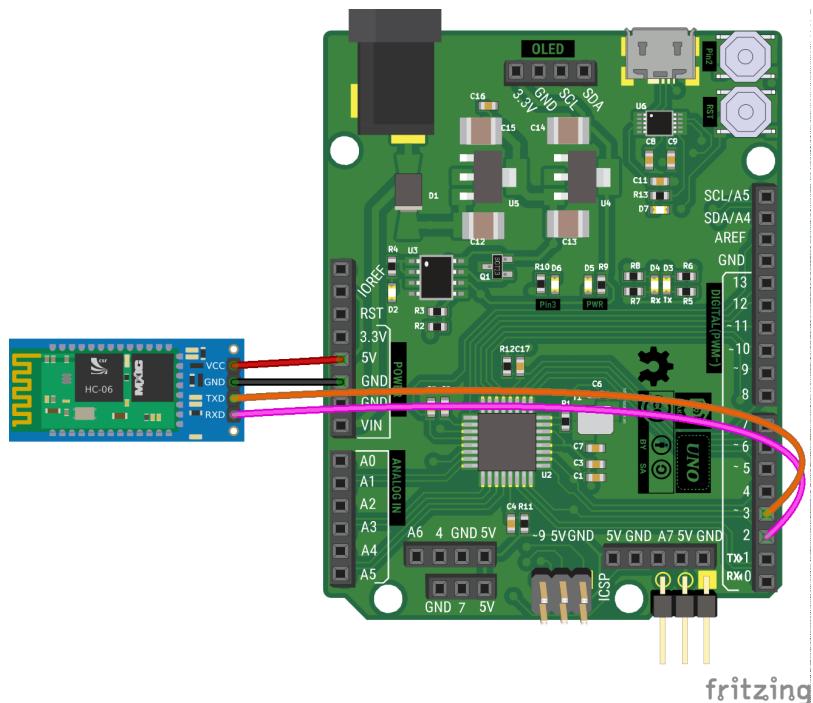
Với module HC-06

Khác với module BLE BT16, các module Bluetooth Classic có nhiều app hỗ trợ với các tính năng nút nhấn tiện lợi. Mỗi nút nhấn tương ứng với việc gởi dữ liệu tương ứng đến module Bluetooth.

Bảng 24. Kết nối board IoT Maker UnoX với module HC-06

IoTmaker Uno X	HC-06
5V	VCC
GND	GND
3	TX

Iotmaker Uno X	HC-06
2	RX



Hình 125. Hình ảnh module Bluetooth HC-06 với IoT Maker UnoX

Source code

Cũng giống như chương trình với module BT16, chương trình điều khiển xe từ xa với module bluetooth classic HC-06 hỗ trợ thêm tính năng xoay trái, xoay phải tương ứng với các nút điều khiển trên app.

```
#include <SoftwareSerial.h>
// Khai báo các chân kết nối với board Iotmaker Uno X
#define IN1 7
#define IN2 6
#define IN3 5
#define IN4 4
#define IN5 11
#define IN6 10
#define IN7 9
#define IN8 8
// Khai báo tốc độ maximum và minimum để cài đặt trong chương trình
#define MAX_SPEED 255
#define MIN_SPEED 0

SoftwareSerial mySerial(2, 3); // 2 chân 2,3 tương ứng với RX,TX
char state = '0'; // Biến lưu dữ liệu đọc từ mySerial

void setup()
{
    Serial.begin(9600);
    mySerial.begin(9600); // Khởi tạo Serial đã kết nối với chân 2 và 3 nhằm đọc dữ liệu
    // Cài đặt hướng ngõ ra cho các chân điều khiển động cơ
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
    pinMode(IN5, OUTPUT);
    pinMode(IN6, OUTPUT);
```

```
pinMode(IN7, OUTPUT);
pinMode(12, OUTPUT);
pinMode(13, OUTPUT);

digitalWrite(12, HIGH);
digitalWrite(13, HIGH);
}

void motor_stop()
{
    // Motor 1
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
    // Motor 2
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    // Motor 3
    digitalWrite(IN5, LOW);
    digitalWrite(IN6, LOW);
    // Motor 4
    digitalWrite(IN7, LOW);
    digitalWrite(IN8, LOW);
}

void motor_forward(int speed)
{
    // Motor 1
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN1, HIGH);
    analogWrite(IN2, 255 - speed);
    // Motor 2
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN4, HIGH);
    analogWrite(IN3, 255 - speed);
    // Motor 3
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN5, HIGH);
    analogWrite(IN6, 255 - speed);
    // Motor 4
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN8, HIGH);
    analogWrite(IN7, 255 - speed);
}

void motor_backward(int speed)
{
    // Motor 1
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN1, LOW);
    analogWrite(IN2, speed);
    // Motor 2
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN4, LOW);
    analogWrite(IN3, speed);
    // Motor 3
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN5, LOW);
    analogWrite(IN6, speed);
    // Motor 4
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN8, LOW);
    analogWrite(IN7, speed);
}

void motor_turnRight(int speed)
{
    // Motor 1
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
```

```

// Motor 4
digitalWrite(IN7, LOW);
digitalWrite(IN8, LOW);
// Motor 2
speed = constrain(speed, MIN_SPEED, MAX_SPEED);
digitalWrite(IN4, HIGH);
analogWrite(IN3, 255 - speed);
// Motor 3
speed = constrain(speed, MIN_SPEED, MAX_SPEED);
digitalWrite(IN5, HIGH);
analogWrite(IN6, 255 - speed);

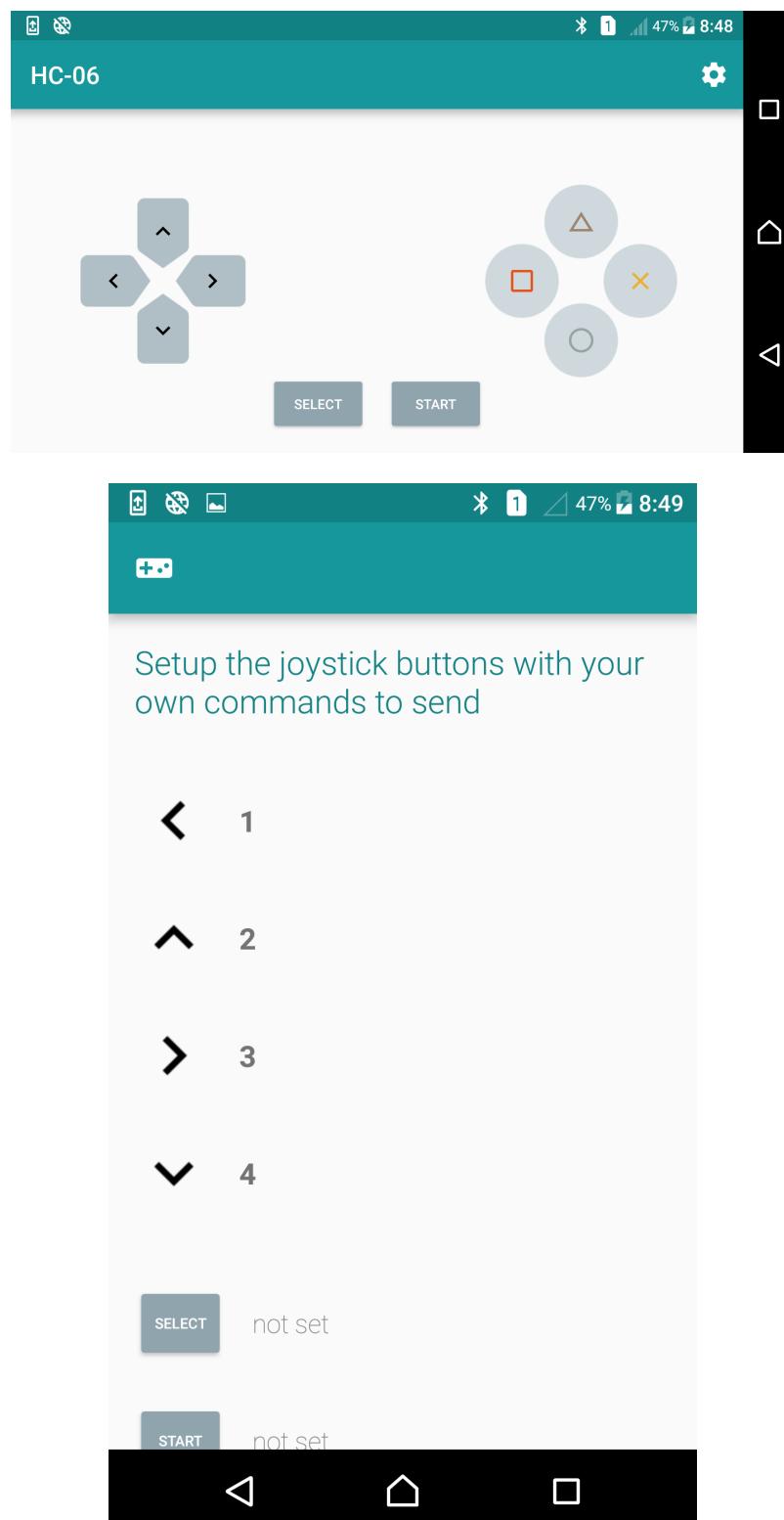
}

void motor_turnLeft(int speed)
{
    // Motor 2
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    // Motor 3
    digitalWrite(IN5, LOW);
    digitalWrite(IN6, LOW);
    // Motor 1
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN1, HIGH);
    analogWrite(IN2, 255 - speed);
    // Motor 4
    speed = constrain(speed, MIN_SPEED, MAX_SPEED);
    digitalWrite(IN8, HIGH);
    analogWrite(IN7, 255 - speed);
}
void loop()
{
    state = '0';
    if (mySerial.available() > 0) { // Nếu có dữ liệu nhận được từ mySerial
        state = mySerial.read(); // Đọc dữ liệu nhận được và lưu vào biến state
        Serial.println(state); // In ra màn hình serial dữ liệu nhận được
    }
    if (state == '1') { // Nếu ký tự nhận được là 1, xe rẽ trái
        motor_turnLeft(MAX_SPEED);
        Serial.println("turn left");
    } else if (state == '2') {
        motor_forward(MAX_SPEED);
        Serial.println("forward");
    } else if (state == '3') {
        motor_turnLeft(MAX_SPEED);
        Serial.println("turn right");
    } else if (state == '4') {
        motor_backward(MAX_SPEED);
        Serial.println("backward");
    } else if (state == '5') {
        motor_stop();
        Serial.println("stop");
    }
}

