ARDUINO

CHO NGƯỜI MỚI BẮT ĐẦU

(*Quyển căn bản*)

LỜI GIỚI THIỆU

Vi điều khiển là một thiết bị có thể lập trình được nhằm thực hiện các tác vụ định sẵn. Thường việc lập trình này dành cho các kỹ sư có trình độ kỹ thuật nhất định và thường rất khó khăn đối với các bạn học sinh tiếp cận. Nhờ sự ra đời của nền tảng Arduino để đơn giản hóa việc tiếp cận với vi điều dành cho một cá nhân chưa biết gì về lập trình vi điều khiển mà vẫn có thể tạo ra hàng nghìn ứng dụng khác nhau.

Quyển sách "**Arduino cho người mới bắt đầ**u" (quyển căn bản) được thiết kế cho các bạn chưa biết hoặc biết một ít về lập trình Arduino. Nội dung sách chủ yếu viết về định hướng suy nghĩ liên quan đến lập trình vi điều khiển nhằm mục tiêu phát triển lâu dài, tạo một nền tảng căn bản nhất cho các bạn làm quen với lập trình vi điều khiển. Bên cạnh đó, cung cấp cho bạn một số từ ngữ chuyên ngành thường dùng trong ngành lập trình vi điều khiển.

Quyển sách bao gồm 4 chương đi từ khái niệm căn bản đến hướng dẫn một số cách lập trình để có thể điều khiển được Arduino.

Chương 1 (*Một số khái niệm căn bản*) sẽ cho chúng ta một số khái niệm căn bản về lập trình vi điều khiển.

Chương 2 (*Lập trình C căn bản*) sẽ cung cấp một số kiến thức căn bản nhất về lập trình C hỗ trợ cho lập trình cho vi điều khiển.

Chương 3 (*Tổng quan về Arduino*) sẽ giới thiệu sơ lược về Arduino và các công cụ hỗ trợ lập trình.

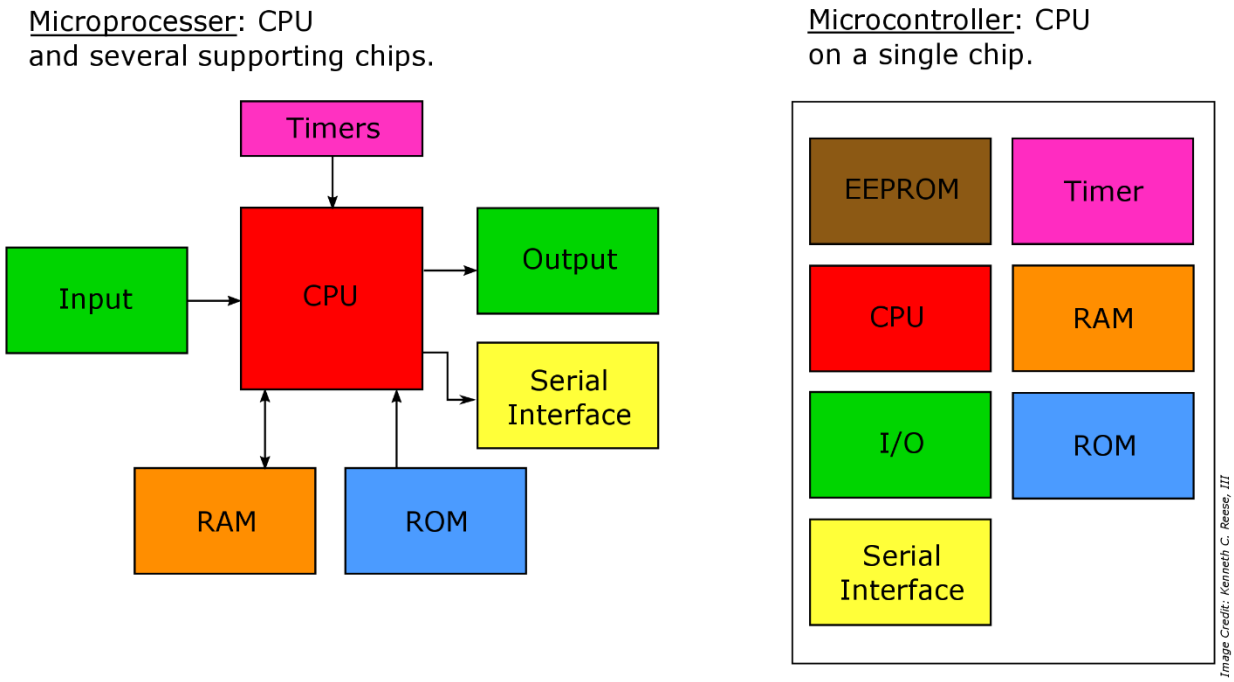
Chương 4 (*Một số module ngoại vi quan trọng*) sẽ cung cấp những kiến thức cốt lõi trong lập trình vi điều khiển nói chung và Arduino nói riêng.

MỤC LỤC

Chương 1 MỘT SỐ KHÁI NIỆM CĂN BẢN

1. Một số thuật ngữ

- Vi xử lý (MPU – MicroProcessing Unit/CPU – Central Processing Unit): là một máy tính, đơn vị tính toán thực hiện các phép toán logic theo chương trình được lập trình trước.
- Vi điều khiển: chứa một hoặc nhiều lõi vi xử lý và các module khác (RAM, ROM, Timer,...) trên cùng một chip.



- Arduino: là nền tảng vi điều khiển mã nguồn mở được sử dụng rộng rãi trên thế giới. Đây là dòng vi điều khiển khá đơn giản để bắt đầu vào con đường lập trình nhúng.
- Ngôn ngữ lập trình: là ngôn ngữ tuân theo các cú pháp đặt trước được dùng để lập trình ra các chương trình phần mềm/nhúng để thực hiện một mục đích nào đó.

2. Một số từ viết tắt

- MCU – MicroController Unit: vi điều khiển.
- CPU – Central Processing Unit: vi xử lý.

- SoC – System on chip: một con chip sẽ chứa nhiều module để có thể xây dựng một hệ thống.
- IDE - Integrated Development Environment: môi trường phát triển tích hợp.
- Code: chương trình.
- Func/Function: hàm.
- Var/Variable: biến.
- Decleration: khai báo.
- Initialization: khởi tạo.
- Build: dịch mã chương trình.
- Debug: gỡ lỗi.
- Program: nạp code.
- Pin: chân trên vi điều khiển.
- IO - Input and Output: vào và ra.
- Digital: số.
- Analog: tương tự.
- GPIO – General Peripheral Input/Output: các tín hiệu vào/ra của ngoại vi với mục đích chung.
- PWM – Pulse-Width Modulation: điều chế độ rộng xung.

Chương 2 LẬP TRÌNH C CĂN BẢN

1. Giới thiệu

- C là ngôn ngữ lập trình cấu trúc được sử dụng rộng rãi trong lập trình nhúng. Tài liệu này được viết ra nhằm tóm tắt một số lý thuyết cơ bản trong lập trình C nhằm mục đích cho việc phục vụ lập trình Arduino.

2. Các thành phần cơ bản trong ngôn ngữ C

- **Từ khóa** là từ có ý nghĩa xác định dùng để khai báo dữ liệu, viết hàm,... Ví dụ: `if`, `else`, `for`, `switch`,...
- **Tên** nhằm thể hiện rõ ý nghĩa của hằng, biến, mảng, con trỏ,... trong chương trình.
- Kiểu dữ liệu:

No	Type	Length	Range
1	unsigned char	1 byte	0 → 255
2	signed char	1 byte	-128 → 127
3	unsigned int	2 bytes	0 → 65,535
4	int	2 bytes	-32,768 → 32,767
5	unsigned long	4 bytes	0 → 2^{32}
6	long	4 bytes	-2^{31} → 2^{31}
7	unsigned long long	8 bytes	0 → 2^{64}
8	long long	8 bytes	-2^{63} → 2^{63}
9	float	4 bytes	$3.4 * 10^{-38}$ → $3.4 * 10^{38}$
10	double	8 bytes	$1.7 * 10^{-308}$ → $1.7 * 10^{308}$

- Khai báo và khởi tạo:
  - Khai báo (declaration):

```
int this_is_parameter;
```

  - Khởi tạo (initialization):

```
int this_is_parameter = 100;
```

3. Các hệ đếm

- Có các hệ đếm cơ bản: hệ nhị phân (hệ 2 – binary), hệ bát phân (hệ 8 – octal), hệ thập phân (hệ 10 – decimal) và hệ thập lục phân (hệ 16 – hexadecimal):

Hex	Decimal	Octal	Binary
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111
10	16	20	10000
20	32	40	100000
40	64	100	100000
80	128	200	10000000
100	256	400	100000000
200	512	1000	1000000000
400	1024	2000	10000000000

4. Biểu thức và phép toán

- Biểu thức là sự phối hợp của những toán tử và toán hạng.

```
a = b + 1
index += 2
```

- Các loại phép toán:
  - Phép toán số học: cộng (+), trừ (–), nhân (*), chia (/), chia lấy dư (%).

```
10 + 8 / 2 = 14
12 % 5 = 2
```

  - Phép quan hệ: lớn hơn (>), lớn hơn hoặc bằng (>=), bé hơn (<), bé hơn hoặc bằng (<=), bằng (==) và khác (!=).

`10 > 2` → có giá trị 1 (đúng)

`20 == 30` → có giá trị 0 (sai)

  - Phép toán luận lý: phủ định (!), và (&&) và hoặc (||).

`5 && (8 < 9)` → có giá trị 1 (đúng)

`1 || 0` → có giá trị 1 (đúng)

`1 && 0` → có giá trị 0 (sai)

  - Phép toán trên bit (bitwise): và (&), hoặc (|), xor (^), dịch trái (<<), dịch phải (>>), đảo (~).

Bit x	Bit y	x & y	x | y	x ^ y	~x
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

2_{10} >> 1 = 0010_2 >> 1 = 0001_2

2_{10} << 1 = 0010_2 << 1 = 0100_2

- Các phép toán khác: phép gán (=), phép gán kết hợp (+=, -=, *=, /=, %=, >>=, <<=, &=, |=, ^=), phép tăng giảm (++, --).
- Độ ưu tiên của các phép toán:

Độ ưu tiên	Các phép toán											Trình tự
1	()	[]	->									Trái → Phải
2	!	~	&	*	++	--						Phải → Trái
3	*	/	%									Trái → Phải
4	+	-										Trái → Phải
5	<<	>>										Trái → Phải
6	<	<=	>	>=								Trái → Phải
7	==	!=										Trái → Phải
8	&											Trái → Phải
9	^											Trái → Phải
10	|											Trái → Phải
11	&&											Trái → Phải
12	||											Trái → Phải
13	?	:										Phải → Trái
14	=	+=	-=	*=	/=	%=	<<=	>>=	&=	^=	|=	Phải → Trái

5. Một số hàm C thường dùng

a. Hàm rẽ nhánh có điều kiện (if ... else if ... else ....)

```
if  (biểu thức luận lý 1) {
   <khối lệnh 1>
}
else if (biểu thức luận lý 2) {
   <khối lệnh 2>
}
...
else if (biểu thức luận lý n) {
```

```
   <khối lệnh n>
}
else {
   <khối lệnh n + 1>
}
```

- Giải thích:

  Nếu (`if`) `biểu thức luận lý 1` đúng thì thực hiện `khối lệnh 1` và thoát khỏi cấu trúc if, ngược lại nếu (`else if`) `biểu thức luận lý 2` đúng thì `thực hiện khối lệnh 2` và thoát khỏi cấu trúc if,

  ...

  ngược lại (`else if`) nếu `biểu thức luận lý n` đúng thì thực hiện `khối lệnh n` và thoát khỏi cấu trúc if,

  ngược lại (`else`) `thực hiện khối lệnh n`.

- Lưu ý:
  - Từ khóa `if`, `else if`, `else` phải viết thường.
  - Biểu thức luận lý đúng khi kết quả khác 0 (`≠ 0`) và sai khi kết quả bằng 0 (`= 0`).

- Một số dạng rút gọn của cấu trúc if

```
if  (biểu thức luận lý 1) {
<khối lệnh 1>
}
```

hoặc

```
if  (biểu thức luận lý 1) {
<khối lệnh 1>
}
else {
<khối lệnh 2>
}
```

- Ví dụ:

```
int a;
<Lấy giá trị của a>
if (a > 0) {
      printf ("The value of %d is greater than 0", a);
}
else if (a == 0) {
      printf ("The value of %d is equal to 0", a);
}
else {
      printf ("The value of %d is less than 0", a);
}
```

b. Hàm rẽ nhánh có điều kiện (switch ... case ...)

```
switch (biểu thức) {
      case giá trị 1:
            <khối lệnh 1>
            break;
      case giá trị 2:
            <khối lệnh 2>
            break;
      ...
      case giá trị n:
            <khối lệnh n>
            break;
      default:
            <khối lệnh n+1>
}
```

- Giải thích:

  Trong trường hợp `biểu thức` có giá trị là

  `giá trị 1` (`case`) thì thực hiện `khối lệnh 1` và thoát ra khỏi cấu trúc switch (`break`),

hoặc `giá trị 2` (`case`) thì thực hiện `khối lệnh 2` và thoát ra khỏi cấu trúc switch (`break`),

...

hoặc `giá trị n` (`case`) thì thực hiện `khối lệnh n` và thoát ra khỏi cấu trúc switch (break),

ngược lại (`default`) thì thực hiện `khối lệnh n+1`.

- Lưu ý:
  - Từ khóa `switch`, `case`, `break`, `default` phải viết thường.
  - Biểu thức phải có kết quả là giá trị nguyên.
- Ví dụ:

```
int month = 0;
<Nhập tháng>
switch (month) {
     case 1:
          printf (“Tháng 1\n\r”);
          break;
     case 2:
          printf (“Tháng 3\n\r”);
          break;
     case 3:
          printf (“Tháng 3\n\r”);
          break;
     case 4:
          printf (“Tháng 4\n\r”);
          break;
     case 5:
          printf (“Tháng 5\n\r”);
          break;
     case 6:
          printf (“Tháng 6\n\r”);
```

```
            break;
      case 7:
            printf (“Tháng 7\n\r”);
            break;
      case 8:
            printf (“Tháng 8\n\r”);
            break;
      case 9:
            printf (“Tháng 9\n\r”);
            break;
      case 10:
            printf (“Tháng 10\n\r”);
            break;
      case 11:
            printf (“Tháng 11\n\r”);
            break;
      case 12:
            printf (“Tháng 12\n\r”);
            break;
      default:
            printf (“Nhập tháng sai!\n\r”);
            break;
}
```

c. Vòng lặp for

```
for (biểu thức 1; biểu thức 2; biểu thức 3) {
      <khối lệnh>
}
```

- Giải thích:

  Đầu tiên sẽ thực hiện `biểu thức 1`,

  Kiểm tra `biểu thức 2` nếu đúng thì thực hiện `khối lệnh`. Sau đó thực hiện `biểu thức 3`.

Kiểm tra `biểu thức 2` nếu vẫn còn đúng thì thực hiện `khối lệnh`. Sau đó thực hiện `biểu thức 3`.

...

Kiểm tra `biểu thức 2` nếu vẫn còn đúng thì thực hiện `khối lệnh`. Sau đó thực hiện `biểu thức 3`.

Kiểm tra `biểu thức 2` nếu sai thì thoát khỏi vòng for.

- Lưu ý:
  - Từ khóa `for` phải viết thường.
- Ví dụ:

```
int idx = 0;
for (idx = 0; idx < 10; idx++) {
      Printf ("Đây là vòng lặp thứ %d\n\r", idx + 1);
}
```

- Một số từ khóa đi với vòng lặp:
  - `break`: dùng để thoát vòng lặp gần nhất chứa nó.
  - `continue`: dùng để tiếp tục vòng lặp gần nhất, bỏ qua các lâu lệnh phía sau lệnh `continue`.
- Một số vòng lặp khác: ngoài ra còn có một số vòng lặp khác như `while`, `do .. while ...`

d. Hàm/Chương trình con

- Hàm được dùng để thực hiện một công việc nào đó cụ thể mà người dùng muốn đặt ra. Người lập trình thường viết chương trình lớn thành nhiều chương trình con để cho chương trình đỡ phức tạp và dễ dàng trong đọc hiểu, bảo trì code.
- Chương trình con có thể gọi một chương trình con khác. Nếu chương trình con gọi chính nó thì gọi là phương pháp đệ quy. Lưu ý khi lập trình đệ quy

là phải có điều kiện thoát, nếu không chương trình sẽ lập vô tận và sinh ra các kết quả không mong muốn.

- Ví dụ:

```
int Sum (int a, int b) {
      return (a + b);
}

void main() {
      int result = 0;
      result = Sum(2, 4);
      printf("2 + 4 = %d\n", result);
}
```

Ở chương trình trên thì hàm Sum là chương trình con.

e. Mảng một chiều

- Mảng là tập hợp nhiều phần tử có chung kiểu dữ liệu. Chỉ số của các phần tử trong mảng luôn bắt đầu là 0 và tăng dần 1 đơn vị cho các phần tử tiếp theo.
- Ví dụ: tạo ra mảng 10 phần tử có địa chỉ liên tiếp nhau có chung kiểu dữ liệu là int.

```
int array[10];
```

- Để tham chiếu phần tử trong mảng, ta thực hiện cú pháp sau:

  Tham chiếu phần tử thứ 2:

```
int value = array[2];
```

0	1	2	3	4	5	6	7	8	9

- Ngoài ra, chúng ta còn có mảng nhiều chiều. Bạn đọc có thể tìm hiểu thêm.

f. Con trỏ

- Con trỏ sẽ chứa địa chỉ dùng để truy cập các biến/hàm thông qua địa chỉ này.

- Ví dụ:

  `int par;` → Khai báo biến `par` có kiểu dữ liệu là `int`.

  `int * px;` → Khai báo con trỏ sẽ trỏ đến ô nhớ có kiểu dữ liệu là `int`.

  `px = &par;` → Lấy địa chỉ của biến `par` gán cho con trỏ `px`;

  `(* px) = 10;` → Gán `10` vào ô nhớ được chứa trong con trỏ `px` (tức là gán giá trị cho `par` thông qua con trỏ `px`).

g. Một số chia sẻ của tác giả

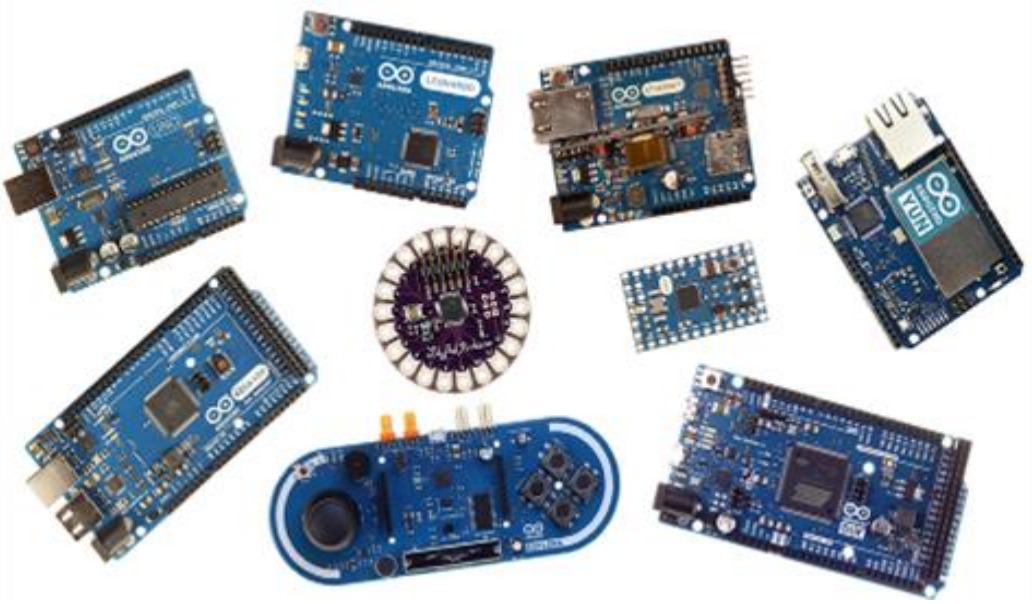
- Trên đây là những kiến thức cần biết của ngôn ngữ C để bước đầu có thể lập trình Arduino nhưng đó không phải là tất cả ngôn ngữ C. Bạn đọc nên tìm hiểu thêm về C và chi tiết hơn về các mục phía trên được tác giả liệt kê.

Chương 3

TỔNG QUAN VỀ ARDUINO

1. Phần cứng

- Hiện nay trên thị trường có rất nhiều dòng arduino khác nhau với các đặc tính và sức mạnh khác nhau như Arduino Uno R3, Arduino Mega 2560, Arduino Nano CH40,...



- Trong tài liệu này, chúng ta sẽ dùng Arduino Uno R3 trong suốt phần còn lại của tài liệu. Một số thông số kỹ thuật cần biết:

Vi điều khiển	**AT Mega328**
Nguồn (USB)	5V
Nguồn ngoài	7 – 12V
Số chân digital	14 (6 PWM)
Số chân analog	6
Dòng tối đa trên GPIO	40mA
Dòng tối đa từ chân cấp nguồn 3.3VDC	150mA
Flash	32 KB với 0.5 KB dùng cho boot loader.
SRAM	2KB

EEPROM	1KB

2. Phần mềm

- Arduino IDE là môi trường để lập trình và nạp code cho các dòng Arduino. Arduino IDE được xây dựng trên nền tảng Java nên hỗ trợ hầu hết các hệ điều hành hiện nay.

a. Cài đặt

- **Bước 1**: Để có thể dùng được Arduino IDE, chúng ta cần download JRE (Java Runtime Enviroment) cho máy tính. Bạn vào trang chủ của Oracle (www.oracle.com) và chọn file download phù hợp với hệ điều hành đang sử dụng.

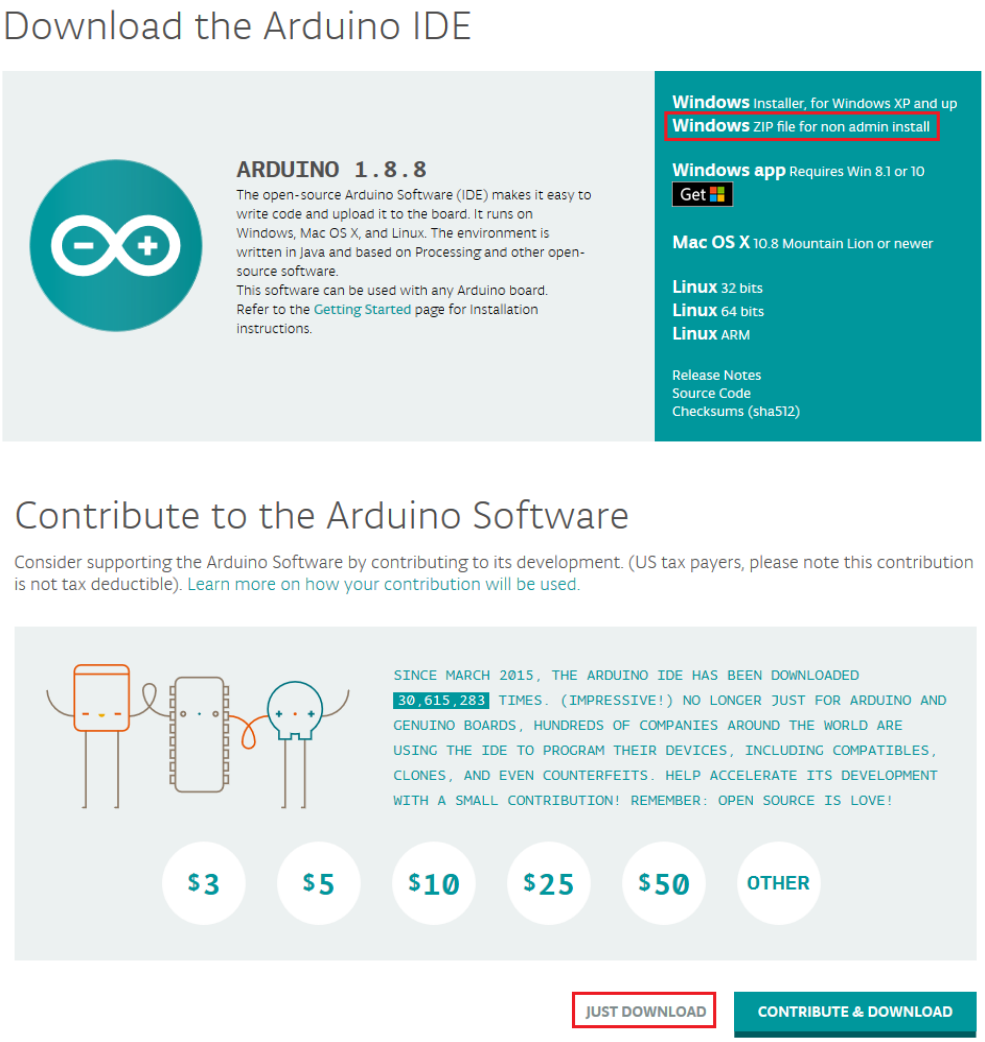
Java SE Runtime Environment 8u201

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement ◉ Decline License Agreement

Product / File Description	File Size	Download
Linux x86	68.1 MB	jre-8u201-linux-i586.rpm
Linux x86	83.8 MB	jre-8u201-linux-i586.tar.gz
Linux x64	64.91 MB	jre-8u201-linux-x64.rpm
Linux x64	80.73 MB	jre-8u201-linux-x64.tar.gz
Mac OS X x64	76.18 MB	jre-8u201-macosx-x64.dmg
Mac OS X x64	67.77 MB	jre-8u201-macosx-x64.tar.gz
Solaris SPARC 64-bit	46.27 MB	jre-8u201-solaris-sparcv9.tar.gz
Solaris x64	50.14 MB	jre-8u201-solaris-x64.tar.gz
Windows x86 Online	1.87 MB	jre-8u201-windows-i586-iftw.exe
Windows x86 Offline	63.53 MB	jre-8u201-windows-i586.exe
Windows x86	66.51 MB	jre-8u201-windows-i586.tar.gz
Windows x64	71.44 MB	jre-8u201-windows-x64.exe
Windows x64	71.29 MB	jre-8u201-windows-x64.tar.gz

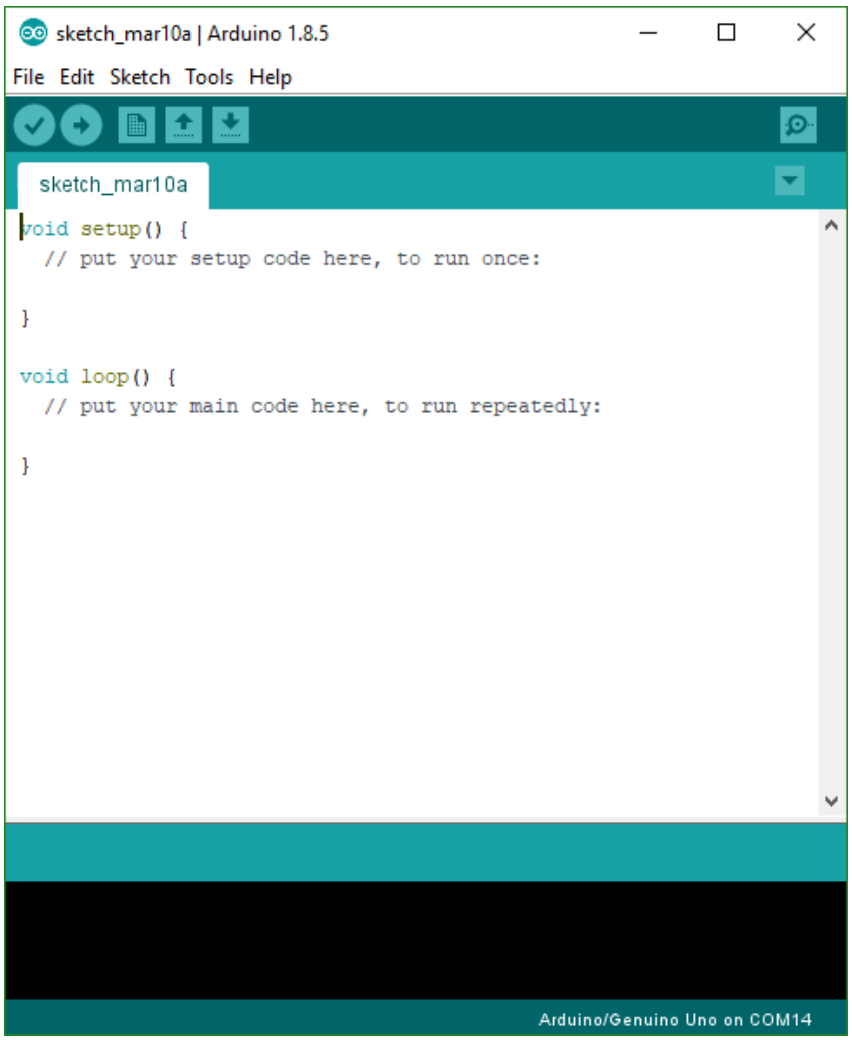
- **Bước 2**: Truy cập vào trang chủ của Arduino (https://www.arduino.cc-/en/Main/Software) và chọn file cài đặt phù hợp với hệ điều hành đang sử dụng.



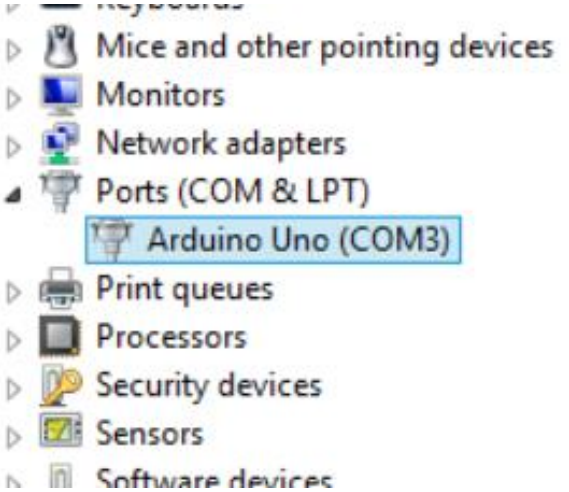
- Sau khi giải nén tập tin tải về, mở file arduino.exe để khởi động chương trình.

drivers	13-Apr-18 08:44 AM	File folder	
examples	13-Apr-18 08:45 AM	File folder	
hardware	13-Apr-18 08:45 AM	File folder	
java	13-Apr-18 08:46 AM	File folder	
lib	13-Apr-18 08:46 AM	File folder	
libraries	13-Apr-18 08:46 AM	File folder	
reference	13-Apr-18 08:46 AM	File folder	
tools	13-Apr-18 08:46 AM	File folder	
tools-builder	13-Apr-18 08:46 AM	File folder	
arduino.exe	02-Oct-17 15:37 PM	Application	395 KB
arduino.l4j.ini	02-Oct-17 15:37 PM	Configuration sett...	1 KB
arduino_debug.exe	02-Oct-17 15:37 PM	Application	393 KB
arduino_debug.l4j.ini	02-Oct-17 15:37 PM	Configuration sett...	1 KB
arduino-builder.exe	02-Oct-17 15:37 PM	Application	3,214 KB
libusb0.dll	02-Oct-17 15:37 PM	Application extens...	43 KB
msvcp100.dll	02-Oct-17 15:37 PM	Application extens...	412 KB
msvcr100.dll	02-Oct-17 15:37 PM	Application extens...	753 KB
revisions.txt	02-Oct-17 15:37 PM	Text Document	84 KB
wrapper-manifest.xml	02-Oct-17 15:37 PM	XML Document	1 KB

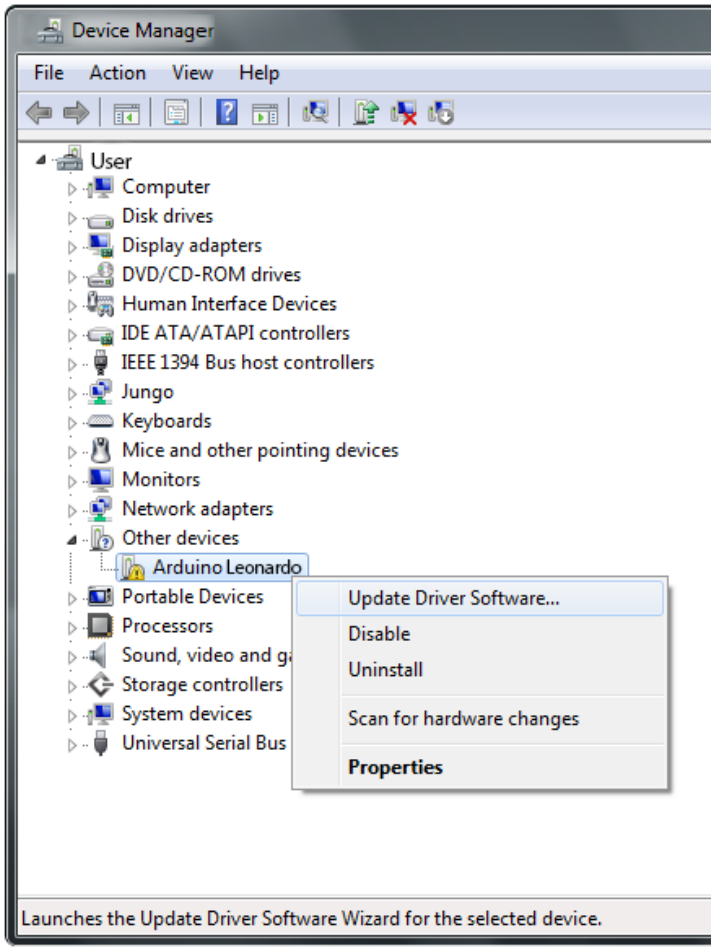
- Ta có giao diện chương trình như sau:



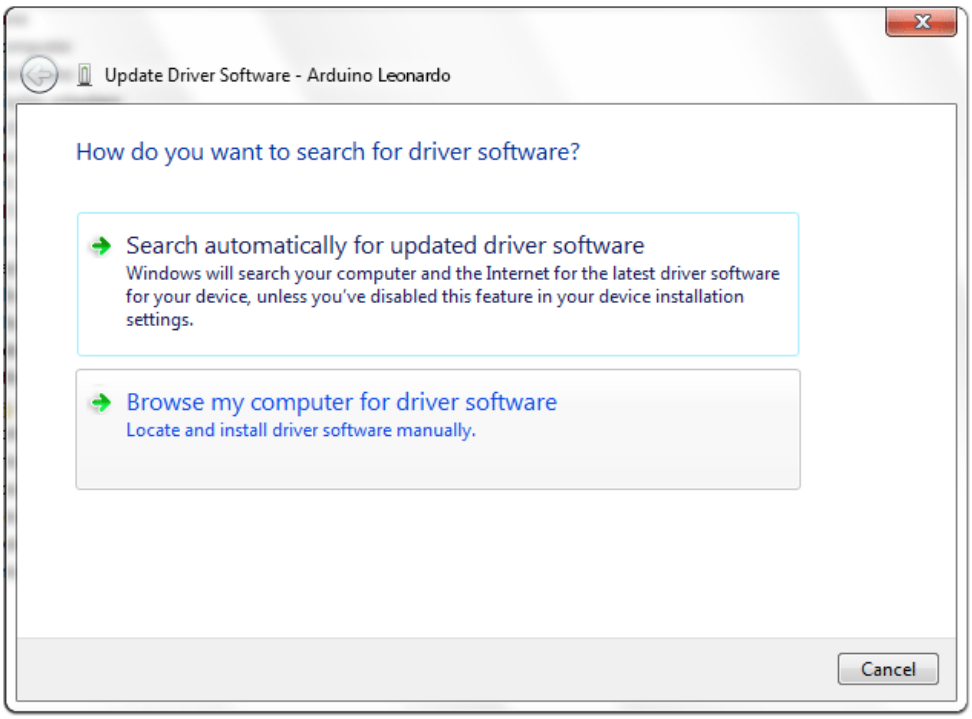
- **Bước 3**: Cài đặt driver:
  - Sử dụng cáp USB kết nối Arduino với máy tính, máy tính sẽ tự động nhận driver.



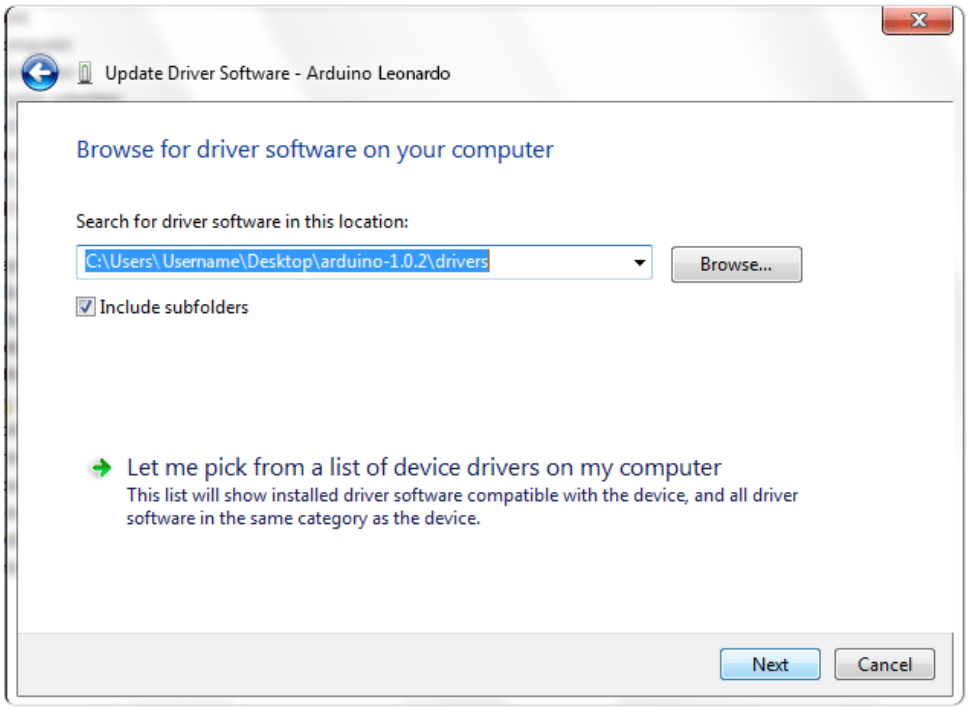
  - Nếu máy tính hiển thị thông báo ***"Device driver software was not successfully installed"*** thì thực hiện các bước sau:
  - ***Bước 3.1***: Mở cửa sổ **Run** (phím **windows + R**).
  - ***Bước 3.2***: Gõ `devmgmt.msc`. Cửa sổ được hiển thị như hình bên dưới. Sau đó, bấm chuột phải vào thiết bị là Arduino và chọn ***"Update Driver Software"***.