```

Sau khi nạp chương trình cho Arduino, người dùng sẽ sử dụng ứng dụng Arduino bluetooth controller với các phím điều khiển tương ứng để điều khiển xe. Khi mở app kết nối được với HC-06 rồi thì bạn phải đặt giá trị gửi cho các nút điều khiển, ví dụ như nút "^" là nút xe đi thẳng mà trong code nạp cho Arduino bạn xét xe đi thẳng khi nhận được giá trị là "2" thì bạn sẽ phải đặt nút "^" trong app là "2".

Hình minh họa



Giám sát nhiệt độ, độ ẩm và bật tắt thiết bị thông qua WiFi

Yêu cầu

- Đọc giá trị nhiệt độ, độ ẩm thông qua [Cảm biến nhiệt độ, độ ẩm DHT11](#).
- Tạo một điểm truy cập WiFi (Access Point) để các thiết bị có thể kết nối.
- Các thiết bị khi kết nối WiFi này có thể giám sát được nhiệt độ, độ ẩm đọc được từ cảm biến và điều khiển các thiết bị.

Linh kiện cần dùng

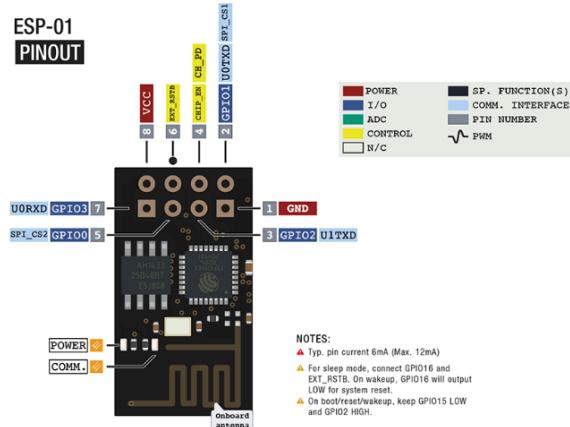
- [Board IoT Maker UnoX](#).
- [Module ESP-01](#).
- [Cảm biến nhiệt độ, độ ẩm DHT11](#).

Giới thiệu về ESP8266

ESP8266 là một dạng vi điều khiển tích hợp WiFi 2.4GHz (WiFi SoC) được phát triển bởi [Expressif](#). Với giá thành rất rẻ nhưng có khả năng hoạt động như một Modem WiFi:

- Có thể quét (scan) và kết nối đến một mạng WiFi bất kỳ (Chế độ WiFi Station) để thực hiện các tác vụ như lưu trữ, truy cập dữ liệu từ server.
- Tạo điểm truy cập WiFi (Chế độ WiFi Access Point) cho phép các thiết bị khác kết nối, giao tiếp và điều khiển.
- Là một server để xử lý dữ liệu từ các thiết bị sử dụng Internet khác.

Hiện nay, trên thị trường có rất nhiều [phiên bản của ESP8266](#) nhưng trong ví dụ này, chúng ta chỉ cần sử dụng module [ESP-01](#) với những ưu điểm như: thiết kế cực kì nhỏ gọn, dễ dàng kết nối và điều khiển, giá thành rẻ...



Hình 126. Hình ảnh pinout của module ESP01 (Nguồn www.acrobotic.com)

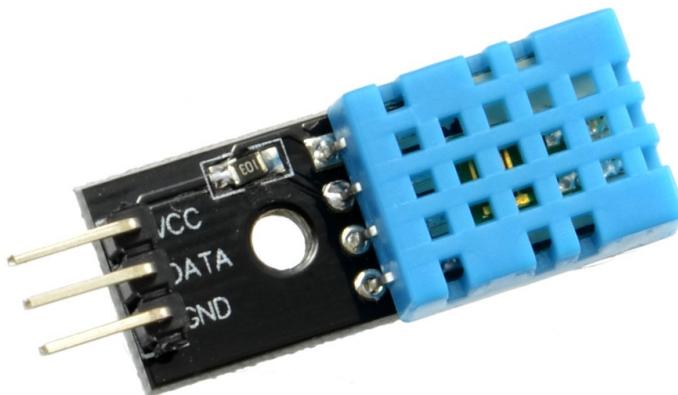
Để board IoT Maker UnoX giao tiếp với ESP-01 ta sử dụng [tập lệnh AT](#).