- **Bước 3.3**: Chọn ***"Browse my computer for driver software".***

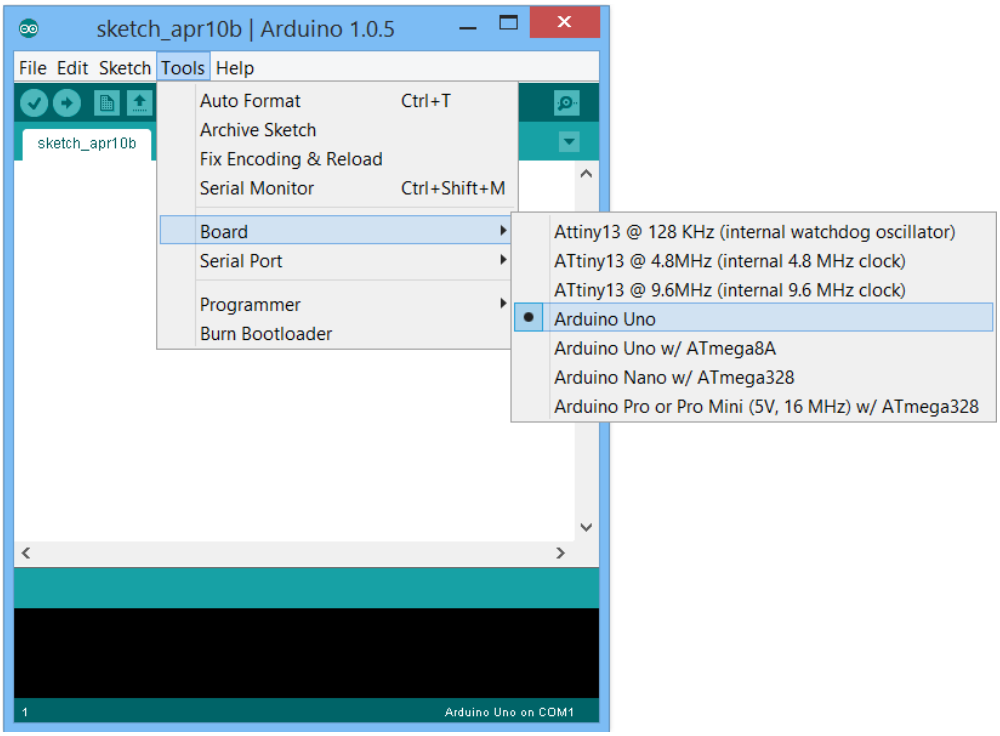


- **Bước 3.4**: Chọn nút nhấn ***Browse...*** và chọn đến nơi chứa driver (nằm trong thư mục tải về của Arduino).

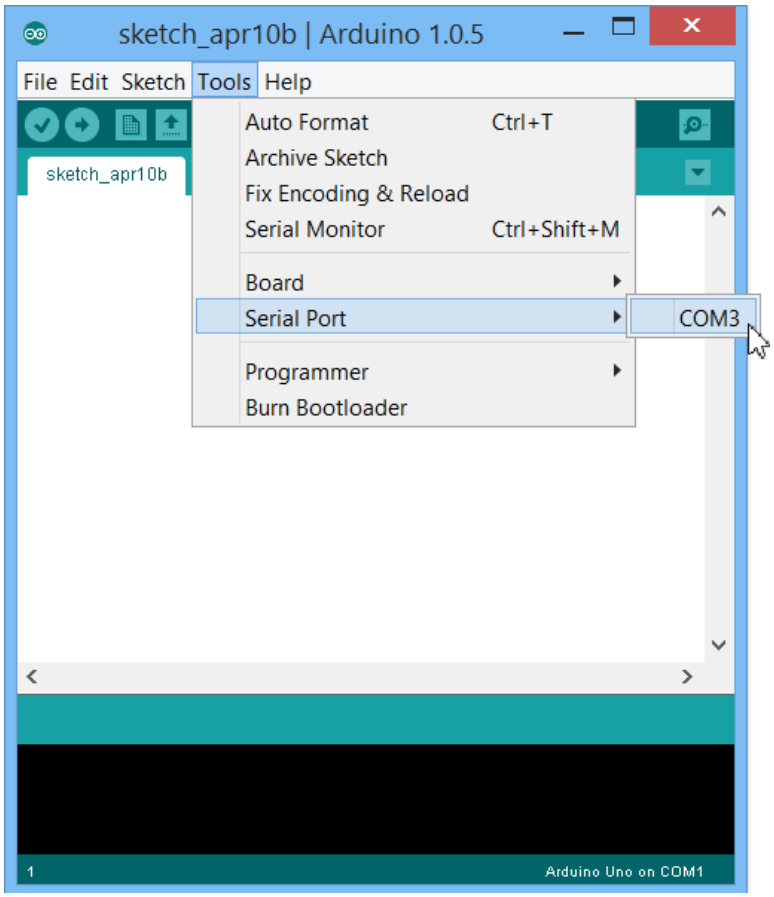


- **Bước 3.5:** Chọn ***Next*** và đợi cài đặt driver.
- **Bước 3.6**: Sau khi thành công, chúng ta sẽ thấy xuất hiện cổng COM như ở đầu bước 3.

- **Bước 4**: Chọn loại board tương ứng. Chúng ta mở chương trình Arduino IDE và vào ***Tools*** → ***Board*** → Chọn board đang sử dụng (ở đây là ***Arduino Uno***).



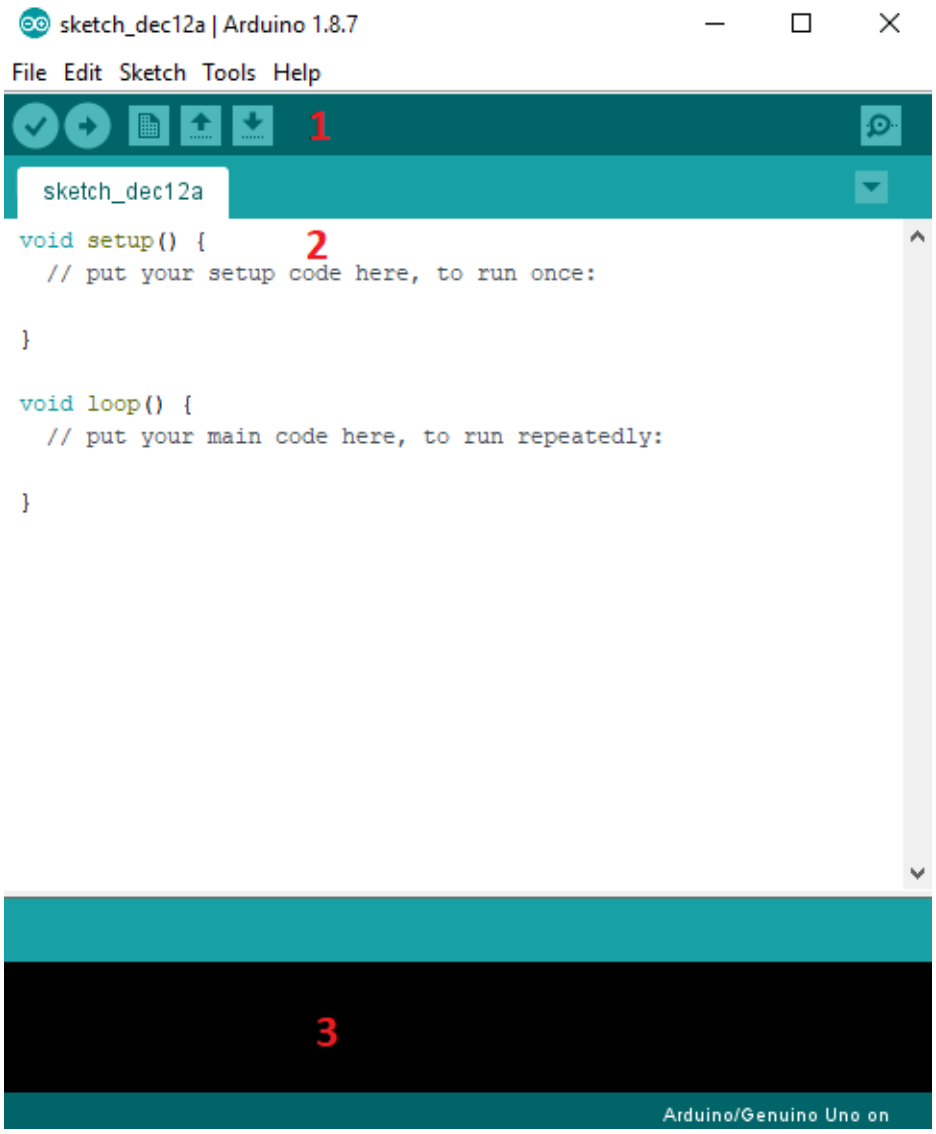
- **Bước 5**: Cấu hình cổng COM để nạp code cho Arduino. Chọn ***Tools*** → ***Serial Port*** → ***COMxx*** (chọn cổng COM tương ứng với board Arduino).



- **Lưu ý**: Nếu chúng ta gắn nhiều board vào máy tính đồng thời thì chỉ có thể nạp cho một board duy nhất (cho tới phiên bản hiện tại 1.8.8), vì thế ta nên xác định đúng cổng COM với board ta cần nạp.

b. Giới thiệu về Arduino IDE

- Giao diện được chia thành 3 vùng chính:



- **Vùng 1**: Các phím chức năng

Icon	Chức năng
	Biên dịch chương trình đang soạn thảo để kiểm tra các lỗi lập trình.
	Biên dịch và upload chương trình đang soạn thảo.
	Mở một trang soạn thảo mới.
	Mở các chương trình đã lưu.
	Lưu chương trình đang soạn.
	Mở cửa sổ Serial Monitor để gửi và nhận dữ liệu giữa máy tính và board Arduino.

- **Vùng 2**: Cửa sổ để viết chương trình
  - Chương trình (code) sẽ được viết tại đây. Ở đây có 2 hàm quan trọng là `setup()` và `loop()`.
  - Hàm `setup()` sẽ được khởi chạy một lần duy nhất. Chức năng của hàm này dùng để khởi tạo các biến, khai báo chức năng, khởi tạo các thông số cho các module trong Arduino.
  - Hàm `loop()` là nơi chương trình được chạy và lặp đi lặp lại cho đến khi tắt nguồn vi điều khiển.
- **Vùng 3**: Hiển thị các thông tin liên quan đến chương trình
  - Là cửa sổ để hiển thị các về việc build chương trình, nạp chương trình thành công xuống vi điều khiển và các cảnh báo khác liên quan đến chương trình và vi điều khiển của chúng ta. Lưu ý, mọi thông báo và trạng thái của cả quá trình viết chương trình (write code), xây dựng chương trình (build code) và nạp chương trình (program code) đều được hiển thị tại đây. Của sổ này còn được gọi là của sổ debug.

Chương 4 MỘT SỐ MODULE NGOẠI VI QUAN TRỌNG

Bài 1 GENERAL PURPOSE INPUT/OUTPUT – GPIO

1. Giới thiệu

- General Purpose Input/Output (GPIO) là module dùng đọc tín hiệu vào hay xuất tín hiệu ra một hoặc một số chân vi điều khiển. GPIO chỉ làm việc với 2 mức điện áp (điện áp thấp và điện áp cao) tương ứng với mức logic 1 (mức điện áp cao – tùy thuộc vào vi điều khiển: có thể là 1.8V, 3.3V, 5V,...) và mức logic 0 (mức điện áp thấp – thường là 0V).
- Xuất tín hiệu logic ra ngoài (output) là đưa điện áp mức cao (mức logic 1) hay điện áp thấp (mức logic 0).
- Đọc tín hiệu điện áp vào có kết quả là mức logic 0/1 (input) nhằm để xác định xem điện áp tại chân vi điều khiển tương ứng là mức thấp hay cao.
- Một số đặc tính khi chân của vi điều khiển được cấu hình là `INPUT`: cấu hình mặc định của các chân Arduino (Atmega) là input, vì thế chúng ta không cần khai báo khi hàm `pinMode()` khi sử dụng chúng như một chân input. Chân vi điều khiển được cấu hình bằng cách này sẽ ở trạng thái trở kháng cao (high-impedance state) – khoảng 100MΩ.
- Thêm trở kéo lên/xuống cho một chân của vi điều khiển khi được cấu hình là `INPUT`: một điều như không bắt buộc nhưng được xem là thông lệ (nên được xem là bắt buộc) là phải xác định trạng thái của chân (được cấu hình là input) nếu không có tín hiệu input từ ngoài vào. Việc này được thực hiện bằng cách thêm một điện trở kéo lên (lên mức điện áp cao - +5V đối với Arduino Atmega) hoặc điện trở kéo xuống (xuống đất – 0V). Giá trị điện trở này thường được dùng là 10 KΩ.
- Đặc tính của chân của vi điều khiển khi được cấu hình là `INPUT_PULLUP`: bên trong vi điều khiển Atmega đã có một điện trở 20K được kéo lên nguồn có thể truy cập bằng phần mềm. Để truy cập được điện

trở được tích hợp này thì ta khai báo `pinMode()` với biến là `INPUT_PULLUP`.

- Đặc tính của chân của vi điều khiển khi được cấu hình là `OUTPUT`: khi được cấu hình là `OUTPUT` thì chân này ở trạng thái trở kháng thấp (low-impedance state). Điều này có nghĩa là vi điều khiển có thể cung cấp một lượng dòng điện lớn (vài milliampere đến vài chục milliampere) ra ngoài. Các chân của vi điều khiển Atmega có thể dùng mạch kiểu source (cung cấp dòng điện dương) hoặc kiểu sink (cung cấp dòng điện âm) lên đến 40mA. Với lượng dòng điện này có thể đủ làm sáng LED (phải mắc nối tiếp điện trở để hạn dòng) hoặc một vài cảm biến nhưng không đủ dòng để bật tắt relay (rờ-le), motor.
- *Lưu ý*: việc ngắn mạch hoặc để các thiết bị tiêu thụ dòng cao trên chân của Arduino có thể hỏng một chân hoặc cả con vi điều khiển. Vì vậy, khi kết nối bất cứ thiết bị nào vào vi điều khiển phải xem xét đến dòng điện tiêu thụ.

2. Một số hàm thường dùng

a. `pinMode()`

- Chức năng: dùng để cấu hình một chân là input hay output.
- Cú pháp:

```
pinMode(pin, mode);
```

  - `pin`: chân ta muốn cấu hình.
  - `mode`: `INPUT`, `OUTPUT` hoặc `INPUT_PULLUP`.

- Kết quả trả về: không.
- Ví dụ: thiết lập chân 13 là ouput.

```
pinMode(13, OUTPUT);    // sets the digital pin 13 as output
```

- *Lưu ý*: các chân analog vẫn có thể sử dụng như digital (input/output).

b. `digitalWrite()`

- Chức năng: ghi mức logic `HIGH` (cao) hoặc `LOW` (thấp) ra chân được chỉ định.
- Cú pháp:

```
digitalWrite(pin, value);
```

  - `pin`: chân ta muốn ghi mức giá trị.
  - `value`: `HIGH` hoặc `LOW`.
  - Kết quả trả về: không.

- Lưu ý:
  - Nếu không thiết lập `pinMode()` là `OUTPUT` và kết nối đèn LED vào pin này, khi gọi hàm `digitalWrite(HIGH)` thì đèn LED có thể sẽ sáng mờ. Nguyên nhân là ta không thiết lập `pinMode` là `OUTPUT` thì mặc định là `INPUT` và gọi hàm `digitalWrite(HIGH)` sẽ bật điện trở nội kéo lên nên sẽ xem như đèn LED mắc nối tiếp với trở hạn dòng có giá trị lớn.
  - Các chân analog vẫn có thể sử dụng như chân digital (ngoại trừ chân A6, A7 của Arduino Mini và của board Arduino Nano, Pro Mini).

c. `digitalRead()`

- Chức năng: đọc giá trị logic từ chân được chỉ định.
- Cú pháp:

```
digitalRead(pin);
```

  - `pin`: chân mà ta muốn đọc.

- Giá trị trả về: `HIGH` hoặc `LOW`.
- *Lưu ý*:
  - Nếu chân được đọc tín hiệu mà không được nối vào đâu thì giá trị trả về có thể là `HIGH` hoặc `LOW` (và có thể thay đổi đột ngột).

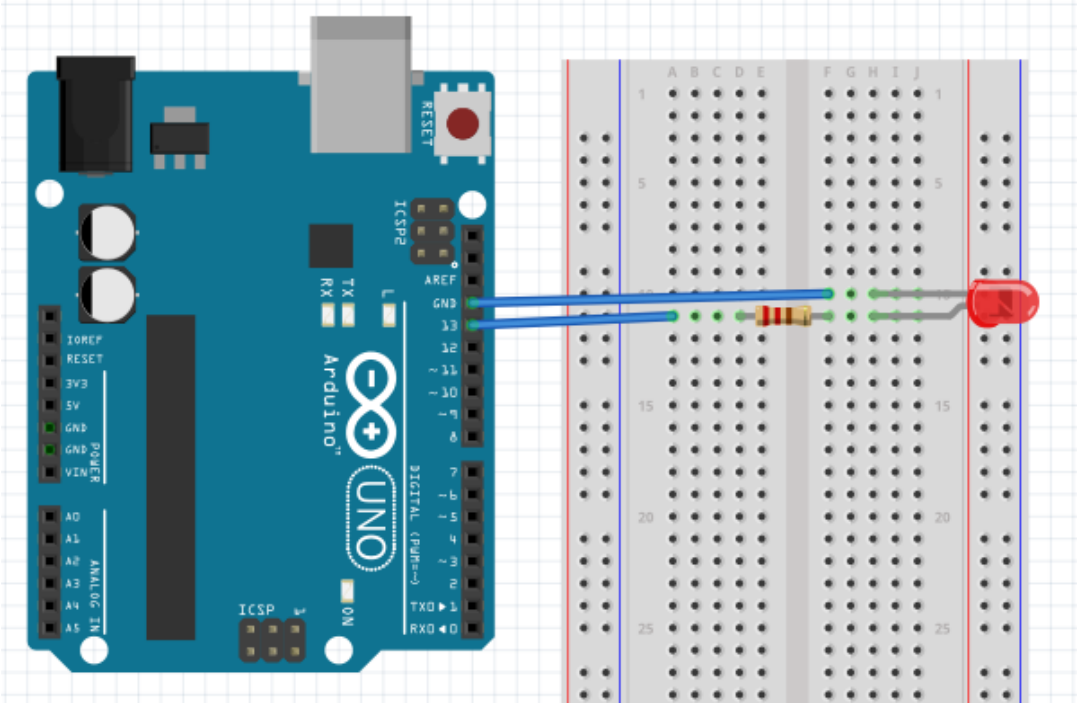
- Các chân analog vẫn có thể sử dụng như chân digital (ngoại trừ chân A6, A7 của Arduino Mini và của board Arduino Nano, Pro Mini).

3. Một số module mẫu

a. Output

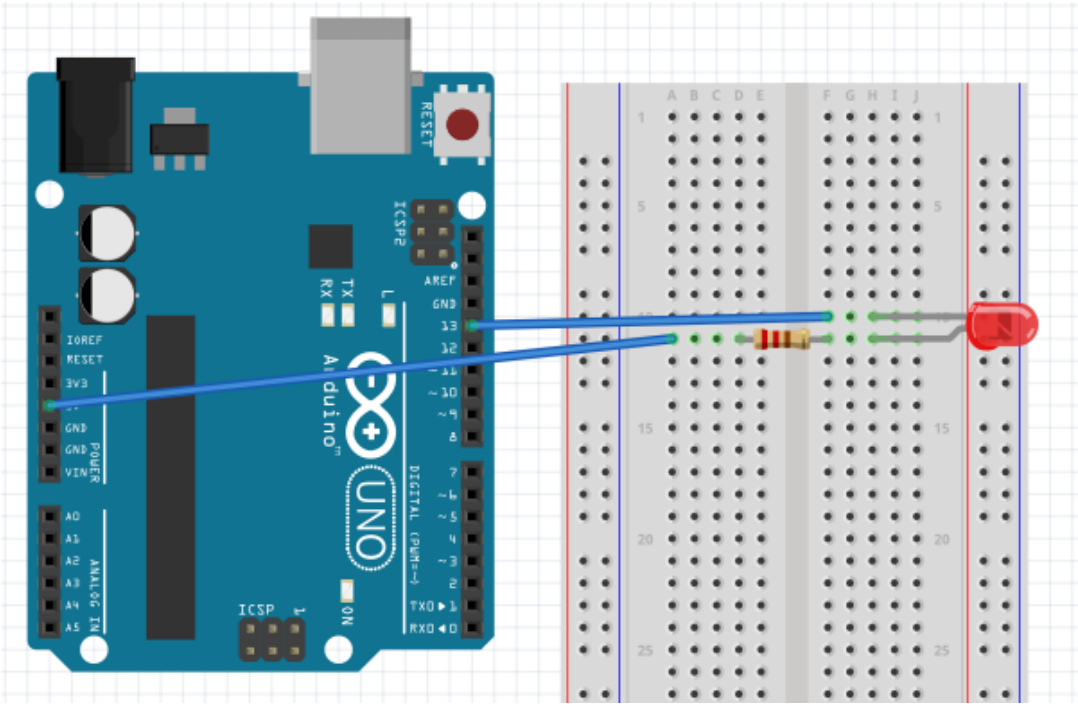
i. Đèn LED

- Chức năng: thường dùng để làm đèn tín hiệu.
- Cách lập trình: ta sẽ xem xét hai mạch sau để cùng bật một đèn LED.