Trang web học ESP8266 với Arduino phiên bản tiếng Việt: arduino.esp8266.vn, trang web học ESP8266 với Arduino phiên bản tiếng Anh: arduino-esp8266.readthedocs.io/en/latest/.

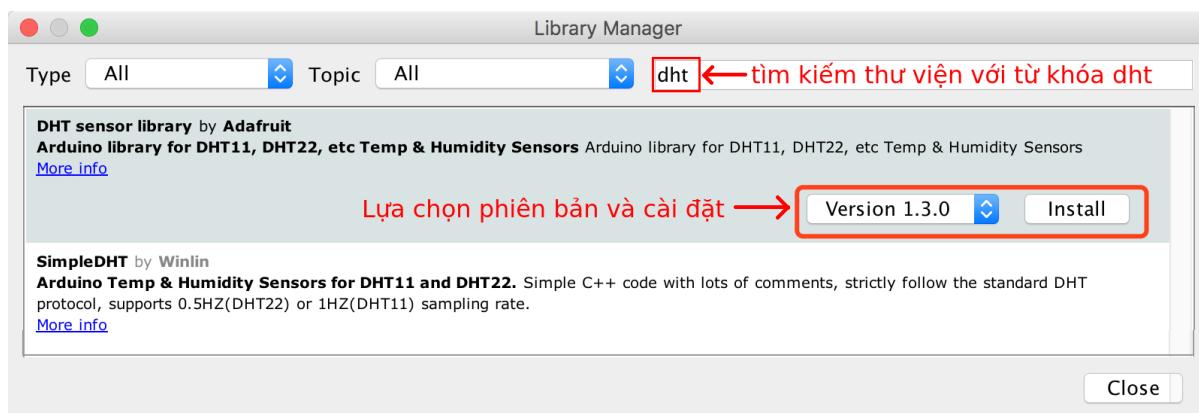
Cảm biến DHT11

[DHT11](#) là một cảm biến có khả năng đo nhiệt độ và độ ẩm không khí với độ chính xác vừa phải, giá cả phải chăng. Có thể lấy dữ liệu đo được của cảm biến bằng giao thức 1-Wire.



Hình 127. Hình ảnh cảm biến DHT11

Thư viện hỗ trợ lấy dữ liệu của DHT11. Dựa theo chuẩn truyền nhận 1-Wire và sự phổ biến của dòng sensor DHTXX (DHT11, DHT22,...), có rất nhiều thư viện được xây dựng lên để việc lập trình với DHT11 trở nên dễ dàng hơn. Trong bài này chúng ta sẽ cài đặt và sử dụng thư viện [DHT sensor library](#) của Adafruit.



Hình 128. Hình ảnh tìm kiếm và cài đặt thư viện DHT11

Ngoài ra, để sử dụng được thư viện DHT sensor library chúng ta cần thêm thư viện [Adafruit Unified Sensor Driver](#). Sau khi tải thư viện về, bạn mở cửa sổ Arduino, chọn Sketch → Import Library... → Add Library.... Sau đó chọn file .zip mà bạn vừa tải về để có thể sử dụng thư viện.

Phân tích

- Chúng ta sẽ kết nối ESP-01 với Board Arduino bằng giao tiếp UART trên chân 10 và 11 (Sử dụng thư viện [SoftwareSerial.h](#)).
- ESP-01 hoạt động ở chế độ Access Point sẽ tạo một điểm phát sóng WiFi để các thiết bị khác truy cập vào.
- Thiết bị khi kết nối WiFi của ESP-01 có thể điều khiển và giám sát thiết bị thông qua trình duyệt WEB (Google Chrome, IE, Cốc Cốc, EAGLE) với đường dẫn ([link](#)) và các chức năng liệt kê như bên dưới:
 - IPAdress:** Sẽ hiển thị nhiệt độ, độ ẩm.
 - IPAdress/LED=ON:** Sẽ bật LED trên board.
 - IPAdress/LED=OFF:** Sẽ tắt LED trên board.

Với **IPAdress** là địa chỉ IP của ESP-01. Thông số này sẽ hiển thị trên SerialMonitor (Thông thường là 192.168.4.1).

Sơ đồ kết nối

Bảng 25. Bảng đấu nối của IoT Maker UnoX và ESP-01

IoT Maker UnoX	ESP-01
3V3	8 (Vcc)
GND	1 (GND)
3	4 (RXD)
2	5 (TXD)
3V3	6 (CH_PD)

Bảng 26. Bảng đấu nối của IoT Maker UnoX và DHT11

IoT Maker UnoX	DHT11
5V	Vcc
GND	GND
9	Data

[dht11 esp01 unox] | 08-project/project-wifi/dht11-esp01-unox.png

Hình 129. Hình ảnh kết nối cảm biến DHT11 và ESP-01 với board IoT Maker UnoX

Source code

```
#include <SoftwareSerial.h>
#include <DHT.h>

#define DEBUG false
const int DHTPIN = 9;           // Chân đọc dữ liệu từ DHT
const int DHTTYPE = DHT11;      // Chọn kiểu cảm biến (DHT11 hoặc DHT22)
DHT dht(DHTPIN, DHTTYPE);

SoftwareSerial Serial1(2, 3); // 10-RX, 11-TX

// Đoạn Code HTML sẽ hiển thị lên trình duyệt.
String html =
  "<html>
  <head>
  <title>ESP8266 Webserver</title>
  </head>
  <body>
  <h1>ARDUINO STATER</h1>
  <a>LED </a>
  <a href=\"/LED=ON\">ON </a>
  <a href=\"/LED=OFF\">OFF </a>
  </body>
</html>";

void setup()
{
  Serial.begin(9600);          // Khởi động Serial Monitor
  Serial1.begin(115200);        // Khởi động UART kết nối ESP-01
  pinMode(13, OUTPUT);         // Khai báo LED 13
  digitalWrite(13, LOW);        // Tắt LED 13

  sendData("AT+RST\r\n", 2000, DEBUG);           // Reset module
  sendData("AT+CWMODE=2\r\n", 1000, DEBUG);        // Khai báo ESP-01 là Access Point
  sendData("AT+CWSAP=\"CONTROL\", \"12345678\", 5, 3\r\n", 5000, DEBUG); // Tạo điểm phát Wifi với ID và Pass.
  sendData("AT+CIFSR\r\n", 1000, true);            // Hiện IP của ESP-01
  sendData("AT+CIPMUX=1\r\n", 1000, DEBUG);         // Configure for multiple connections
  sendData("AT+CIPSERVER=1,80\r\n", 1000, DEBUG);   // Turn on server on port 80
}

void loop()
{
  String IncomingString = "";
  boolean StringReady = false;
  while (Serial1.available()) {
    IncomingString = Serial1.readString();
    StringReady = true;
    if (StringReady) {
      Serial.println("Received String: " + IncomingString);
      if (IncomingString.indexOf("LED=ON") != -1) {
        digitalWrite(13, HIGH);
      }
    }
  }
}
```

```

if (IncomingString.indexOf("LED=OFF") != -1) {
    digitalWrite(13, LOW);
}
else {
    String webpage = "<h1>Hello World</h1><a href=\"\">link text</a>";
    espSend(html);
    float temp = dht.readTemperature(); // Lấy giá trị nhiệt độ từ cảm biến
    float humi = dht.readHumidity(); // Lấy giá trị độ ẩm từ cảm biến
    if (isnan(temp) || isnan(humi)) { // Kiểm tra dữ liệu đọc được có phải là số không
        String c = "sensor is not connected"; // Nếu không phải là số thì báo không kết nối
        espSend(c);
    }

    else { // Nếu là số thì hiển thị nhiệt độ và độ ẩm
        String add1 = "<h4>Temperature=</h4>";
        add1 += String(temp);
        add1 += "&#x2103"; // Mã HEX của độ C
        add1 += "<h4>Humidity=</h4>";
        add1 += String(humi);
        add1 += "%";
        espSend(add1);
    }
}

String closeCommand = "AT+CIPCLOSE=0\r\n"; // Lệnh đóng kết nối với thiết bị sau khi gửi song dữ liệu
sendData(closeCommand, 3000, DEBUG);
}
}
}

// Hàm gửi dữ liệu cho thiết bị truy cập
void espSend(String d)
{
    String cipSend = " AT+CIPSEND=0,";
    cipSend += d.length();
    cipSend += "\r\n";
    sendData(cipSend, 1000, DEBUG);
    sendData(d, 1000, DEBUG);
}

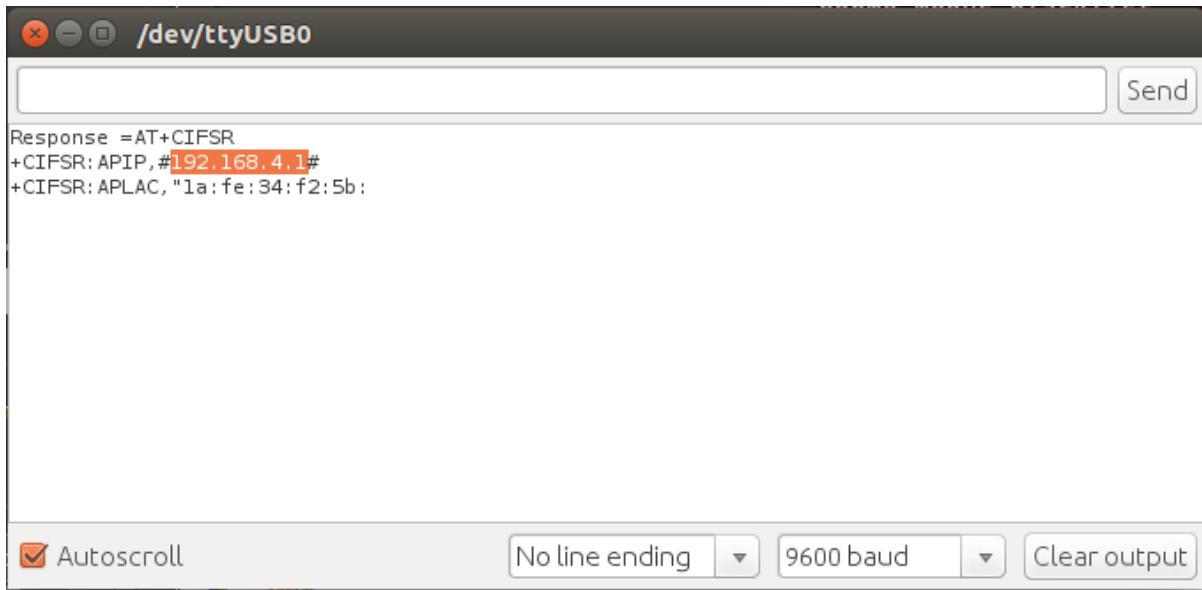
// Hàm gửi lệnh và nhận dữ liệu phản hồi từ ESP
String sendData(String command, const int timeout, boolean debug)
{
    String response = "";
    Serial1.print(command);
    long int time = millis();
    while ( (time + timeout) > millis()) {
        while (Serial1.available()) {
            char c = Serial1.read(); // Đọc ký tự tiếp theo.
            response += c;
        }
    }

    if (debug) { // Nếu chọn chế độ DEBUG thì sẽ hiện lên Serial Monitor
        Serial.print("Response =" + response);
    }
    return response;
}

```

Nội dung của biến `html` là 1 đoạn code html được chuyển qua kiểu string. Chúng ta có thể tìm hiểu về HTML tại: w3schools.com.

Kết nối board với máy tính bằng cổng COM, nạp chương trình. Màn hình SerialMonitor sẽ hiển thị `IPAddress` của bạn như sau:

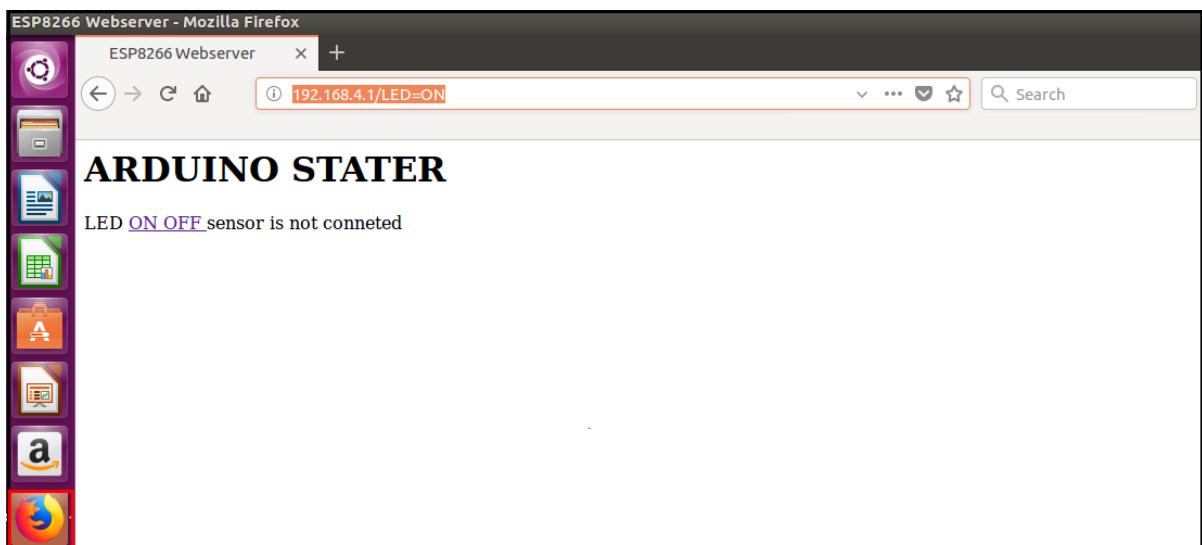


The screenshot shows a terminal window titled '/dev/ttyUSB0'. It displays the following AT command responses:

```
Response =AT+CIFSR
+CIFSR: APIP,#192.168.4.1#
+CIFSR: APLAC,"1a:fe:34:f2:5b:"
```

At the bottom of the window, there are several configuration buttons: 'Autoscroll' (checked), 'No line ending' (dropdown), '9600 baud' (dropdown), and 'Clear output'.

Chúng ta sẽ sử dụng IPAddress này truy cập server để đọc nhiệt độ hoặc điều khiển thiết bị. Ví dụ điều khiển bật/tắt LED như hình dưới.