- Theo như hình trên, đèn LED sáng khi điện áp xuất ra từ Arduino là điện áp cao.

```
void setup() {
  pinMode(13, OUTPUT);        // sets the digital pin 13 as output
}
void loop() {
  digitalWrite(led, HIGH);  // sets the digital pin 13 on
}
```



- Theo như hình trên, đèn LED sáng được khi điện áp xuất ra từ Arduino là điện áp thấp.

```
void setup() {
  pinMode(13, OUTPUT);        // sets the digital pin 13 as output
}
void loop() {
  digitalWrite(led, LOW);  // sets the digital pin 13 off
}
```

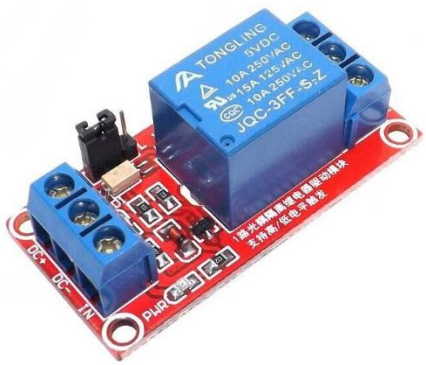
- Để bật/tắt một thiết bị, chúng ta cần biết được đặc tính của thiết bị đó là để hoạt động cần mức điện áp thấp (mức logic 0) hay điện áp cao (mức logic 1).

ii. Module Relay

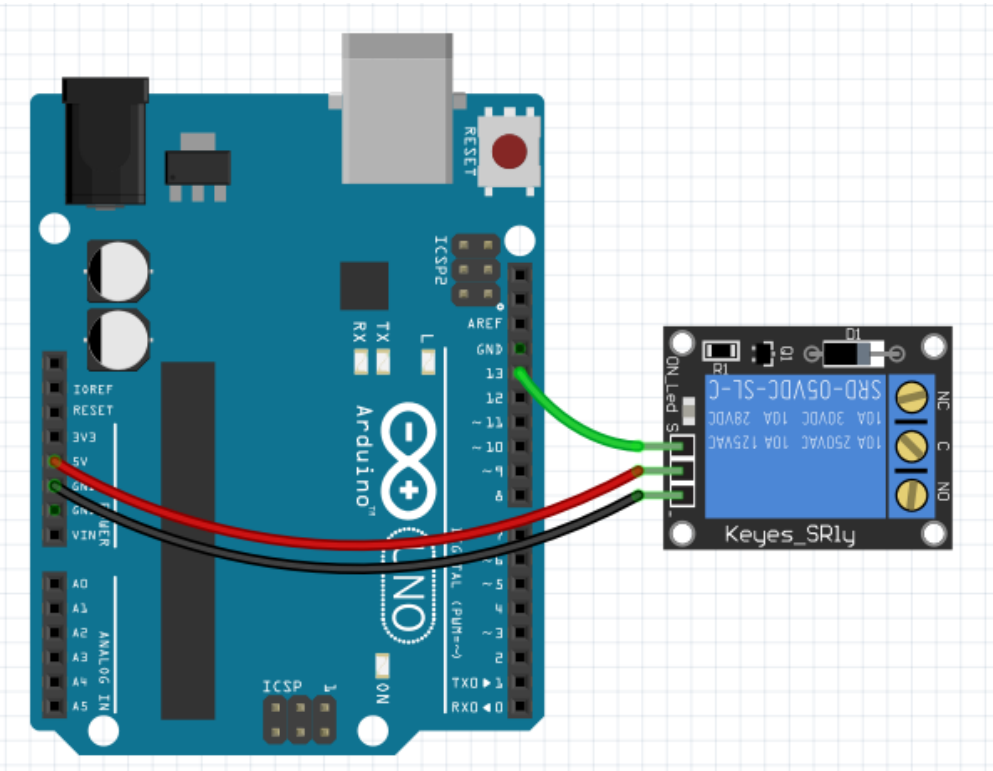
- Chức năng: dùng để bật tắt các thiết bị có công suất lớn/điện áp lớn/dòng điện lớn.
- Sơ đồ chân:
  - Hiện nay, trên thị trường có rất nhiều loại module relay khác nhau với sơ đồ nguyên lý khác nhau (cách ly hoặc không cách ly/tích cực mức

thấp hoặc cao), điện áp hoạt động khác nhau (5V, 12V hoặc 24V) nên khi sử dụng ta cần chú ý đọc kỹ hướng dẫn trước khi sử dụng.

- Thông thường, module sẽ có 2 domino (hoặc 1 header và 1 domino) để điều khiển module. Một domino để cấp nguồn (sẽ có 2 chân DC+ và DC- hoặc GND và VCC) và tín hiệu điều khiển bật tắt relay (IN). Domino còn lại dùng để điều khiển các thiết bị công suất với 3 ngõ COM (cổng chung), NO (normal open – thường mở) và NC (normal close – thường đóng). Nếu relay được kích (hoạt động) thì COM sẽ nối với NO, ngược lại nếu relay không được kích thì COM sẽ nối với NC.



- Cách kết nối với Arduino:



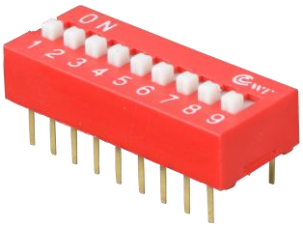
- Cách lập trình: tương tự như lập trình với LED.

b. Input

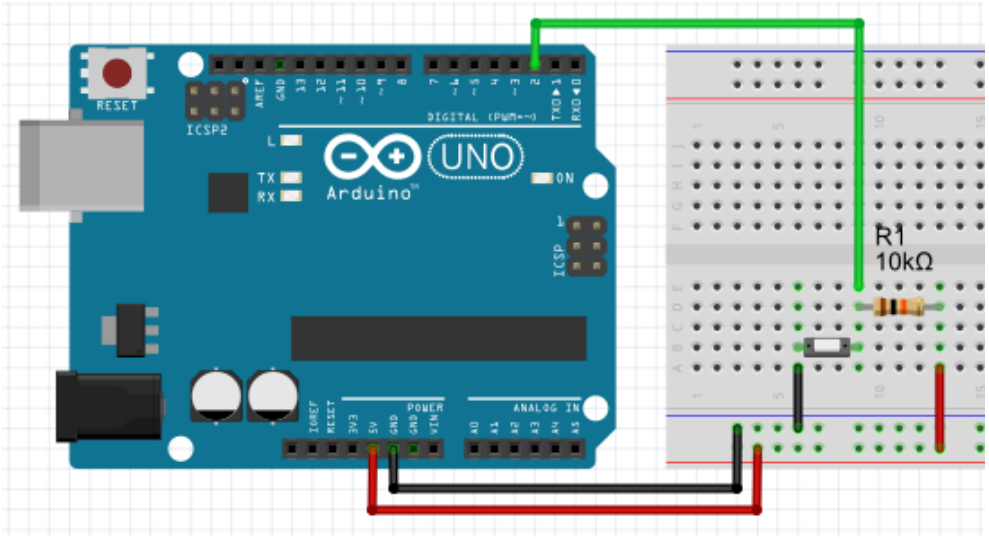
i. Nút nhấn (button)

- Chức năng: dùng để chọn chế độ hoạt động, nhận một tín hiệu điều khiển từ người dùng mà vi điều khiển có thể hiểu được.
- Sơ đồ chân: nút nhấn được chia thành hai loại cơ bản là push button và switch.
    - Push button: sau khi tác động lực thì nút nhấn quay về trạng thái ban đầu, trạng thái đóng/mở có thể được giữ hoặc không giữ.

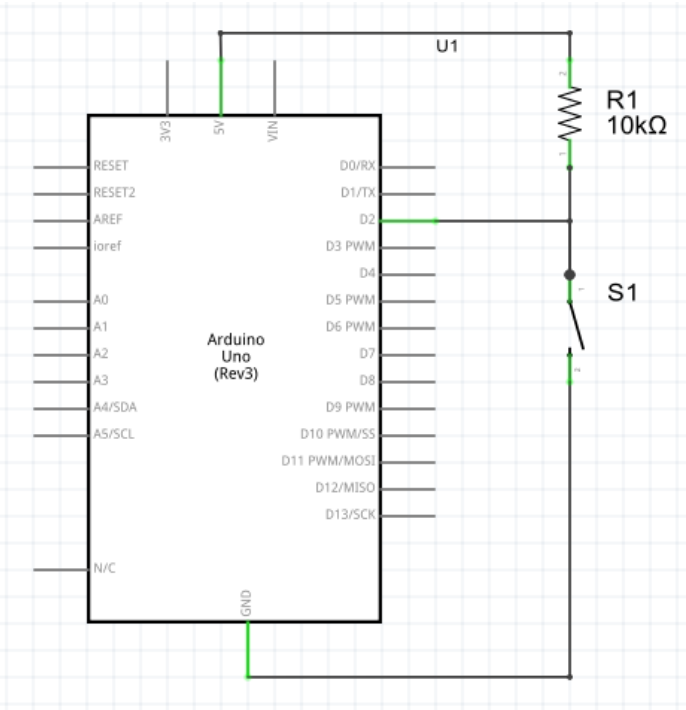
    - Switch: sau khi tác động lực để thay đổi trạng thái thì nút nhấn không quay về trạng thái ban đầu, trạng thái của nút nhấn được giữ khi ngừng tác dụng lực



- Cách lập trình: chúng ta cùng xem xét hai mạch đọc nút nhấn sau.



- Từ mạch trên, ta có sơ đồ nguyên lý như sau:



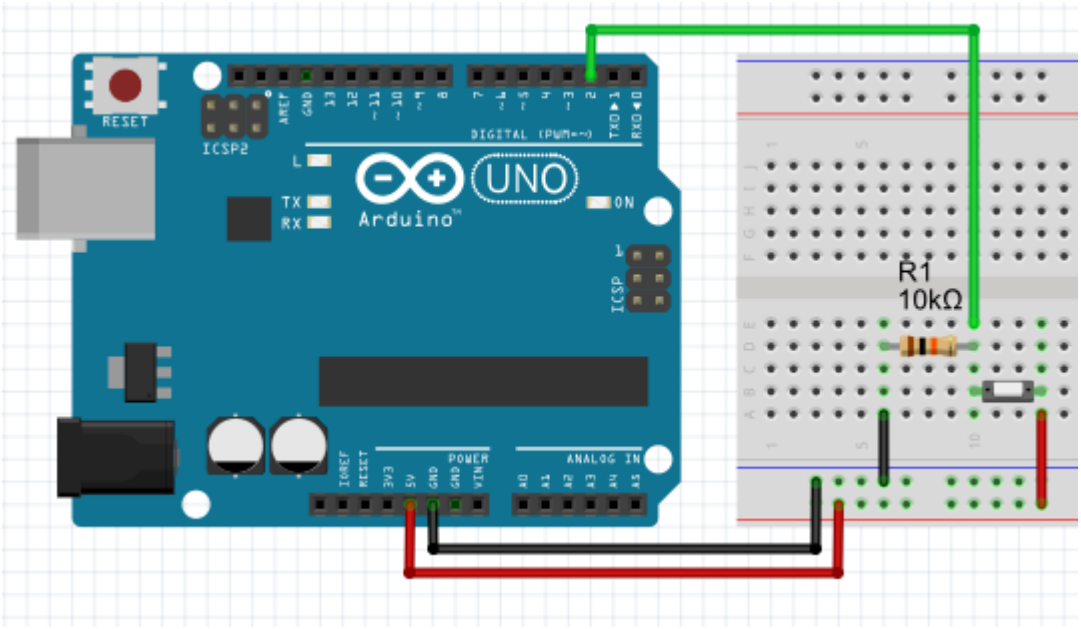
- Xét trạng thái tại chân D2, giả sử chân D2 chỉ nhận mức tín hiệu điện áp và không có dòng điện đi vào chân D2.
- Nếu S1 không được nhấn thì điện áp tại chân D2 là 5V. $I_{R1} = 0A$ vì đã giả sử không có dòng điện vào chân D2.

$$U_{D2} = 5V - I_{R1}R1 = 5V - 0(A).10(k\Omega) = 5\text{V}$$

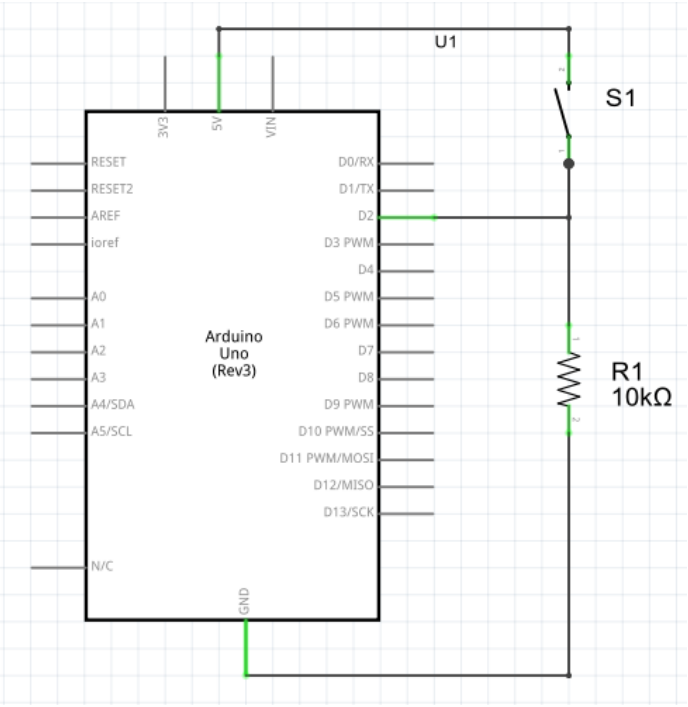
- Ngược lại, nếu S1 được nhấn thì chân D2 được nối với 0V nên điện áp đọc vào là 0V.

Trạng thái nút nhấn	Giá trị điện áp	Mức logic
Không nhấn	5V	HIGH
Nhấn	0V	LOW

```
int buttonPin = 2;     // the number of the pushbutton pin
int buttonState = 0;   // the state of pushbutton
void setup() {
  // Initialize the pushbutton pin as an input
  pinMode(buttonPin, INPUT);
}
void loop() {
  buttonState = digitalRead(buttonPin);
  if (buttonState == LOW) {
    // The pushbutton is pressed
  }
  else {
    // The pushbutton isn't pressed
  }
}
```



- Ta xét mạch điện thứ 2 như hình trên, ta có sơ đồ nguyên lý như sau:



- Tương tự, ta cũng xét trạng thái tại chân D2.
- Nếu S1 không được nhấn thì điện áp tại chân D2 là 0V. $I_{R1} = 0A$ vì đã giả sử không có dòng điện vào chân D2.

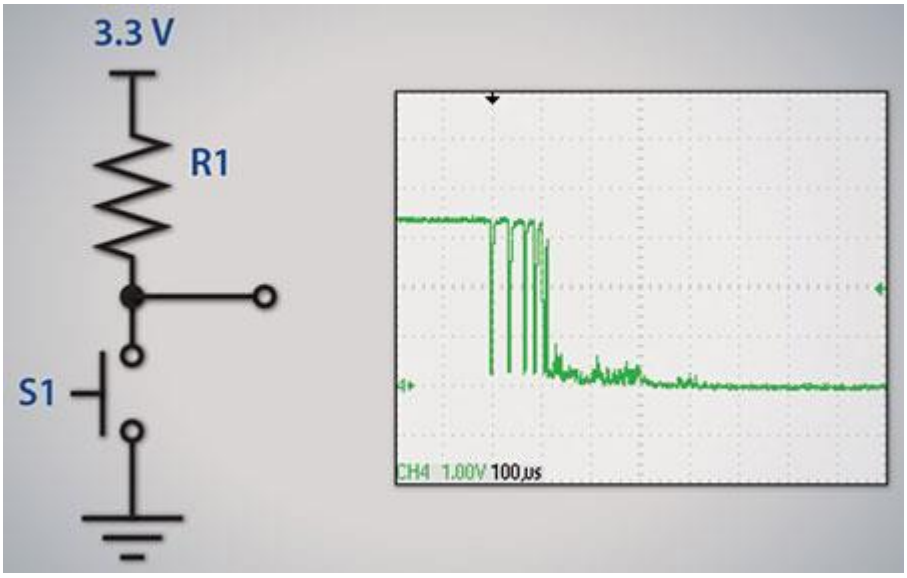
$$U_{D2} = I_{R1}R1 = 0(A).10(k\Omega) = \ 0\text{V}$$

- Ngược lại, nếu S1 được nhấn thì chân D2 được nối với 5V nên điện áp đọc vào là 5V.

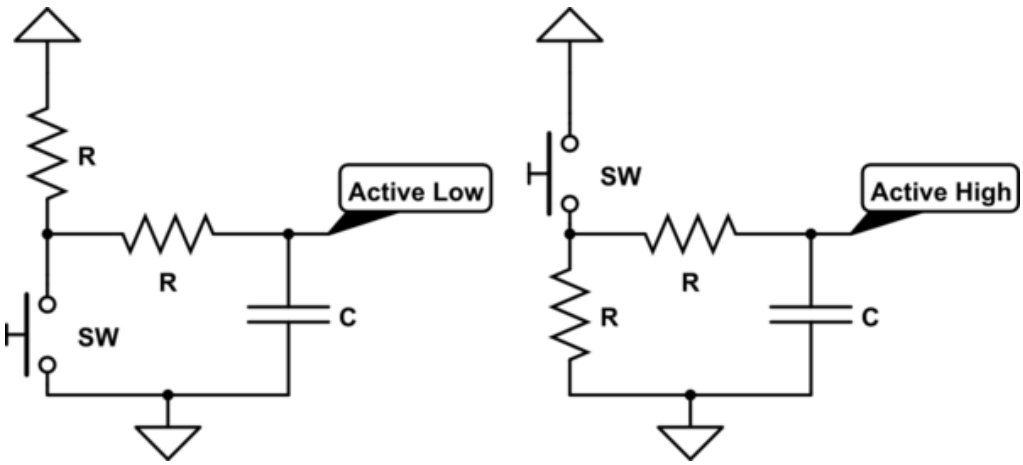
Trạng thái nút nhấn	Giá trị điện áp	Mức logic
Không nhấn	0V	LOW
Nhấn	5V	HIGH

```
int buttonPin = 2;     // the number of the pushbutton pin
int buttonState = 0;   // the state of pushbutton
void setup() {
  // Initialize the pushbutton pin as an input
  pinMode(buttonPin, INPUT);
}
void loop() {
  buttonState = digitalRead(buttonPin);
  if (buttonState == HIGH) {
    // The pushbutton is pressed
  }
  else {
    // The pushbutton isn't pressed
  }
}
```

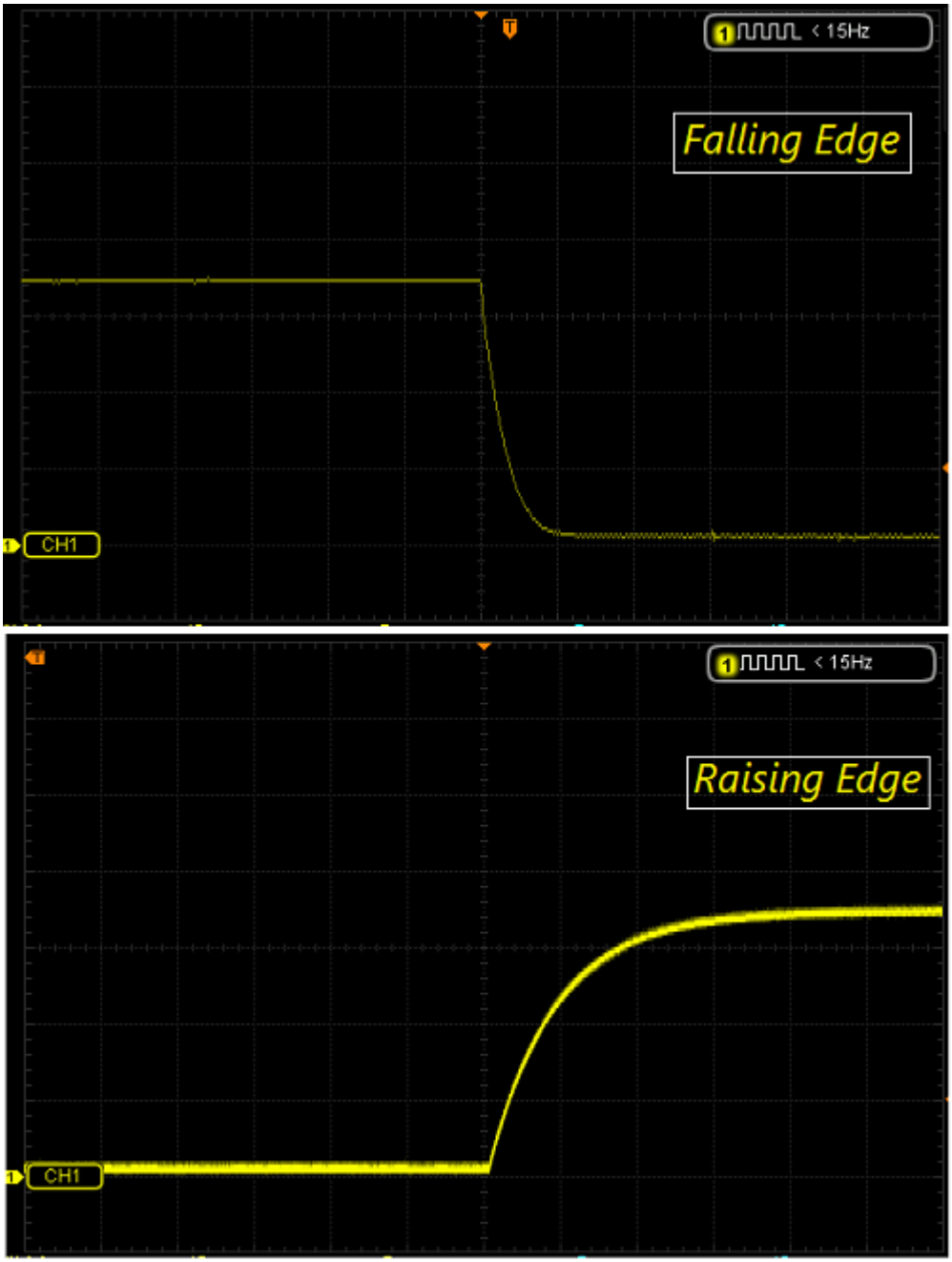
- Lưu ý: Trên đây là hai cách mắc nút nhấn cơ bản thường được sử dụng. Bên cạnh đó, ta xem xét một khía cạnh khác là tín hiệu điện áp đọc vào có ổn định không? Do cơ cấu cơ học của nút nhấn nên sẽ sinh ra hiện tượng rung phím khi nhấn nên tín hiệu đọc vào không được ổn định khi mới bắt đầu nhấn.



- Để giải quyết hiện tượng này, ta có cách chống rung bằng phần cứng hoặc phần mềm.
  - Phần mềm: khi phát hiện nút nhấn được nhấn thì ta đợi một khoảng thời gian sau (thường khoảng 20ms) thì mới đọc tín hiệu để điện áp được ổn định.
  - Phần cứng: ta mắc thêm tụ vào để tạo ra mạch lọc thông thấp nhằm hạn chế những gai nhiễu do nút nhấn tạo ra.



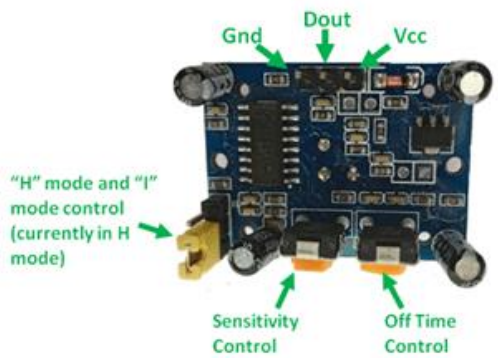
- Tín hiệu điện áp sau khi mắc thêm tụ đã không còn những gai nhiễu không mong muốn.

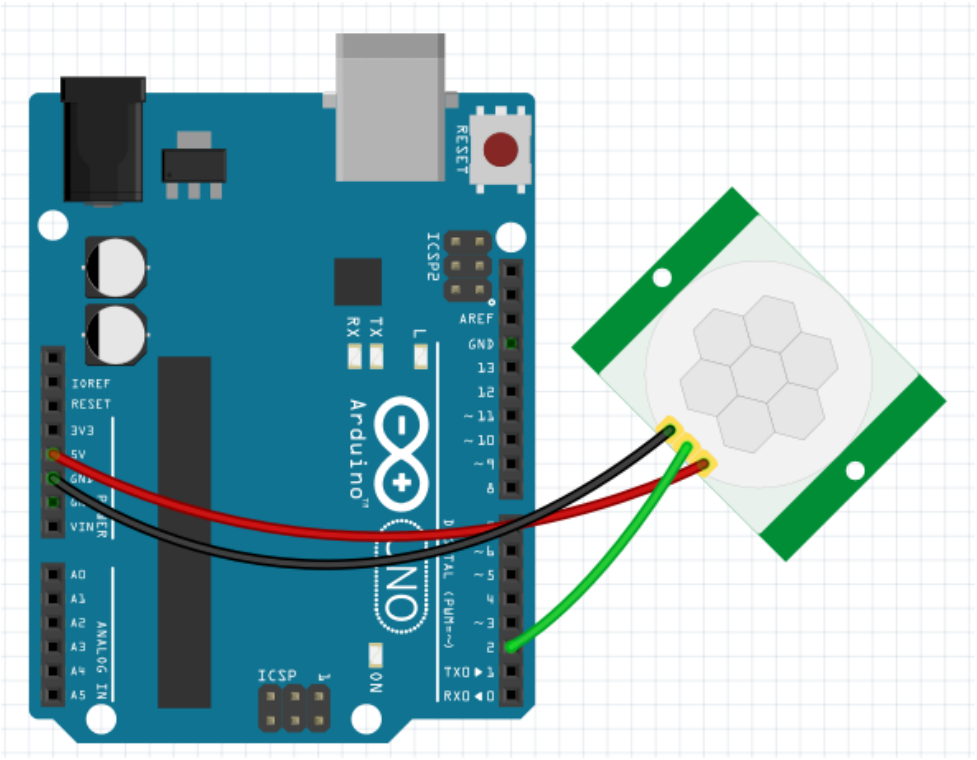


ii. Cảm biến chuyển động (PIR)

- Chức năng: được sử dụng để phát hiện chuyển động của các vật thể phát ra hồng ngoại (con người, con vật, các vật phát nhiệt,...); cảm biến có thể chỉnh được độ nhạy cũng như thời gian kích trễ.

- Sơ đồ chân:
  - VCC/GND: chân cấp nguồn.
  - Dout: chân tín hiệu dạng digital. Khi phát hiện có vật thể thì chân Dout sẽ ở mức điện áp cao, thời gian chuyển từ mức điện áp cao sang thấp tùy thuộc vào chế độ hoạt động (xem bên dưới).
  - Sensitivity control: điều khiển độ nhạy của cảm biến theo khoảng cách.
  - Off time control: thời gian trễ.
- Chế độ hoạt động:
  - Single trigger (L mode): chân Dout sẽ xuống mức điện áp thấp sau thời gian trễ khi phát hiện sự chuyển động lần đầu tiên.
  - Repeatable trigger (H mode): chân Dout sẽ xuống mức điện áp thấp sau thời gian trễ khi phát hiện sự chuyển động lần cuối cùng.





- Cách lập trình: tương tự như lập trình với nút nhấn. Giả sử đang ở mode H, khi đọc được mức logic cao thì có sự chuyển động và mức logic thấp thì không có chuyển động.

4. Lời kết

- Một vấn đề cần quan tâm là thiết bị/module của chúng ta cần được cấp dòng bao nhiêu là đủ để hoạt động? Đây là một vấn đề thiết yếu vì khả năng cấp dòng của vi điều khiển tối đa là 40mA đối với mạch Arduino UNO R3 IC cắm và những dòng vi điều khiển khác nhau thì sẽ có dòng cấp khác nhau.
- Vấn đề đặt ra là nếu chúng ta muốn bật tắt một thiết bị/module có dòng tiêu thụ lớn hơn dòng định mức của vi điều khiển thì cần phải làm gì? Để giải quyết được vấn đề này, chúng ta cần đến một mạch đệm dòng điện cho các thiết bị tiêu thụ dòng điện lớn.

Bài 2 TIME

1. Giới thiệu

- Time (thời gian) trong lập trình vi điều khiển khá quan trọng vì giúp cho vi điều khiển nhận biết thời gian nó cần phải thực hiện/thời gian đã thực hiện một nhiệm vụ mà người lập trình đưa ra.

2. Một số hàm thường dùng

a. `delay()`

- Chức năng: dùng để dừng chương trình trong khoảng thời gian xác định (đơn vị: milliseconds).
- Cú pháp:

```
delay(ms);
```

  - `ms`: số milliseconds cần dừng chương trình (kiểu dữ liệu: `unsigned long`).

- Kết quả trả về: không.
- Ví dụ:

```
delay(1000);     // wait for a second
```

- Lưu ý: trong thời gian thực hiện hàm delay thì vi điều khiển không thực hiện bất cứ chương trình nào khác như đọc giá trị cảm biến, tính toán hay thao tác trên các chân. Tuy nhiên, hàm delay sẽ không vô hiệu các ngắt (interrupt) nên các giao tiếp nối tiếp bằng phần cứng vẫn hoạt động (đường nhận dữ liệu), phát xung PWM hay trạng thái các Pin không thay đổi. Vì vậy, chúng ta nên thật sự lưu ý khi dùng các hàm delay.Ngoài ra, hàm delay cũng được dùng như một cách chống rung phím bằng phần mềm.

b. `delayMicroseconds()`

- Chức năng: dùng để dừng chương trình trong khoảng thời gian xác định (đơn vị: microseconds).

- Cú pháp:

```
delayMicroseconds(us);
```

  - `us`: số microseconds cần dừng chương trình (kiểu dữ liệu: `unsigned long`).

- Kết quả trả về: không.

- Ví dụ:

```
delayMicroseconds(1000);    // wait for a millisecond
```

- Lưu ý: hàm này làm việc chính xác khi `us` > 3 microsenconds.

c. `millis()`

- Chức năng: trả về số milliseconds kể từ khi bắt đầu hoạt động. Giá trị này sẽ bị tràn và quay về 0 khi đạt giá trị tối đa (khoảng 50 ngày).

- Cú pháp:

```
time = millis();
```

- Kết quả trả về: số milliseconds (kiểu dữ liệu: `unsigned long`).

- Ví dụ:

```
unsigned long time;

void loop() {
  time = millis();
  delay(1000);          // wait a second
}
```

- Lưu ý: kiểu dữ liệu là `unsigned long` nên khi ta thực hiện phép toán thì phải đảm bảo chung kiểu dữ liệu để không gây ra lỗi sai về giá trị tối đa của các kiểu dữ liệu.

d. `micros()`

- Chức năng: trả về số microseconds kể từ khi bắt đầu hoạt động. Giá trị này sẽ bị tràn và quay về 0 khi đạt giá trị tối đa (khoảng 70 phút).

- Cú pháp:

```
time = micros();
```

- Kết quả trả về: số microseconds (kiểu dữ liệu: `unsigned long`).

- Ví dụ:

```
unsigned long time;


void loop() {
  time = micros();
  delay(1000);          // wait a second
}
```

3. Lời kết

- Khi sử dụng các hàm delay thì ta cần phải cân nhắc có nên sử dụng hay không vì nó sẽ làm ảnh hưởng đến tốc độ chương trình.

- Ta cũng có thể sử dụng các phương pháp khác để thay thế các hàm delay như tính toán thời gian dựa vào việc lấy các hàm giá trị thời gian hiện tại.

Bài 3 UART

1. Giới thiệu

- UART là một chuẩn giao tiếp nối tiếp để giao tiếp giữa các thiết bị với nhau.
- Trong Arduino, UART được dùng để giao tiếp giữa board Arduino với các module khác có cùng giao thức hoặc giữa board Arduino với các board vi điều khiển khác.
- Giao thức UART sẽ truyền với 2 dây tín hiệu là TX và RX. Khi 2 board muốn nói chuyện với nhau thì phải mắc chéo (nghĩa là TX thằng này sẽ nối với RX thằng kia, RX thằng này nối với TX thằng kia).
- Một thông số quan trọng là tốc độ truyền (baudrate), các thông số này là những số được quy định trước 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200,...

2. Một số hàm thường dùng

a. `Serial.begin()`

- Chức năng: thiết lập baudrate (và các thông số khác) cho giao tiếp nối tiếp. Đây là thiết lập baudrate cho module phần cứng (chân 0 và 1 trên vi điều khiển). Thông số khác như số lượng data, parity và stop bits (giá trị mặc định là 8 data bits, no parity và 1 stop bit).
- Cú pháp:

```
Serial.begin(speed);
Serial.begin(speed, config);
```

  - `Serial`: đối tượng cổng nối tiếp (serial port).
  - `speed`: baudrate (bits per second – kiểu dữ liệu: `long`).
  - `config`: `SERIAL_5N1`, `SERIAL_6N1`, `SERIAL_7N1`, `SERIAL_8N1` (the default), `SERIAL_5N2`, `SERIAL_6N2`, `SERIAL_7N2`, `SERIAL_8N2`, `SERIAL_5E1`, `SERIAL_6E1`, `SERIAL_7E1`, `SERIAL_8E1`, `SERIAL_5E2`,

SERIAL_6E2, SERIAL_7E2, SERIAL_8E2, SERIAL_5O1, SERIAL_6O1, SERIAL_7O1, SERIAL_8O1, SERIAL_5O2, SERIAL_6O2, SERIAL_7O2, SERIAL_8O2.

- Kết quả trả về: không.
- Ví dụ:

```
void setup() {
  // opens serial port, sets data rate to 9600 bps
  Serial.begin(9600);
}

void loop() {}
```

b. `Serial.end()`

- Chức năng: ngừng giao tiếp nối tiếp.
- Cú pháp:

```
Serial.end();
```

- Kết quả trả về: không.

c. `Serial.print()/Serial.println()`

- Chức năng: in dữ liệu ra thông qua cổng nối tiếp với các ký tự nhìn thấy được (human-readble ASCII text). Với lệnh `Serial.println()` thì sau dữ liệu in ra thì có kèm theo 2 ký tự: ký tự trả về đầu dòng (a carriage return character - ASCII 13 – '`\r`') và ký tự xuống hàng (a newline character – ASCII 10 – '`\n`').
- Cú pháp:

```
Serial.print(val);
Serial.print(val, format);
```

  - `Serial`: đối tượng cổng nối tiếp.
  - `val`: giá trị cần in ra (với bất kỳ kiểu dữ liệu nào).
  - `format`: định dạng dữ liệu của `val` (xem ví dụ).

- Kết quả trả về: số bytes dữ liệu được ghi ra cổng nối tiếp (kiểu dữ liệu: `size_t`).

- Ví dụ:

```
void setup() {
  Serial.begin(9600); // open the serial port at 9600 bps:
}

void loop() {
  Serial.print(78);              // gives "78"
  Serial.println();              // enter new line
  Serial.print(1.23456);         // gives "1.23"
  Serial.println();              // enter new line
  Serial.print('N');             // gives "N"
  Serial.println();              // enter new line
  Serial.print("Hello world.");  // gives "Hello world."
  Serial.println();              // enter new line

  Serial.println(78, BIN);       // gives "1001110"
  Serial.println(78, OCT);       // gives "116"
  Serial.println(78, DEC);       // gives "78"
  Serial.println(78, HEX);       // gives "4E"
  Serial.println(1.23456, 0);    // gives "1"
  Serial.println(1.23456, 2);    // gives "1.23"
  Serial.println(1.23456, 4);    // gives "1.2346"
  delay (1000);                  // wait a sencond
}
```

d. `Serial.write()`

- Chức năng: ghi dữ liệu nhị phân (binary data) thông qua cổng nối tiếp. Dữ liệu này được gửi dạng byte hoặc chuỗi các bytes.

- Cú pháp:

```
Serial.write(val);
Serial.write(str);
Serial.write(buf, len);
```

- `Serial`: đối tượng cổng nối tiếp.
- `val`: giá trị cần gửi đi – 1 byte.
- `str`: chuỗi các ký tự gửi đi – chuỗi các bytes.
- `buf`: mảng các byte cần gửi.
- `len`: chiều dài chuỗi dữ liệu cần gửi.

- Kết quả trả về: số bytes dữ liệu được ghi ra cổng nối tiếp (kiểu dữ liệu: `size_t`).
- Ví dụ:

```
void setup() {
  Serial.begin(9600); // open the serial port at 9600 bps:
}


void loop() {
  Serial.write(45); // send a byte with the value 45


  int bytesSent = Serial.write("hello");  /* Send the string
"hello" and return the length of the string.*/
  delay (1000);             // wait a sencond
}
```

- Lưu ý: nếu bộ đệm (buffer) đủ khoảng trống để truyền thì hàm `Serial.write()` sẻ trả về trước khi tất cả các ký tự được gửi, các ký tự gửi đi sẽ được lưu trữ trong bộ đệm. Nếu độ đệm truyền bị đầy thì hàm này sẽ khóa (block) mãi cho đến khi đủ khoảng trống cho việc truyền. Để tránh trường hợp bị khóa (block) thì nên kiểm tra trước số byte còn trống trong bộ đệm bằng hàm `Serial.availableForWrite()`.

e. `Serial.availableForWrite()`

- Chức năng: số bytes còn lại trong bộ đệm truyền.
- Cú pháp:

```
Serial.availableForWrite();
```

  - `Serial`: đối tượng cổng nối tiếp.

- Kết quả trả về: số bytes có sẵn để ghi.

f. `Serial.available()`

- Chức năng: số bytes có sẵn trong bộ đệm nhận (bộ đệm tối đa là 64 bytes).
- Cú pháp:

```
Serial.available();
```

  - `Serial`: đối tượng cổng nối tiếp.