Tổng kết

Sau khi hoàn thành dự án này, chúng ta đã hiểu được tổng quan được cách kết nối, giao tiếp Arduino với ESP8266 đồng thời hiểu được cơ bản một Web Server hoạt động như thế nào. Đây sẽ là nền tảng để chúng ta phát triển những dự án IoT sau này.

Cheatsheet

Phần này cung cấp thông tin cho việc tra cứu nhanh các hàm có thể sử dụng với Arduino và ngôn ngữ lập trình C. Với Arduino sẽ có rất nhiều thư viện cung cấp các tính năng hữu ích thông qua các API cho phép người sử dụng gọi, nội dung tóm lược ở đây chỉ đề cập tới các thư viện thường xuyên được sử dụng nhất. Nếu bạn sử dụng các thư viện khác, hoàn toàn có thể tra cứu dễ dàng trong tài liệu sử dụng của thư viện đó. Thông thường các thư viện được phát hành nguồn mở trên [Github](#) sẽ có file **README.md** cung cấp đầy đủ các thông tin.

Arduino Cheatsheet

```

/* CẤU TRÚC CƠ BẢN CỦA 1 SKETCH */
void setup() {
/*
Hàm được gọi khi bắt đầu sketch. Dùng để khởi tạo
biến, cấu hình chân, gọi thư viện, ...
Code trong setup chỉ chạy 1 lần (khi khởi động hoặc
reset board)
*/
}
void loop() {
// Nội dung trong loop() lặp lại liên tục
}

/* LỆNH RẼ NHÁNH */

/* Lệnh if else */
if (x < 5)      // thực thi code nếu x<5
{ code }
else if (x > 10)// thực thi code nếu x>10
{ code }
else { code }   // thực thi code các trường hợp còn lại

/* Lệnh switch case */
switch (var) { // thực thi case có giá trị var
case 1:
{source code}
break;
case 2:
{source code}
break;
default:
{source code}
}

/* CÁC KIỂU VÒNG LẶP */

/* Vòng lặp while */
while (x < 5) { code };
/* Thực hiện code nếu x<5 */

/* Vòng lặp do ... while */
do { code } while(x < 0);
/* Thực hiện code, so sánh, nếu x<0 tiếp tục thực hiện
code */

for (int i = 0; i < 10; i++) { code };
/* Khởi tạo i, thực hiện code và tăng i nếu i < 10 */
break; /* Thoát ra vòng lặp (for, while, do-while) ngay
lập tức */
continue; /* Đi đến chu kỳ lặp tiếp theo của vòng lặp
hiện tại */

```

```

/* CÁC ĐỊNH NGHĨA VỀ HÀM*/
<ret. type> <name>(<params>) { ... }
/* ret.type : Kiểu trả về của hàm
name : Tên của hàm
params : Các đối số truyền vào hàm
{ ... } : source code được viết trong {}
Ví dụ : int func_name(int x) {return x*2;}
*/

return x; // x phải trùng khớp với kiểu trả về của hàm
return; // loại return dành cho hàm void
/* INCLUDE */
#include <stdio.h>
/* include thư viện chuẩn */
#include "your-library.h"
/* include thư viện tạo bởi người dùng */

/* DỮ LIỆU KIỂU CHUỖI */

char str1[8] = {'A','r','d','u','i','n','o','\0'};
/* Chuỗi bao gồm kí tự kết thúc chuỗi \0 (null) */

char str2[8] = {'A','r','d','u','i','n','o','\0'};
/* Trình biên dịch tự động thêm kí tự \0 vào cuối chuỗi */

char str3[] = "Arduino";
char str4[8] = "Arduino";
/* Khai báo và gán giá trị cho chuỗi */

/* DỮ LIỆU KIỂU MẢNG */

int myPins[] = {2, 4, 8, 3, 6}; /* Khai báo mảng kiểu
int có 6 phần tử và gán giá trị cho mỗi phần tử */

int myInts[6];
/* Mảng kiểu int 6 phần tử và không gán giá trị */

myInts[0] = 42; // Gán giá trị 42 cho phần tử đầu tiên
myInts[6] = 12; // LỐI ! chỉ số của mảng chỉ từ 0 đến 5

/*Qualifiers*/
static // Không thay đổi giá trị ở các lần gọi
volatile // In RAM (Thường dùng trong ngắn)
const // Không đổi (chỉ đọc)
PROGMEM /* Cho phép lưu trữ dữ liệu trong bộ nhớ
FLASH thay vì SRAM */

/* CÁC TOÁN TỬ, PHÉP TOÁN THƯỜNG DÙNG */
/* Các toán tử thường dùng */

```

```

= toán tử bằng
+ toán tử cộng
- toán tử trừ
* toán tử nhân
/ toán tử chia lấy phần nguyên
% toán tử chia lấy phần dư
== phép so sánh bằng
!= phép so sánh không bằng (khác)
< phép so sánh nhỏ hơn
> phép so sánh lớn hơn
<= phép so sánh nhỏ hơn hoặc bằng
>= phép so sánh lớn hơn hoặc bằng
&& phép toán logic (AND)
|| phép toán logic (OR)
! phép toán logic (NOT)
/* Các toán tử hợp nhất */
++ tăng 1 đơn vị
-- giảm 1 đơn vị
+= phép toán cộng và gán giá trị
  ex: x = 5; x+= 1; //x = 6
-= phép toán trừ và gán giá trị
  ex: x = 5; x-= 1; //x = 4
*= phép toán nhân và gán giá trị
  ex: x = 5; x*= 3; //x = 15
/= phép toán chia lấy phần nguyên và gán giá trị
  ex: x = 6; x/= 2; //x = 3
&= phép toán logic AND và gán giá trị
  ex: x = 0b1010; x&= 0110; //x = 0b0010
|= phép toán logic OR và gán giá trị
  ex: x = 0b1010; x|= 0110; //x = 0b1110

/* Các toán tử trên bit */
& and          ^ xor
<< dịch trái    >> dịch phải
| or           ~ not

/* Thực thi với con trỏ */
&reference: // lấy địa chỉ của biến mà con trỏ trỏ tới
*dereference:// lấy giá trị của biến mà con trỏ trỏ tới

/* HÀNG SỐ VÀ KIỂU DỮ LIỆU */
123 Số thập phân
0b0111 Số nhị phân
0173 Số Octal - base 8
0x7B Số thập lục phân base 16
123U Số nguyên không dấu
123L Số nguyên có dấu 4 bytes
123UL Số nguyên không dấu 4bytes
123.0 Số thực
1.23e6 Số thực dùng cơ số mũ ex: 1.23*10^3 = 1230

/* PHẠM VI CỦA KIỂU DỮ LIỆU */
boolean true | false
char -128 - 127, 'a' '$' etc.
unsigned char 0 - 255
byte 0 - 255
int -32768 - 32767
unsigned int 0 - 65535
word 0 - 65535
long -2147483648 - 2147483647

```

```

unsigned long 0 - 4294967295
float -3.4028e+38 - 3.4028e+38
double -3.4028e+38 - 3.4028e+38
void i.e., no return value

/* KHAI BÁO BIẾN */
int a;
int a = 0b0111011, b = 0123, c = 1, d;
float fa = 1.0f;
double da = 1.0;
char *pointer;
char *other_pointer = NULL;

/**
 * BUILT-IN FUNCTIONS
 * Pin Input/Output
 * Digital I/O - pins 0-13 A0-A5
 */
pinMode(pin,[INPUT, OUTPUT, INPUT_PULLUP])
/* Thiết lập cấu hình chân */
int a = digitalRead(pin_6);
/* Đọc giá trị của pin_6 và gán cho a */
digitalWrite(pin_5, [HIGH, LOW])
/* Cài đặt mức HIGH/LOW cho pin_5 */
int a = analogRead(pin)
/* Đọc giá trị của pin và gán cho a, pin từ A0-->A5 */

analogWrite(pin, value)
/* PWM ngõ ra - pins 3 5 6 9 10 11
 * ESP8266: pin 0..16, range = 0..1023, 1KHz default
 */
/* Đặt giá trị PWM cho pin */
analogWriteRange(new_range)
/* ESP8266: thay đổi RANGE PWM output */
analogWriteFreq(new_frequency)
/* ESP8266: Tần số PWM output */

/* ADVANCED I/O */
tone(pin, freq_Hz)
/* Tạo sóng vuông tần số freq_Hz với duty cycle=50% */
tone(pin, freq_Hz, duration_ms)
/* Tạo sóng vuông tần số freq_Hz, duration mili giây */
noTone(pin)
/* Ngừng việc tạo sóng vuông khi dùng tone() */

shiftOut(dataPin, clockPin,[MSBFIRST, LSBFIRST], value)
/* Dịch 1 byte, mỗi lần dịch 1 bit, dịch từ bit cao */

unsigned long pulseIn(pin,[HIGH, LOW])
/* Trả về (ms) của xung HIGH/LOW trên chân pin */

/* CHỨC NĂNG NGẮT */
attachInterrupt(interrupt, func, mode)
/* Thiết lập chức năng ngắt ở các chân digital */
/*
interrupt: số ngắt (thường là chân sử dụng chức năng ngắt)
func : hàm sẽ được gọi khi ngắt xảy ra (lưu ý : hàm không có tham số đầu vào cũng như kiểu trả về)
mode : gồm các chế độ LOW, CHANGE, RISING, FALLING. Ngắt

```

```

sẽ được kích hoạt khi chân ngắt ở mode tương ứng
*/
detachInterrupt(interrupt)
/* Vô hiệu hóa ngắt interrupt */
noInterrupts()
/* Vô hiệu hóa tất cả các ngắt */
interrupts()
/* Cho phép tái ngắt sau khi dùng noInterrupts() */

/*****************
 *          THƯ VIỆN PHỐ BIỂN
 *****************/
/*****************
 *           Serial
 *Thư viện giao tiếp với PC hoặc thông qua RX/TX
*****************/
begin(long speed)
/* Thiết lập giao tiếp serial-UART với tốc độ speed */

end()
/* Vô hiệu hóa giao tiếp serial */

int available()
/* Trả về số bytes có sẵn để đọc */

int read()
/* đọc dữ liệu đến từ serial (trả về byte đầu tiên của
dữ liệu từ serial hoặc -1 nếu dữ liệu không có */

flush()
/* Chờ quá trình truyền dữ liệu serial hoàn tất */

print(data)
/* In ra serial port dữ liệu data (với bất kỳ kiểu dữ
liệu nào được thiết lập */

println(data)
/* Tương tự như print(data) nhưng sau khi in ra serial
-port, con trỏ sẽ xuống dòng tiếp theo */

write(byte)
/* Gửi dữ liệu value/string/array đến serial port */

SerialEvent()
/* Hàm được gọi khi có dữ liệu đến từ chân RX */

/*****************
 *          Servo.h
 *Thư viện hỗ trợ điều khiển động cơ servo
*****************/
attach(pin, [min_uS, max_uS])
/*
Thiết lập chân kết nối với servo và độ rộng xung
pin : Chân kết nối với servo
[min_uS, max_uS] : Độ rộng xung tính theo us tương ứng
với góc xoay từ 0 đến 180
*/

write(angle)
/* Ghi dữ liệu góc xoay cho động cơ angle từ 0~180 */

writeMicroseconds(uS)

```

```

/* Viết giá trị để điều khiển góc quay cho servo, giá
trị từ 700 ~ 2300 */

int read()
/* Đọc giá trị góc xoay (0 đến 180 độ) */

bool attached()
/* Trả về true nếu biến servo đã kết nối đến pin */

detach()
/* Gỡ bỏ biến servo ra khỏi chân đã kết nối */

/*****************
 *          Wire.h
 *Dùng trong giao tiếp I2C
*****************/
begin()
/* Master khởi tạo thư viện Wire với giao tiếp I2C */

begin(addr)
/* Slave tham gia vào kết nối i2C, addr là 7 bits địa
chỉ của slave */

requestFrom(address, count, stop)
/*
Master yêu cầu 1 số byte từ slave:
address: 7bits địa chỉ của slave.
count: Số lượng byte master yêu cầu
stop: Kiểu boolean, nếu true, master tín hiệu stop sau
khi yêu cầu và giải phóng bus I2C, nếu false, master
gửi yêu cầu restart để giữ kết nối
*/

beginTransmission(addr)
/* Gửi tín hiệu bắt đầu, truyền dữ liệu đến slave có
địa chỉ addr */

send(byte)
/* Gửi dữ liệu (1 byte) đến slave */

send(char * string)
/* Gửi dữ liệu (string) đến slave */

send(byte * data, size)
/* Gửi dữ liệu (1 mảng) với số byte là size */

endTransmission()
/* Gửi tín hiệu kết thúc truyền dữ liệu tới slave */

int available()
/* Trả về số byte available sau khi đọc bởi read() */

byte receive()
/* truy xuất đến 1 byte đã truyền từ slave đến master
hoặc truyền ở chiều ngược lại khi nhận được
requestFrom. Trả về byte tiếp theo đã nhận được */

onReceive(handler)
/* Hàm handler sẽ được gọi khi slave nhận dữ liệu */

onRequest(handler)
/* Handler sẽ được gọi khi master yêu cầu dữ liệu */

```

C - CheatSheet

```

/* CẤU TRÚC CƠ BẢN */
Viết chú thích trên 1 dòng dùng //
    ex: x++ ; // tăng x 1 đơn vị
/* */ Viết chú thích trên nhiều dòng.
ex : ****
    * Chú thích được viết *
    * trên nhiều dòng *
*****
/* CẤU TRÚC 1 CHƯƠNG TRÌNH */
#include <stdio.h> //include thư viện chuẩn của C
#include "iLib.h" // include thư viện tạo bối cảnh người dùng
int global_var; // biến được dùng trong chương trình
/* Khai báo hàm bắt đầu của 1 chương trình C với kiểu trả về là integer. Đổi số arg kiểu int được truyền vào hàm */
int main (int argc){
    float local_var ; // Biến chỉ được dùng trong hàm main
Lệnh 1
...
Lệnh n ;
return 0; //chương trình thực hiện thành công và thoát
}

/* KIỂU DỮ LIỆU VÀ PHẠM VI */
boolean      true | false
char         -128          - 127, 'a' '$' etc.
unsigned char 0            - 255
byte          0             - 255
int           -32768        - 32767
unsigned int  0             - 65535
word          0             - 65535
long          -2147483648   - 2147483647
unsigned long 0             - 4294967295
float         -3.4028e+38   - 3.4028e+38
double        -3.4028e+38   - 3.4028e+38
void          i.e., no return value

/* ĐẶT TÊN BIẾN */
/* Đặt tên đúng */
int x;          // Một biến
int x = 1;       // Biến được khai báo và khởi tạo
float x, y, z;  // Nhiều biến cùng kiểu dữ liệu
const int x = 88; // Biến tĩnh, không ghi được
int tenBien1ok; // Đặt tên biến này đúng
int ten_bien_nay_ok;
/* Đặt tên sai */
int 2001_tensai; // Vì số ở đầu
int ten-sai;     // Dấu '-' không phải là alphanumeric
int while;       // Sai, vì dùng từ khóa vòng lặp while

/* HẰNG SỐ VÀ KIỂU DỮ LIỆU */
123      Số thập phân
0b0111  Số nhị phân
0173    Số Octal - base 8
0x7B    Số thập lục phân base 16
123U    Số nguyên không dấu