- Kết quả trả về: số bytes có sẵn để đọc.
- Ví dụ:

```
void setup() {
  Serial.begin(9600); // open the serial port at 9600 bps:
}

void loop() {
  // reply only when you receive data:
  if (Serial.available() > 0) {
  // Read data at here. Please see next example.
  }
}
```

g. `Serial.read()`

- Chức năng: đọc một byte dữ liệu đến.
- Cú pháp:

```
Serial. read();
```

  - `Serial`: đối tượng cổng nối tiếp.

- Kết quả trả về: ký tự đầu tiên trong chuỗi dữ liệu nhận được (kiểu dữ liệu: `int`).

- Ví dụ:

```
int incomingByte = 0; // for incoming serial data

void setup() {
  // opens serial port, sets data rate to 9600 bps
  Serial.begin(9600);
}

void loop() {
  // send data only when you receive data:
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();

    // say what you got:
    Serial.print("I received: ");
    Serial.println(incomingByte, DEC);
  }
}
```

h. Một số hàm khác:

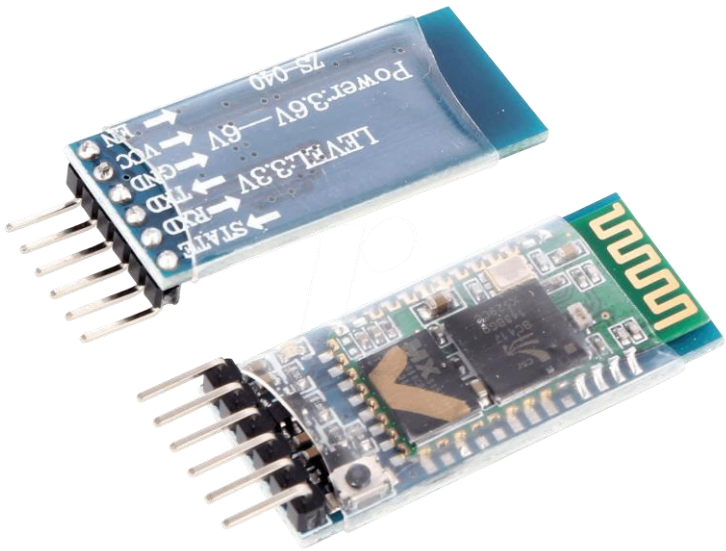
- Ngoài các hàm trên, Serial vẫn còn nhiều hàm khác sử dụng cho những trường hợp khác:
  - `if(Serial)`: kiểm tra cổng nối tiếp có sẵn sàng chưa.
  - `Serial.find()`: tìm xem trong bộ đệm có ký tự muốn tìm hay không bằng cách đọc dữ liệu trong bộ đệm đến khi tìm được ký tự mong muốn.

- `Serial.findUntil()`: tìm chuỗi ký tự cần tìm có tồn tại trong bộ đệm hay không bằng cách đọc dữ liệu trong bộ đệm đến khi tìm được chuỗi mục tiêu hoặc chuỗi để kết thúc sự tìm kiếm.
- `Serial.flush()`: đợi cho việc truyền hoàn tất cho tất cả các ký tự trong bộ đệm.
- `Serial.parseFloat()`: trả về giá trị có kiểu dấu chấm động phù hợp đầu tiên trong bộ đệm. Các ký tự không phải số hoặc dấu trừ sẽ bị bỏ qua. Hàm này sẽ chấm dứt khi gặp ký tự đầu tiên không phải dấu chấm động hoặc hết thời gian (thời gian này được thiết lập bởi hàm `Serial.setTimeout()`).
- `Serial.parseInt()`: tương tự như hàm `Serial.parseFloat()` nhưng giá trị cần đọc là kiểu `int`.
- `Serial.peek()`: trả về ký tự tiếp theo trong bộ đệm mà không có xóa nó đi ra khỏi bộ đệm.
- `Serial.readBytes()`: đọc về số byte dữ liệu được xác định trước.
- `Serial.readBytesUntil()`: đọc về số byte dữ liệu được xác định trước và phát hiện ký tự kết thúc hoặc hết thời gian định trước.
- `Serial.readString()`: đọc dữ liệu trong bộ đệm dưới kiểu dữ liệu là `String`.
- `Serial.readStringUntil()`: đọc dữ liệu trong bộ đệm dưới kiểu dữ liệu là `String` cho đến khi ký tự kết thúc được phát hiện hoặc hết thời gian định trước.
- `Serial.setTimeout()`: thiết lập thời gian tối đa cho việc đợi dữ liệu từ cổng nối tiếp. Giá trị mặc định là 1 giây.
- `serialEvent()`: hàm này được gọi khi có dữ liệu đến.

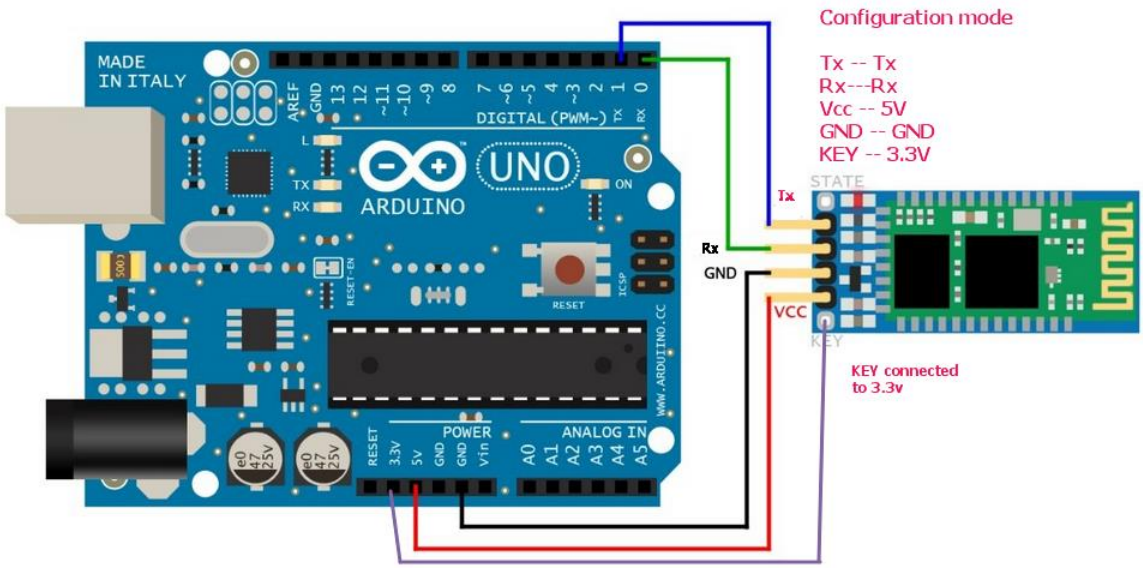
- Để biết thêm thông tin chi tiết, vui lòng truy cập vào link https://www.arduino.cc/reference/en/language/functions/communication/serial/

3. Một số module mẫu

a. Module bluetooth HC-05



- Chức năng: là module chuyển đổi chuẩn giao tiếp nối tiếp sang bluetooth. Khi máy tính kết nối module này thì máy tính nhận là một cổng COM ảo.
    - Giao tiếp bằng bluetooth là giao tiếp tầm gần (dưới 10m).
- Sơ đồ chân:



- Kết nối như hình trên, ta có thể không cần thiết phải nối chân KEY với 3.3V.
- Trên mặc định của bluetooth là HC-05, mật khẩu kết nối là 1234 và baudrate mặc định là 9600.
- Chúng ta có thể thay đổi cấu hình của module bluetooh bằng chế độ AT command, baudrate cho chế độ này là 38400. Bạn có thể tìm kiếm trên mạng bằng từ khóa "hc05 at commands".
- Lưu ý, TX và RX phải được mắc chéo.

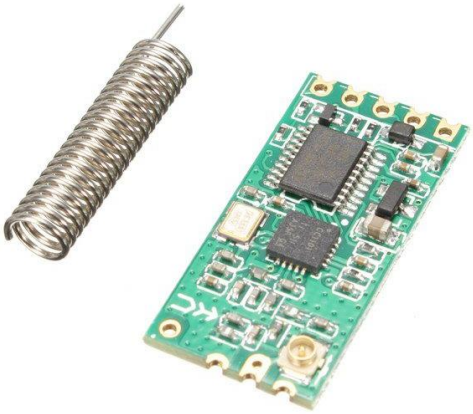
- Cách lập trình:

```
int incomingByte = 0; // for incoming serial data


void setup() {
  Serial.begin(9600); // opens serial port, sets data rate to
9600 bps
}


void loop() {
  Serial.println("Nguyen Binh Khiem, Vinh Long");
  // send data only when you receive data:
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();


    // say what you got:
    Serial.print("I received: ");
    Serial.println(incomingByte, DEC);
  }
  delay(1000);
}
```

- Lưu ý: đây là chuẩn bluetooth 2.0 nên chỉ giao tiếp 1 – 1 (tức là một master và 1 slave giao tiếp với nhau). Nếu bạn muốn tìm hiểu về nhiều module

giao tiếp với nhau có thể tham khảo với từ khóa “module bluetooth 4.0” hoặc “module bluetooth BLE”.

b. Module RF HC-11



- Chức năng: mạch thu phát RF UART C1101 (HC-11) truyền nhận ở tần số 433Mhz với khoảng cách lý tưởng lên đến 200m (khoảng cách truyền nhận xa hơn bluetooth).
    - Khi giao tiếp với module này cần phải lưu ý là thiết lập cùng kênh sóng, cùng địa chỉ kênh nhưng khác địa chỉ module.
    - Thay đổi thiết lập này bạn cũng vào chế độ AT như module bluetooth HC-05, bạn có thể tham khảo với từ khóa “module uart cc1101 at command”.
- Sơ đồ chân: tương tự như module HC-05.
- Cách lập trình: tương tự như module HC-05.
- Lưu ý:
    - Tất cả các module có chung kênh sóng và địa chỉ kênh cùng đều nhận được dữ liệu khi một module trong mạng truyền.
    - Trên thị trường có hai loại module CC1101 là giao tiếp bằng UART và SPI, trên đây là module giao tiếp bằng UART. Mình khuyến khích các bạn dùng module này vì đơn giản để sử dụng hơn module giao tiếp bằng SPI.

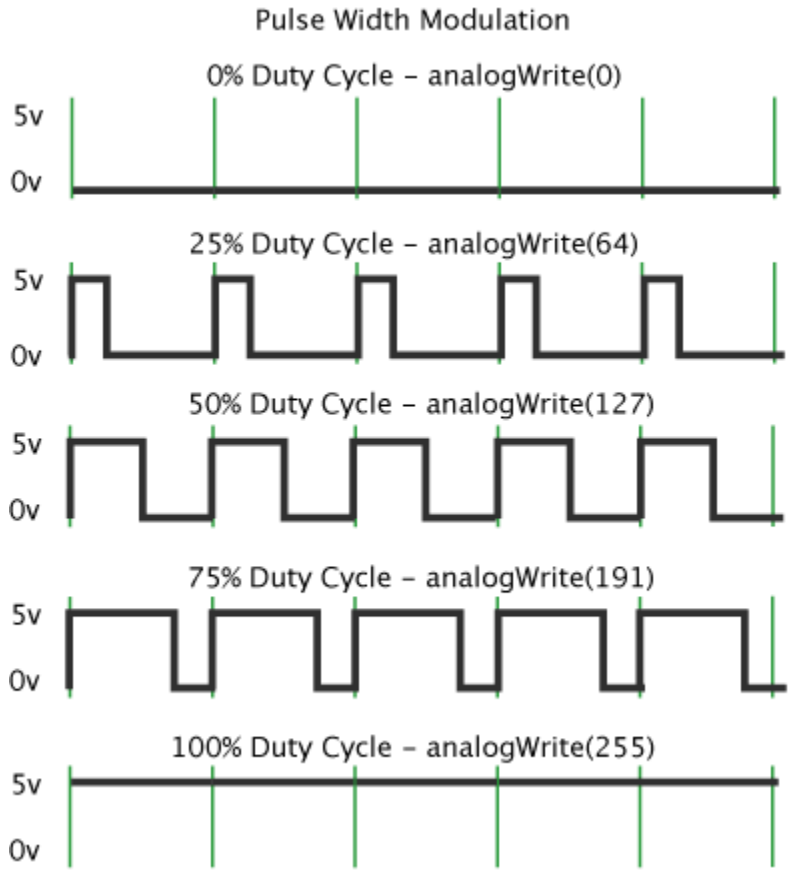
4. Lời kết

- UART (hay giao tiếp nối tiếp) là một chuẩn giao tiếp rất phổ biến và thông dụng khi lập trình vi điều khiển vì tính đơn giản để sử dụng truyền dữ liệu nhỏ.
- Chúng ta nên phân biệt rằng giao tiếp nối có rất nhiều chuẩn và UART là một trong số đó nên cẩn thận khi nói các khái niệm:
  - UART là giao tiếp nối tiếp (đúng).
  - Giao tiếp nối tiếp chỉ là UART (sai).
- Có một chuẩn giao tiếp cũng khá phổ biến trong công nghiệp là RS232, vì mức điện áp hoạt động của 2 chuẩn giao tiếp là khác nhau nên khi vi điều khiển với các module có chuẩn giao tiếp này thì chúng ta cần thêm một module chuyển đổi từ UART sang RS232.
- Lưu ý rằng đối tượng Serial là đối tượng hoạt động với ngoại vi phần cứng bên trong chip (chân 0 và chân 1) và một số board Arduino chỉ có 1 module giao tiếp nối tiếp, nếu ứng dụng của chúng ta muốn hoạt động nhiều module giao tiếp nối tiếp hơn thì có các bạn có thể tham khảo thư viện "**SoftwareSerial**". Khi bạn sử dụng thư viện này thì không có các chức năng liên quan đến phần cứng như interrupt hay event.

Bài 4 **ANALOG**

1. Giới thiệu

- Các tín hiệu ta cảm nhận được từ môi trường đều là tín hiệu analog (tín hiệu tương tự) nhưng vi điều khiển ta chỉ hoạt động với tín hiệu digital (tín hiệu số), vì thế ta muốn vi điều khiển hiểu được tín hiệu tương tự thì phải có module chuyển từ analog sang digital (module ADC – Analog Digital Converter). Ngược lại, ta cũng có thể xuất từ tín hiệu digital sang tín hiệu analog bằng module DAC (Arduino Due có hỗ trợ module này, nếu board Arduino không hỗ trợ thì ta gắn thêm chip rời để chuyển đổi ra giá trị analog thực sự).
- Ngoài hai module ADC và DAC, PWM (Pulse Width Modulation) cũng được Arduino hỗ trợ như một tính năng analog dùng để xuất xung với độ rộng xung thay đổi với chu kỳ xác định trước.



2. Một số hàm thường dùng

a. `analogRead()`

- Chức năng: đọc giá trị analog từ một chân được chỉ định trước. Board Arduino có chứa nhiều kênh analog (mỗi nguồn tín hiệu là một kênh).
    - Tín hiệu trả về có độ phân giải 10-bit, có nghĩa là vi điều khiển đọc điện áp từ 0V – VCC thì giá trị trả về từ 0 đến 1023 (2^{10} - 1). Nếu tín hiệu điện áp nhỏ hơn 0V thì giá trị đọc được vẫn là 0, tương tự nếu điện áp lớn VCC thì giá trị trả về là 1023.
    - Giá trị ngưỡng điện áp tham chiếu có thể thay đổi khi sử dụng hàm analogReference().
    - Các board Arduino như UNO, Nano, Mini, Mega thì thời gian đọc tín hiệu analog ngỏ vào khoảng 100 us.

Board	Operating voltage	Usable pins	Max resolution
Uno	5 Volts	A0 to A5	10 bits
Mini, Nano	5 Volts	A0 to A7	10 bits
Mega, Mega2560, MegaADK	5 Volts	A0 to A14	10 bits
Micro	5 Volts	A0 to A11*	10 bits
Leonardo	5 Volts	A0 to A11*	10 bits
Zero	3.3 Volts	A0 to A5	12 bits**
Due	3.3 Volts	A0 to A11	12 bits**
MKR Family boards	3.3 Volts	A0 to A6	12 bits**

(*) A0 đến A5 sẽ được ghi lên board, A6 đến A11 tương ứng với các chân 4, 6, 8, 9, 10 và 12.

(**) Độ phân giải mặc định là 10 bits. Nếu muốn sử dụng 12 bits thì phải dùng hàm `analogReadResolution()`.

- Cú pháp:

```
analogRead(pin);
```

    - `pin`: tên chân analog muốn đọc.
- Kết quả trả về: giá trị analog đọc được.
- Ví dụ:

```
/* potentiometer wiper (middle terminal) connected to analog
pin 3 outside leads to ground and +5V */
int analogPin = A3;
// variable to store the value read
int val = 0;


void setup() {
  Serial.begin(9600);           //  setup serial
}
void loop() {
  val = analogRead(analogPin);  // read the input pin
  Serial.println(val);          // debug value
}
```

- Lưu ý: nếu chân analog không được kết nối với nguồn tín hiệu nào thì giá trị trả về sẽ bị dao động không biết trước.

b. `analogReference()`

- Chức năng: cấu hình điện áp tham chiếu cho chân analog ngõ vào.
- Cú pháp:

```
analogReference(type);
```

  - `type`: loại tham chiếu được sử dụng.
    - Arduino AVR Boards (Uno, Mega, etc.)
      - `DEFAULT`: điện áp 5V nếu dùng board Arduino 5V hoặc điện áp 3.3V nếu dùng board Arduino 3.3V.
      - `INTERNAL`: điện áp tham chiếu nội, bằng 1.1V trên chip ATmega168 or ATmega328P và 2.56V trên chip ATmega8 (không có sẵn trên board Arduino Mega).
      - `INTERNAL1V1`: điện áp tham chiếu nội 1.1V (chỉ có trên board Arduino Mega).

- `INTERNAL2V56`: điện áp tham chiếu nội 2.56V (chỉ có trên board Arduino Mega).
- `EXTERNAL`: sử dụng điện áp trên chân AREF để tham chiếu (từ 0 đến 5V).

▪ Arduino SAMD Boards (Zero, etc.)

- `AR_DEFAULT`: điện áp tham chiếu 3.3V
- `AR_INTERNAL`: điện áp tham chiếu nội 2.23V
- `AR_INTERNAL1V0`: điện áp tham chiếu nội 1.0V
- `AR_INTERNAL1V65`: điện áp tham chiếu nội 1.65V
- `AR_INTERNAL2V23`: điện áp tham chiếu nội 2.23V
- `AR_EXTERNAL`: sử dụng điện áp trên chân AREF để tham chiếu.

▪ Arduino SAM Boards (Due)

- `AR_DEFAULT`: giá trị mặc định là 3.3V, đây là điện áp tham chiếu duy nhất trên board Arduino Due.

- Kết quả trả về: không.
- Lưu ý:
  - Sau thay đổi điện áp tham chiếu, vài giá trị trả về đầu tiên sẽ không chính xác.
  - Không sử dụng điện áp tham chiếu ngoài (thông qua chân AREF) với mức điện áp nhỏ hơn 0V hay lớn hơn 5V.
  - Nếu sử dụng điện áp tham chiếu ngoài, chúng ta cần thiết lập điện áp tham chiếu là `EXTERNAL` trước khi gọi **`analogRead()`**. Bằng không thì sẽ gây ngắn mạch điện áp tham chiếu nội và điện áp tham chiếu ngoại gây ra "**hư hỏng**" vi điều khiển.

c. `analogWrite()`

- Chức năng: ghi ra dạng sóng PWM đến chân được chỉ định.
    - Sau khi gọi hàm `analogWrite()`, chân được chỉ định sẽ tạo ra sóng PWM với duty cycle được chỉ định mãi tới khi gọi hàm `analogWrite()` lần nữa. Tần số PWM ở hầu hết các chân là 490Hz, riêng chân 5 và 6 trên Uno (hoặc các board tương tự) có tần số xấp xỉ 980Hz.
- Cú pháp:

```
analogWrite(pin, value);
```

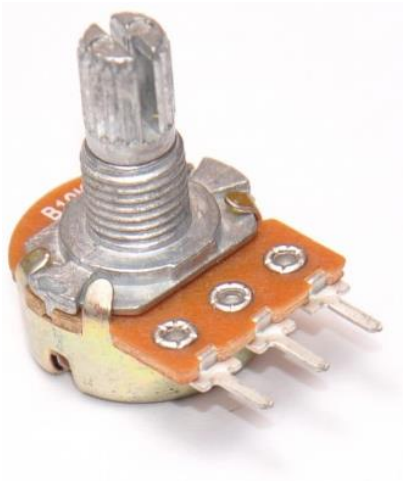
    - `pin`: chân để ghi xung PWM.
    - `value`: duty cycle, giá trị từ 0 đến 255.
- Kết quả trả về: không.
- Ví dụ: thiết lập độ sáng đèn tương ứng với giá trị biến trở.