```

```

123L      Số nguyên có dấu 4 bytes
123UL     Số nguyên không dấu 4bytes
123.0    Số thực
1.23e6   Số thực dùng cơ số mũ ex: 1.23*10^3 = 1230
/* Định nghĩa hằng số a kiểu nguyên, có giá trị là 1 */
const int a = 1;
/* Định nghĩa hằng số x kiểu thực, có giá trị là 4.0 */
const float x = 4;
/* Định nghĩa hằng số c kiểu integer có giá trị 49 */
const c = '1'; // Kí tự 1 trong mã ASCII là 49
/* Định nghĩa str là hằng số kiểu con trỏ, trỏ tới chuỗi "Cheasheet C" */
const char * str = "Cheasheet C";

/* KHAI BÁO BIẾN */
/* Khai báo biến a kiểu nguyên và không gán giá trị */
int a;
/* khai báo a kiểu binary, b kiểu base8, c kiểu số nguyên, d kiểu số nguyên và không gán giá trị */
int a = 0b0111011, b = 0123, c = 1, d;
/* Khai báo biến fa thuộc kiểu số thực float */
float fa = 1.0f;
/* Khai báo biến da thuộc kiểu số thực double */
double da = 1.0;
/* Khai báo biến con trỏ và trỏ đến 1 vùng nhớ không xác định */
char *pointer;
/* Khai báo biến con trỏ và trỏ về NULL (0) */
char *other_pointer = NULL;

/* CHUỖI KÍ TỰ */
/* Chuỗi bao gồm kí tự kết thúc chuỗi \0 (null) */
char str1[8] = {'A', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
/* Trình biên dịch tự động thêm kí tự \0 vào cuối chuỗi */
char str2[8] = {'A', 'r', 'd', 'u', 'i', 'n', 'o'};
/* Khai báo chuỗi ,không khai báo số phần tử và gán giá trị chuỗi */
char str3[] = "Arduino";
/* Khai báo và gán giá trị cho chuỗi */
char str4[8] = "Arduino";

/* Các hàm xử lý chuỗi thường dùng */
/* Nối các kí tự từ chuỗi source tiếp vào vị trí cuối của chuỗi dest */
strcat(dest, source)
/* Tìm vị trí xuất hiện đầu tiên của kí tự c trong source, trả về con trỏ chỉ tới vị trí đó hoặc null nếu không tìm thấy c trong source */
 strchr(source, c)
/* Hàm trả về độ dài của chuỗi st */
strlen(st)
/* copy và thay các kí tự của chuỗi source vào dest */
strcpy(dest, source)
/* chép kí tự từ đầu đến n từ chuỗi source vào dest */
strncpy(dest, source, n)

```

```

/* MẢNG */
/* Khai báo mảng 1 chiều 6 phần tử kiểu integer và gán
   giá trị cho mỗi phần tử */
int myPins[] = {2, 4, 8, 3, 6};
/* Khai báo mảng 1 chiều 6 phần tử kiểu integer và
   không gán giá trị */
int myInts[6];
myInts[0] = 42; // Gán giá trị 42 cho phần tử đầu tiên
myInts[6] = 12; // LỖI ! chỉ số của mảng chỉ từ 0 đến 5
/* Lấy giá trị của phần tử thứ 3 trong mảng myInts */
int c = myInts[2]; // Có thể dùng *(myInts + 2)
/* Lấy địa chỉ của phần tử thứ 3 trong mảng myInts */
int c = &myInts[2]; // Có thể dùng int c = myInts + int
/* Trả về chiều dài của mảng myInts */
int length = sizeof(myInts) / sizeof(myInts[0]);
/* Khai báo 2 mảng kiểu float, arr1 có 5 phần tử, arr2
   có 10 phần tử */
float arr1[5], arr2[10];
/* Khai báo mảng số nguyên arr có 2 dòng, 5 cột. Tổng
   cộng có 10 phần tử */
int a[2][5];

/* KHỐI LỆNH VÀ CÁC LỆNH DÙNG TRONG VÒNG LẶP */
{} // bao gồm nhiều lệnh, thường được sử dụng trong h
àm
/* Goto : chương trình sẽ nhảy đến nhãn (nhãn phải có
   mặt trong câu lệnh chứa goto) */
goto nhãn;
/* Continue : Chỉ dùng trong các lệnh có vòng lặp sẽ
   chuyển qua chu kỳ mới của vòng lặp trong cùng nhất */
continue; /*
/* Break : Dùng với các vòng lặp thoát khỏi vòng lặp
   trong cùng nhất, hoặc dùng trong cấu trúc switch..case
   để thoát ra khỏi case tương ứng */
break; /*
/* Return */
/* Dùng cho hàm không có kiểu trả về (void) */
return;
/* Value có thể là hằng số, biến, biểu thức hoặc gọi
   đến 1 hàm khác để trả về giá trị */
return <value>

/* LỆNH RẼ NHÁNH */
if (x < 5) // thực thi code nếu x<5
{ code }
else if (x > 10)// thực thi code nếu x>10
{ code }
else { code } // thực thi code các trường hợp còn lại

switch (var) { // thực thi case có giá trị var
case 1:
...
break;
case 2:
...
break;
default:
...
}

/* CÁC KIỂU VÒNG LẶP */
/* While: Thực hiện code nếu x<5 */
while (x < 5) { code };
/* Do-While : Thực hiện code, so sánh, nếu x<0 tiếp tục
   thực hiện code */
do { code } while(x < 0);

```

```

/* for : Khởi tạo và gán giá trị cho i, thực hiện code
   tăng i nếu i < 10 */
for (int i = 0; i < 10; i++) { code };

/* PHÉP TOÁN VÀ TOÁN TỬ THƯỜNG DÙNG
   Các toán tử thường dùng */
= toán tử bằng
+ toán tử cộng
- toán tử trừ
* toán tử nhân
/ toán tử chia lấy phần nguyên
% toán tử chia lấy phần dư
== phép so sánh bằng
!= phép so sánh không bằng (khác)
< phép so sánh nhỏ hơn
> phép so sánh lớn hơn
<= phép so sánh nhỏ hơn hoặc bằng
>= phép so sánh lớn hơn hoặc bằng
&& phép toán logic (AND)
|| phép toán logic (OR)
! phép toán logic (NOT)

/* Các toán tử hợp nhất */
++ tăng 1 đơn vị
-- giảm 1 đơn vị
+= phép toán cộng và gán giá trị
   ex: x = 5; x+= 1; //x = 6
-= phép toán trừ và gán giá trị
   ex: x = 5; x-= 1; //x = 4
*= phép toán nhân và gán giá trị
   ex: x = 5; x*= 3; //x = 15
/= phép toán chia lấy phần nguyên và gán giá trị
   ex: x = 6; x/= 2; //x = 3
&= phép toán logic AND và gán giá trị
   ex: x = 0b1010; x&= 0110; //x = 0b0010
|= phép toán logic OR và gán giá trị
   ex: x = 0b1010; x|= 0110; //x = 0b1110

/* Các toán tử trên bit */
& and ^ xor
<< dịch trái >> dịch phải
| or ~ not

/* THỰC THI VỚI CON TRỎ */
&reference: // lấy địa chỉ của biến mà con trỏ trỏ tới
*dereference:// lấy giá trị của biến mà con trỏ trỏ tới
/* khai báo biến con trỏ kiểu int trỏ tới địa chỉ của
   biến a */
int a = 5; int *pointer = &a;

/* CÁC KÍ TỰ ĐIỀU KHIỂN VÀ KÍ TỰ ĐẶC BIỆT */
\n Nhảy xuống dòng kế tiếp cạnh về cột đầu tiên
\t Cảnh cột tab ngang.
\r Nhảy về đầu hàng, không xuống hàng.
\a Tiếng kêu bip.
\\ In ra dấu \
\" In ra dấu "
\' In ra dấu '
%%: In ra dấu %
\b ~ backspace (xóa 1 ký tự ngay trước)

/* HÀM VÀ CÁC VĂN ĐỀ LIÊN QUAN */
/* Khai báo prototype của hàm max, có 2 đối số đầu vào
   là a và b thuộc kiểu số nguyên, kết quả trả về của hàm
   kiểu số nguyên */
int max(int a, int b);

```

```

/* Khai báo biến c là giá trị trả về của hàm max */
int c = max(5,4);
/* Khai báo prototype của hàm không có đối số và không
có kiểu trả về (void) */
void none();

/* TYPEDEF- Định nghĩa kiểu dữ liệu */
/* Định nghĩa kiểu unsigned char là BYTE, khai báo các
biến a, b thuộc kiểu BYTE */
typedef unsigned char BYTE; BYTE a, b;

/* KIỂU LIỆT KÊ - ENUMERATION (enum) */
/* khai báo kiểu dữ liệu enum là các ngày trong tuần */
enum daysOfWeek { sunday, monday, tuesday, wednesday };
/* Tạo biến toDay thuộc daysOfWeek và gán giá trị */
daysOfWeek toDay = wednesday;

/* STRUCT - KIỂU DỮ LIỆU DO NGƯỜI DÙNG ĐỊNH NGHĨA */
/* Khai báo struct sinhVien */
struct sinhVien{
    char tenSinhVien;
    char MSSinhVien;
    int tuoiSinhVien;
};

/* Truy xuất đến thành phần MSSinhVien trong struct
sinhVien */
sinhVien.MSSinhVien;
/* Đổi tên struct sinhVien thành 1 biến duy nhất là

```

```

SINHVIEN */
typedef struct sinhVien SINHVIEN;
/* Khai báo biến sinhVienA thuộc struct SINHVIEN */
SINHVIEN sinhVienA;

/* CÁC LỆNH XỬ LÝ TẬP TIN (#include <stdio.h>) */
/* Khai báo 1 biến con trỏ là đường dẫn của 1 file */
const char *filePath = "Đường/dẫn/file/document.txt";
/* Tạo 1 biến con trỏ thuộc kiểu FILE */
FILE *file;
/* Mở 1 file ở đường dẫn filePath và đọc dữ liệu */
file = fopen(filePath, "r");// Trả về NULL nếu thất bại
/* Đóng 1 file đã mở, trả về 0 nếu thành công , ngược
lại trả về EOF */
fclose(file);
/* Viết kí tự c lên file đang mở, trả về EOF nếu ghi
thất bại, trả về mã ASCII của c nếu thành công */
int fputc(int c, FILE *f);
/* Viết chuỗi "hello" lên file đang mở */
int c = fputc("hello", file);
/* Đọc (255-1) kí tự từ file đang mở, kết quả đọc
được
sẽ lưu vào mảng str, việc đọc bị dừng nếu gặp kí tự
'\n' hoặc EOL */
fgets(str, 255, file);
/* Thay đổi vị trí trỏ đến trong file của con trỏ
internal file position indicator về lại đầu file */
int fseek(file, 0, SEEK_SET);
/* Trả về kích thước của nội dung có trong file */
ftell(file);

```

www.cheatography.com/ashlyn-black/cheat-sheets/c-reference/

Lời kết

Các thành viên tham gia đóng góp

Để hoàn thiện nội dung của sách có sự đóng góp của các thành viên sau:

- 1. [Phạm Minh Tuấn \(TuanPM\)](#) - Chủ biên
- 2. [Trịnh Hoàng Đức](#) - Kỹ sư thiết kế phần cứng tại IoT Maker Việt Nam.
- 3. [Trần Phúc Vinh](#) - Thực tập sinh tại IoT Maker Việt Nam - Sinh viên Đại Học Bách Khoa, chuyên ngành kỹ thuật điện, khóa học 2014.
- 4. [Phạm Thị Thu Hiền](#) - Designer tại IoT Maker Việt Nam.
- 5. [Võ Trí Dũng](#) - Thực tập sinh tại IoT Maker Việt Nam, sinh viên Đại Học Sài Gòn, chuyên ngành điện tử máy tính, khóa học 2014.
- 6. [Nguyễn Minh Phương](#) - Thực tập sinh tại IoT Maker Việt Nam, sinh viên Đại Học Sài Gòn, chuyên ngành điện tử máy tính , khóa học 2014.
- 7. [Đặng Quang Trung](#) - Thực tập sinh tại IoT Maker Việt Nam, sinh viên Đại Học Sài Gòn, chuyên ngành điện tử máy tính, khóa học 2014.
- 8. [Lâm Nhật Quân](#) Kỹ sư làm việc tại IoT Maker Việt Nam.
- 9. [Trần Thành Lộc](#) - Thực tập sinh tại IoT Maker Việt Nam - Sinh viên Đại Học Bách Khoa, chuyên ngành kỹ thuật máy tính, khóa học 2016.

Lời kết

Thật vui khi bạn đã đồng hành cùng chúng tôi đi đến hết cuốn sách này. Mục đích của cuốn sách là giúp những người mới bắt đầu tìm hiểu về [Arduino](#) có kiến thức cơ bản và hướng đi chính xác để nghiên cứu về lập trình vi điều khiển một cách nhanh chóng hơn. Hy vọng cuốn sách sẽ đến tay thật nhiều bạn đam mê lĩnh vực điện tử lập trình tuy không còn mới mẻ nhưng rất tiềm năng này. Chúc các bạn thành công trên con đường mà mình đã chọn.

Mặc dù đã cố gắng để hoàn thành tốt nhất nội dung cho cuốn sách, tuy nhiên vẫn không tránh khỏi những thiếu sót. Các bạn có thể đóng góp ý kiến để nội dung ebook hoàn thiện hơn tại địa chỉ [Github IoT Maker Viet Nam](#)

Giấy phép sử dụng tài liệu.

creativecommons.org/licenses/by-nc-sa/4.0/