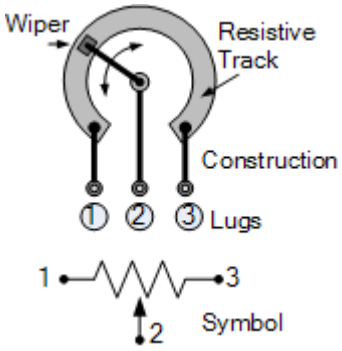
```
// LED connected to digital pin 9
int ledPin = 9;
// potentiometer connected to analog pin 3
int analogPin = 3;
// variable to store the read value
int val = 0;

void setup() {
  pinMode(ledPin, OUTPUT);  // sets the pin as output
}

void loop() {
  val = analogRead(analogPin);  // read the input pin
  /* analogRead values go from 0 to 1023, analogWrite values
from 0 to 255 */
  analogWrite(ledPin, val / 4);
}
```

3. Một số module mẫu

a. Biến trở

- Chức năng: có thể thay đổi được giá trị điện trở.

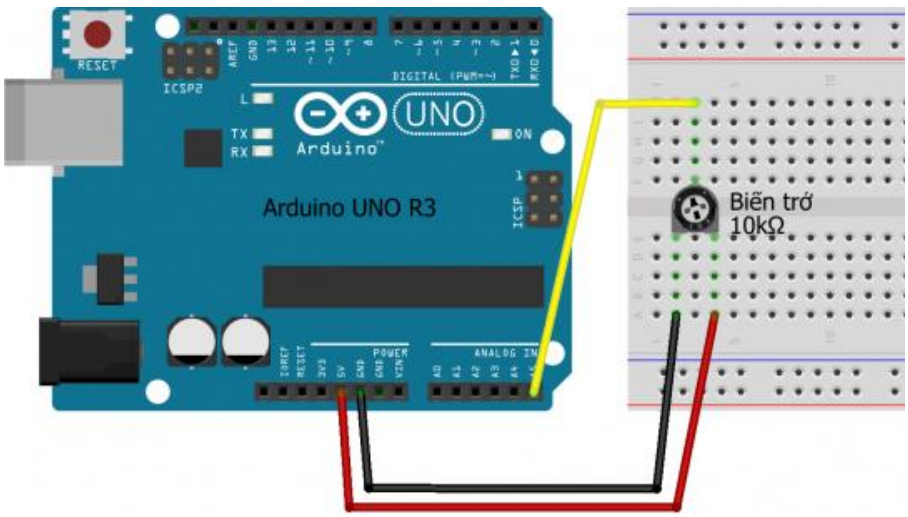




- Sơ đồ chân:
  - Biến trở có 3 chân (như hình trên) với giá trị là R, giá trị điện trở R_{12} và R_{23} sẽ thay đổi giá trị khi xoay núm vặn.

$$R_{12} + R_{23} = R$$

  - Nếu ta nối chân 1 và 3 của biến trở vào nguồn điện 5V, ta có mạch chia áp với giá trị điện áp thay đổi từ 0V đến 5V.

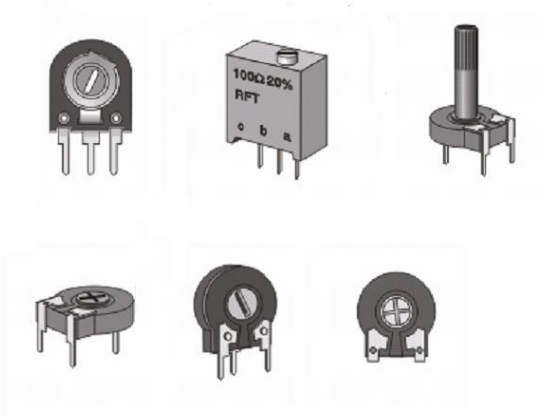


- Cách lập trình:

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

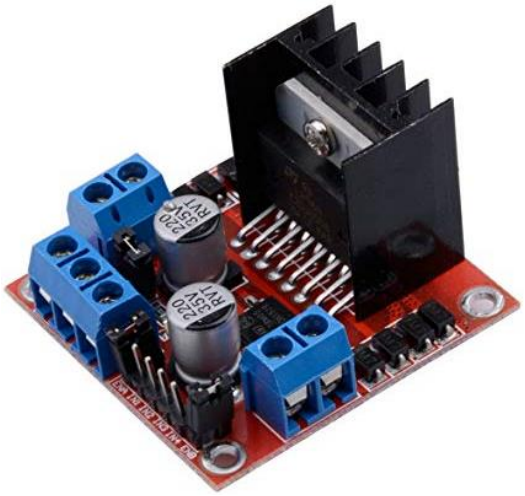
```
void loop() {
  int sensorValue = analogRead(A5);
  Serial.println(sensorValue);
  delay(500);
}
```

- Lưu ý:
  - Biến trở có nhiều hình dạng khác nhau và cách hoạt động khác nhau nhưng mục đích chung vẫn là thay đổi giá trị điện trở.
  - Biến trở có nhiều giá trị điện trở tối đa khác nhau nên tùy vào ứng dụng mà ta chọn giá trị điện trở phù hợp.

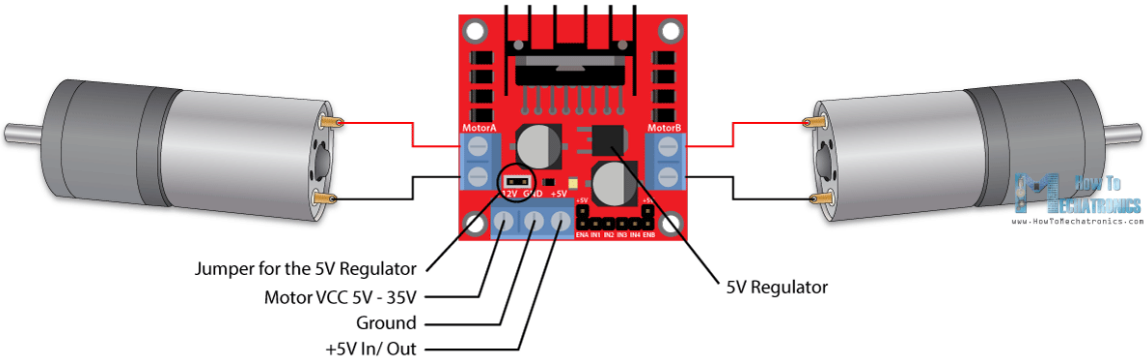


b. Module điều khiển động cơ L298N

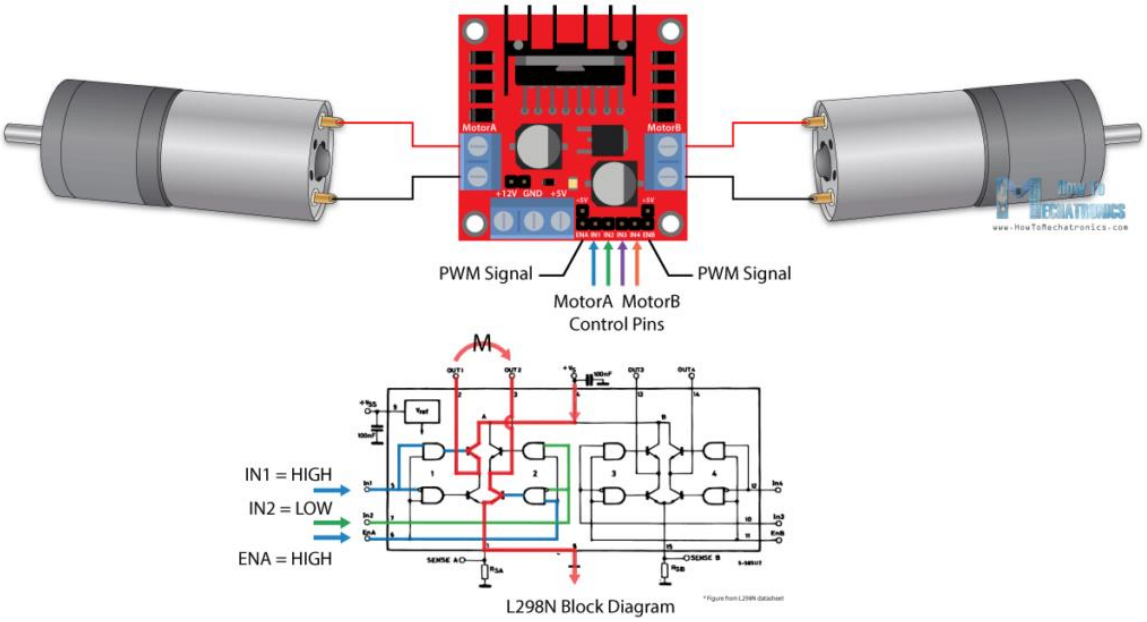
- Chức năng:
  - Để điều khiển tốc độ động cơ (chạy nhanh hay chậm) thì chúng ta cần thay đổi giá trị điện áp cho động cơ và động cơ là một thiết bị tiêu thụ điện với điện áp cao lẫn dòng điện cao nên việc vi điều khiển điều khiển trực tiếp động cơ là không thể. Để hỗ trợ vấn đề này, chúng ta cần một mạch lái (mạch điều khiển – driver) động cơ để nhận tín hiệu điều khiển từ vi điều khiển và cho ngõ ra điện áp tương ứng.
  - Mạch điều khiển động cơ L298N là một module thông dụng điều khiển 2 động cơ DC 12V riêng biệt.



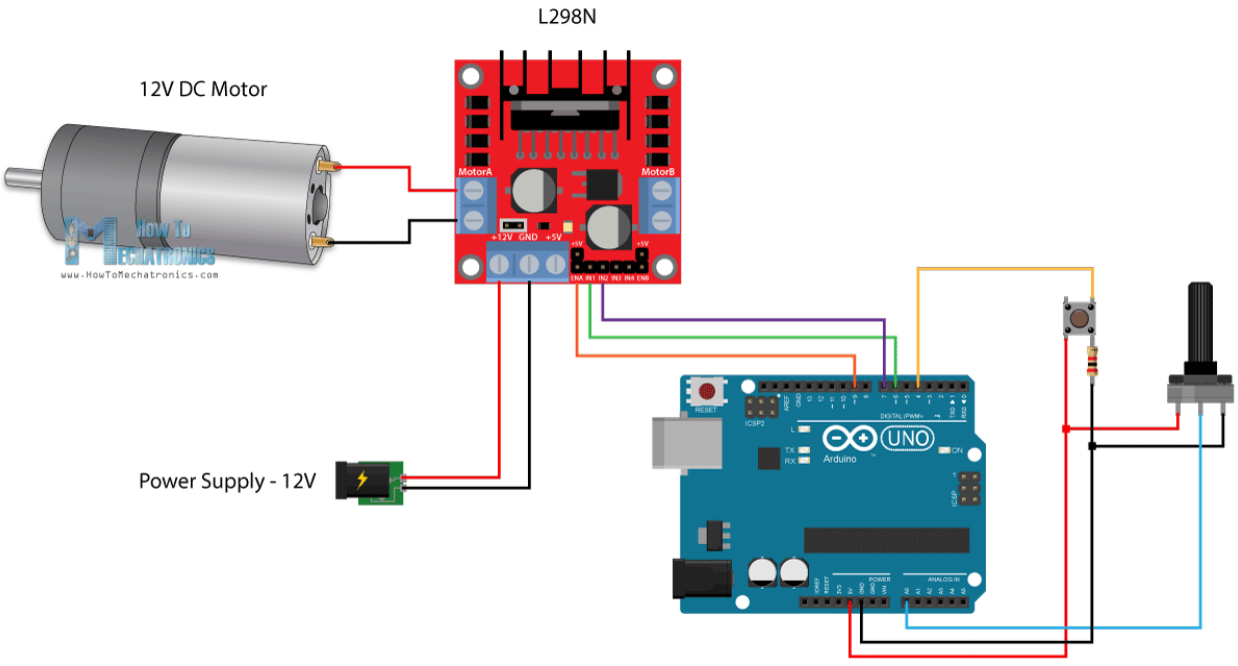
- Sơ đồ chân:
  - Hai domino MotorA và MotorB được nối với động cơ.
  - Chân 12V-GND để cấp nguồn cho động cơ.
  - Chân 5V-GND để cấp nguồn cho mạch điều khiển động cơ. Chúng ta có thể không cấp nguồn 5V thì có thể dùng jumper để lấy điện áp từ nguồn 12V.



  - Các chân ENA, IN1, IN2 để điều khiển MotorA, tương tự các chân ENB, IN3, IN4 để điều khiển MotorB.
  - Các chân INx để điều khiển chiều quay của động cơ, chân ENx để cấp xung thay đổi tốc độ động cơ.



- Cách lập trình:
  - Dùng biến trở để thay đổi tốc độ động cơ:

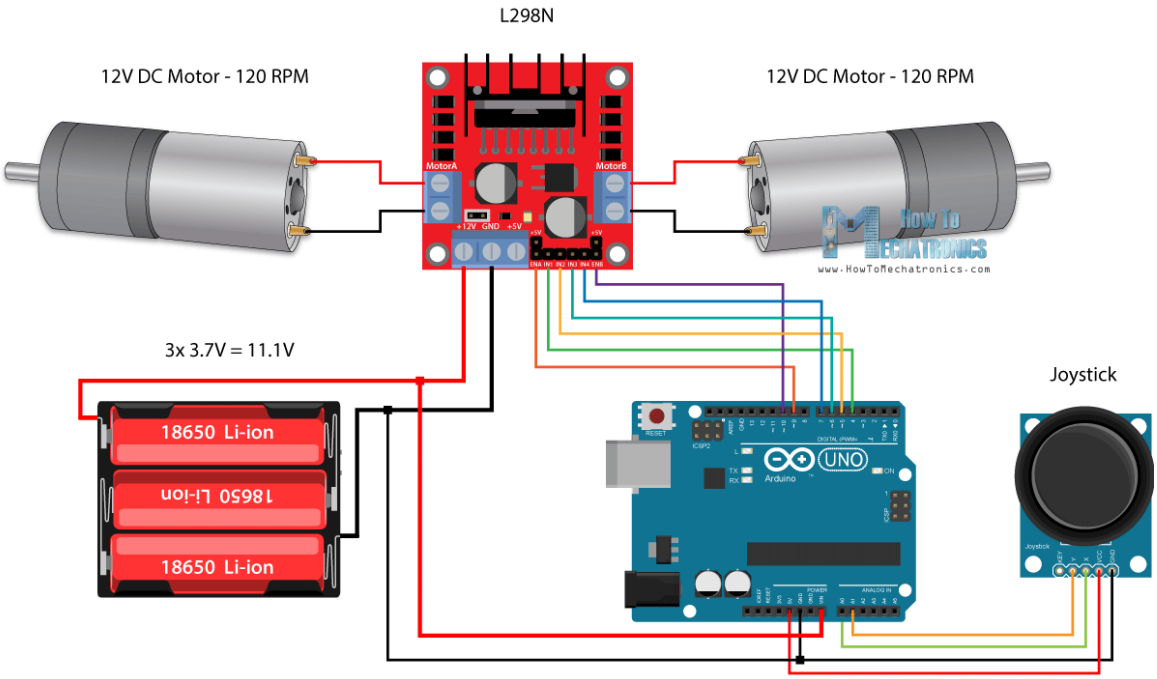



```
/*  Arduino DC Motor Control - PWM | H-Bridge | L298N  - Example 01

    by Dejan Nedelkovski, www.HowToMechatronics.com
*/
```

```
#define enA 9
#define in1 6
#define in2 7
#define button 4

int rotDirection = 0;
int pressed = false;

void setup() {
  pinMode(enA, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(button, INPUT);
  // Set initial rotation direction
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
}

void loop() {
  // Read potentiometer value
  int potValue = analogRead(A0);
  // Map the potentiometer value from 0 to 255
  int pwmOutput = map(potValue, 0, 1023, 0 , 255);
  // Send PWM signal to L298N Enable pin
  analogWrite(enA, pwmOutput);

  // Read button - Debounce
  if (digitalRead(button) == true) {
    pressed = !pressed;
  }
  while (digitalRead(button) == true);
  delay(20);

  // If button is pressed - change rotation direction
```

```
  if (pressed == true  & rotDirection == 0) {
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    rotDirection = 1;
    delay(20);
  }
  // If button is pressed - change rotation direction
  if (pressed == false & rotDirection == 1) {
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    rotDirection = 0;
    delay(20);
  }
}
```

- Dùng joytick để điều khiển xe:




```
/*  Arduino DC Motor Control - PWM | H-Bridge | L298N
        Example 02 - Arduino Robot Car Control
    by Dejan Nedelkovski, www.HowToMechatronics.com
*/
```

```
#define enA 9
#define in1 4
#define in2 5
#define enB 10
#define in3 6
#define in4 7

int motorSpeedA = 0;
int motorSpeedB = 0;

void setup() {
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
}

void loop() {
  int xAxis = analogRead(A0); // Read Joysticks X-axis
  int yAxis = analogRead(A1); // Read Joysticks Y-axis

  // Y-axis used for forward and backward control
  if (yAxis < 470) {
    // Set Motor A backward
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    // Set Motor B backward
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
    // Convert the declining Y-axis readings for going backward from 470 to 0 into 0 to 255 value for the PWM signal for increasing the motor speed
    motorSpeedA = map(yAxis, 470, 0, 0, 255);
```

```
    motorSpeedB = map(yAxis, 470, 0, 0, 255);
  }
  else if (yAxis > 550) {
    // Set Motor A forward
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    // Set Motor B forward
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
    // Convert the increasing Y-axis readings for going forward from 550 to 1023 into 0 to 255 value for the PWM signal for increasing the motor speed
    motorSpeedA = map(yAxis, 550, 1023, 0, 255);
    motorSpeedB = map(yAxis, 550, 1023, 0, 255);
  }
  // If joystick stays in middle the motors are not moving
  else {
    motorSpeedA = 0;
    motorSpeedB = 0;
  }


  // X-axis used for left and right control
  if (xAxis < 470) {
    // Convert the declining X-axis readings from 470 to 0 into increasing 0 to 255 value
    int xMapped = map(xAxis, 470, 0, 0, 255);
    // Move to left - decrease left motor speed, increase right motor speed
    motorSpeedA = motorSpeedA - xMapped;
    motorSpeedB = motorSpeedB + xMapped;
    // Confine the range from 0 to 255
    if (motorSpeedA < 0) {
      motorSpeedA = 0;
    }
    if (motorSpeedB > 255) {
      motorSpeedB = 255;
```

```
    }
  }
  if (xAxis > 550) {
    // Convert the increasing X-axis readings from 550 to 1023
into 0 to 255 value
    int xMapped = map(xAxis, 550, 1023, 0, 255);
    // Move right - decrease right motor speed, increase left
motor speed
    motorSpeedA = motorSpeedA + xMapped;
    motorSpeedB = motorSpeedB - xMapped;
    // Confine the range from 0 to 255
    if (motorSpeedA > 255) {
      motorSpeedA = 255;
    }
    if (motorSpeedB < 0) {
      motorSpeedB = 0;
    }
  }
  // Prevent buzzing at low speeds (Adjust according to your
motors. My motors couldn't start moving if PWM value was below
value of 70)
  if (motorSpeedA < 70) {
    motorSpeedA = 0;
  }
  if (motorSpeedB < 70) {
    motorSpeedB = 0;
  }
  analogWrite(enA, motorSpeedA); // Send PWM signal to motor
A
  analogWrite(enB, motorSpeedB); // Send PWM signal to motor
B
}
```

- Lưu ý:

- Chúng ta nên tách nguồn cho mạch công suất và mạch điều khiển riêng biệt nhau để tránh những ảnh hưởng không mong muốn (không cắm jumper).
- Nếu cắm jumper thì không cấp nguồn 5V.

4. Lời kết

- Tín hiệu analog là tín hiệu rất quan trọng vì nhiều cảm biến sẽ trả về tín hiệu điện áp để xử lý.
- Xung PWM là một tín hiệu điều khiển hiệu quả trong nhiều ứng dụng cần sự thay đổi điện áp dưới sự điều khiển của vi điều khiển.

Bài 5 I2C

1. Giới thiệu

- I2C là một chuẩn giao tiếp 2 dây (SDA và SCL) cho phép kết nối nhiều thiết bị với nhau (lên đến 127 thiết bị) trên cùng một bus (đường truyền). Các chân tín hiệu SDA và SCL phải có điện trở kéo lên nguồn (thường là điện trở 4.7kΩ nối lên nguồn VCC).
- Chuẩn giao tiếp I2C là giao tiếp kiểu chủ/tớ (master/slave). Tất cả các giao tiếp đều do master khởi tạo.
- Các ứng dụng thường dùng khi giao tiếp giữa vi điều khiển và các module thì vi điều khiển luôn đóng vai trò là master.
- Thư viện hỗ trợ giao tiếp I2C là Wire (một số tài liệu sẽ ghi là TWI thay cho I2C). Lưu ý, thư viện này sử dụng bộ đệm 32 bytes nên nếu truyền vượt quá số bytes này thì những bytes vượt quá sẽ bị bỏ qua.
- Các chân hỗ trợ I2C trên một số dòng board Arduino:

Board	I2C / TWI pins
Uno, Ethernet	A4 (SDA), A5 (SCL)
Mega2560	20 (SDA), 21 (SCL)
Leonardo	2 (SDA), 3 (SCL)
Due	20 (SDA), 21 (SCL), SDA1, SCL1

2. Một số hàm thường dùng

a. `Wire.begin()`

- Chức năng: khởi tạo kết nối thiết bị và tham gia vào mạng I2C. Hàm này thường được gọi một lần duy nhất.
- Cú pháp:

```
Wire.begin()
Wire.begin(address)
```

  - `address`: 7-bit địa chỉ của slave (tự chọn), nếu không được khai báo thì tham gia bus với vai trò như master.

- Kết quả trả về: không.

b. `Wire.requestFrom()`

- Chức năng: Yêu cầu dữ liệu slave. Các bytes sau đó có thể được truy xuất với các hàm `available()` và `read()`.

- Cú pháp:

```
Wire.requestFrom(address, quantity);
Wire.requestFrom(address, quantity, stop);
```

  - `address`: 7-bit địa chỉ của thiết bị cần yêu cầu.
  - `quantity`: số bytes yêu cầu.
  - `stop`: true thì sẽ gửi tin nhắn ngừng (stop message) sau yêu cầu và giải phóng bus. Ngược lại nếu false thì sẽ tiếp tục gửi tin nhắn khởi động lại (restart message) sau yêu cầu, trường hợp này thì vẫn giữ bus ở trạng thái tích cực.

- Kết quả trả về: số bytes trả về từ slave.

c. `Wire.beginTransmission()`

- Chức năng: bắt đầu quá trình truyềnn dữ liệu đến địa chỉ I2C được chỉ định. Sau đó, dùng hàm `write()` để ghi các bytes cần truyền và bắt đầu truyền bằng hàm `endTransmission()`.

- Cú pháp:

```
Wire.beginTransmission(address);
```

  - `address`: 7-bit địa chỉ của thiết bị cần truyền tới.

- Kết quả trả về: không.

d. `Wire.endTransmission()`

- Chức năng: kết thúc quá trình truyền tải đến địa chỉ thiết bị được cho trước.

- Cú pháp:

```
Wire.endTransmission();
Wire.endTransmission(stop);
```

- `stop`: true thì sẽ gửi tin nhắn ngừng (stop message) sau quá trình truyền và giải phóng bus. Ngược lại nếu false thì sẽ tiếp tục gửi tin nhắn khởi động lại (restart message) sau quá trình truyền, trường hợp này thì vẫn giữ bus ở trạng thái tích cực.

- Kết quả trả về:
  - `0`: truyền thành công.
  - `1`: dữ liệu quá dài so với bộ đệm.
  - `2`: nhận NACK khi truyền địa chỉ
  - `3`: nhận NACK khi truyền dữ liệu
  - `4`: lỗi khác.

e. `Wire.write()`

- Chức năng: ghi dữ liệu từ slave đến master hoặc từ master tới slave.
- Cú pháp:

```
Wire.write(value) ;
Wire.write(string) ;
Wire.write(data, length);
```

- `value`: giá trị của 1 byte cần gửi.
- `string`: chuỗi cần gửi.
- `data`: mảng các dữ liệu.
- `length`: số byte cần gửi.

- Kết quả trả về: số bytes đã gửi.
- Ví dụ:

```
#include <Wire.h>

byte val = 0;

void setup()
{
  Wire.begin(); // join i2c bus as a master
}

void loop()
{
  Wire.beginTransmission(44); // transmit to device #44
(0x2c)
                              // device address is specified
in datasheet
  Wire.write(val);             // sends value byte
  Wire.endTransmission();     // stop transmitting

  val++;        // increment value
  if(val == 64) // if reached 64th position (max)
  {
    val = 0;    // start over from lowest value
  }
  delay(500);
}
```

f. `Wire.available()`

- Chức năng: trả về số bytes để có thể truy xuất bằng hàm `read()`. Hàm này được gọi ở master sau khi gọi hàm `requestFrom()` hoặc ở slave trong chương trình xử lý `onReceive()`.
- Cú pháp:

```
Wire.available();
```

- Kết quả trả về: số bytes có sẵn để đọc.

g. `Wire.read()`

- Chức năng: đọc một byte đã được truyền từ slave đến master sau khi gọi `requestFrom()` hoặc được truyền từ master đến slave.
- Cú pháp:

```
Wire.read();
```

- Kết quả trả về: byte tiếp theo nhận được.
- Ví dụ:

```
#include <Wire.h>

void setup()
{
  Wire.begin();        // join i2c bus (address optional for
master)
  Serial.begin(9600);  // start serial for output
}

void loop()
{
  Wire.requestFrom(2, 6);    // request 6 bytes from slave
device #2

  while(Wire.available())    // slave may send less than
requested
  {
    char c = Wire.read();    // receive a byte as character
    Serial.print(c);         // print the character
  }

  delay(500);
}
```

h. `Wire.setClock()`

- Chức năng: điều chỉnh tần số clock cho giao tiếp I2C.
- Cú pháp:

```
Wire.setClock(clockFrequency);
```

  - `clockFrequency`: giá trị của tần số giao tiếp (đơn vị là Hz). Chú ý nên xem giới hạn của tất cả phần cứng trong mạng để chọn tần số hợp lý.

- Kết quả trả về: không.

i. `Wire.onReceive()`

- Chức năng: đăng ký tên một chương trình con để khi có sự kiện nhận dữ liệu thì chương trình con này được gọi.
- Cú pháp:

```
Wire.onReceive(handler);
```

  - `handler`: tên chương trình con được gọi. Hàm này nên có dạng như sau

```
void TWIReceiveHandler(int numOfBytes) {
...
}
```

- Kết quả trả về: không.

j. `Wire.onRequest()`

- Chức năng: đăng ký một chương trình con để khi có một yêu cầu từ master thì hàm này sẽ được gọi.
- Cú pháp:

```
Wire.onReceive(handler);
```

  - `handler`: tên chương trình con được gọi. Hàm này nên có dạng như sau

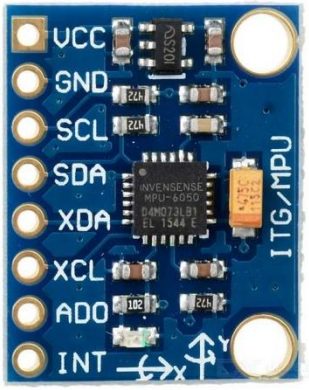
```
void TWIReceiveHandler(void) {
...
}
```

- Kết quả trả về: không

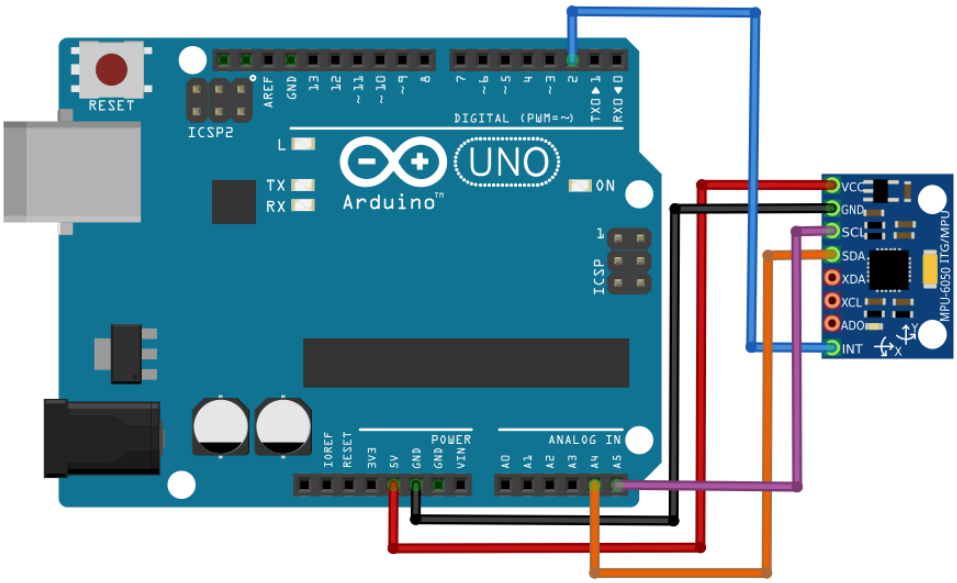
3. Một số module

a. Module IMU MPU6050

- Chức năng:
  - Là một cảm biến trả về các giá trị gia tốc (accelerometer – hay gọi là accel) và con quay hồi chuyển (gyroscope – hay gọi là gyro).
  - IMU thường dùng trong các ứng dụng theo dõi chuyển động.



- Sơ đồ chân:
  - Hai chân VCC-GND được nối vào nguồn 5V.
  - Hai chân SDA, SCL lần lượt mắc vào các chân A4, A5.
  - Có thể không cần đến chân INT.



- Cách lập trình:
  - Đọc về giá trị thô chưa qua xử lý:

```
// MPU-6050 Short Example Sketch
// By Arduino User JohnChi
// August 17, 2014
// Public Domain
#include<Wire.h>
```

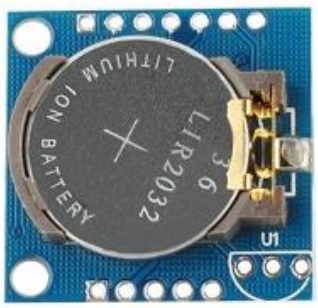
```
    const int MPU_addr=0x68;  // I2C address of the MPU-6050
    int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
    void setup(){
      Wire.begin();
      Wire.beginTransmission(MPU_addr);
      Wire.write(0x6B);  // PWR_MGMT_1 register
      Wire.write(0);     // set to zero (wakes up the MPU-
6050)
      Wire.endTransmission(true);
      Serial.begin(9600);
    }
    void loop(){
      Wire.beginTransmission(MPU_addr);
      Wire.write(0x3B);  // starting with register 0x3B
(ACCEL_XOUT_H)
      Wire.endTransmission(false);
      Wire.requestFrom(MPU_addr,14,true);  // request a
total of 14 registers
      AcX=Wire.read()<<8|Wire.read();  // 0x3B
(ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
      AcY=Wire.read()<<8|Wire.read();  // 0x3D
(ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
      AcZ=Wire.read()<<8|Wire.read();  // 0x3F
(ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
      Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H)
& 0x42 (TEMP_OUT_L)
      GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H)
& 0x44 (GYRO_XOUT_L)
      GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H)
& 0x46 (GYRO_YOUT_L)
      GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H)
& 0x48 (GYRO_ZOUT_L)
      Serial.print("AcX = "); Serial.print(AcX);
      Serial.print(" | AcY = "); Serial.print(AcY);
      Serial.print(" | AcZ = "); Serial.print(AcZ);
      Serial.print(" | Tmp = ");
Serial.print(Tmp/340.00+36.53);  //equation for temperature
in degrees C from datasheet
      Serial.print(" | GyX = "); Serial.print(GyX);
```

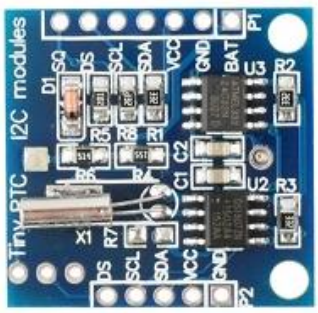
```
    Serial.print(" | GyY = "); Serial.print(GyY);
    Serial.print(" | GyZ = "); Serial.println(GyZ);
    delay(333);
  }
```

- Lưu ý:
  - Code mẫu trên rất đơn giản và chưa được xử lý tốt, bạn nên tìm thêm các nguồn tài liệu khác với từ khóa "MPU6050 Arduino" để có những nguồn tài liệu đầy đủ hơn.
  - Đây là một cảm biến khá khó để xử lý vì có cần phải nhiều thuật khác kết hợp thì mới ra kết quả tốt hơn. Nếu bạn có làm đề tài liên quan thì có thể tìm hiểu bộ lọc Kalman (Kalman filter) và bộ lọc bù (complementary filter) để ra kết quả tốt nhất.

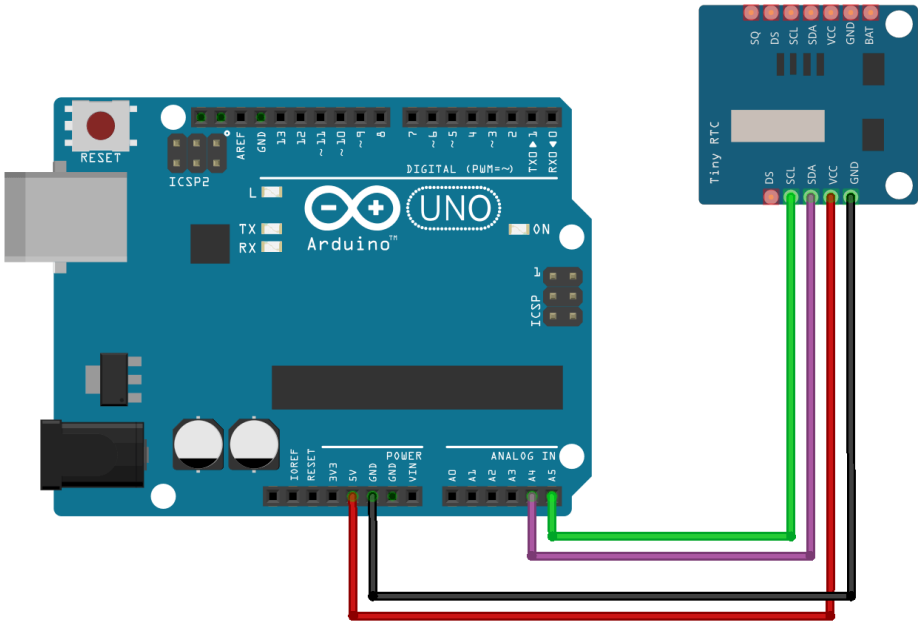
b. Module thời gian thực RTC

- Chức năng:
  - Lưu trữ thời gian thực tế ngay cả khi vi điều khiển không hoạt động.
  - Thường được dùng trong các ứng dụng điều khiển thiết bị theo thời gian hoặc một số ứng dụng cần biết thời gian thực.





- Sơ đồ chân:
  - Hai chân VCC-GND nối với nguồn 5V.
  - Hai chân SDA, SCL lần lượt mắc vào hai chân A4, A5.



- Cách lập trình:

```
#include <Wire.h>
// Address of DS1307
const byte DS1307 = 0x68;
// Number of bytes that need to read from DS1307
const byte NumberOfFields = 7;
  // Time variable
int second, minute, hour, day, wday, month, year;


void setup()
{
  Wire.begin();
  // Setup time: 07:00:10 Sat 27/04/2019
  setTime(07, 00, 10, 7, 27, 4, 19);
  Serial.begin(9600);
}


void loop()
{
  // Read data from DS1307
  readDS1307();
```

```
      // Display time to Serial monitor
      digitalClockDisplay();
      delay(1000);
    }

    void readDS1307()
    {
            Wire.beginTransmission(DS1307);
            Wire.write((byte)0x00);
            Wire.endTransmission();
            Wire.requestFrom(DS1307, NumberOfFields);

            second = bcd2dec(Wire.read() & 0x7f);
            minute = bcd2dec(Wire.read() );
            hour   = bcd2dec(Wire.read() & 0x3f); // 24h
mode
            wday   = bcd2dec(Wire.read() );
            day    = bcd2dec(Wire.read() );
            month  = bcd2dec(Wire.read() );
            year   = bcd2dec(Wire.read() );
            year += 2000;
    }
    // Convert from BCD (Binary-Coded Decimal) to Decimal
    int bcd2dec(byte num)
    {
            return ((num/16 * 10) + (num % 16));
    }
    // Convert from Decimal to BCD
    int dec2bcd(byte num)
    {
            return ((num/10 * 16) + (num % 10));
    }

    void digitalClockDisplay(){
```

```
        // digital clock display of the time
        Serial.print(hour);
        printDigits(minute);
        printDigits(second);
        Serial.print(" ");
        Serial.print(day);
        Serial.print(" ");
        Serial.print(month);
        Serial.print(" ");
        Serial.print(year);
        Serial.println();
    }

    void printDigits(int digits)
  {
        Serial.print(":");
        if(digits < 10)
       {
            Serial.print('0');
        }
          Serial.print(digits);
    }

    /* Set up time for DS1307 */
    void setTime(byte hr, byte min, byte sec, byte wd, byte
d, byte mth, byte yr)
    {
            Wire.beginTransmission(DS1307);
            Wire.write(byte(0x00)); // Reset pointer
            Wire.write(dec2bcd(sec));
            Wire.write(dec2bcd(min));
            Wire.write(dec2bcd(hr));
            Wire.write(dec2bcd(wd)); // day of week: Sunday
= 1, Saturday = 7
            Wire.write(dec2bcd(d));
```

```
            Wire.write(dec2bcd(mth));
            Wire.write(dec2bcd(yr));
            Wire.endTransmission();
    }
```

4. Lời kết

- Một trong những ưu điểm của giao tiếp I2C là có thể giao tiếp rất nhiều thiết bị trên cùng một bus vật lý. Nhưng mỗi thời điểm trên bus chỉ tồn tại duy nhất một kết nối giữa một master và một slave.

Bài 6 **NGẮT - INTERRUPT**

1. Giới thiệu

- Ngắt (interrupt) là một tính năng rất quan trọng của vi điều khiển. Nó có thể can thiệp vào bất cứ thời điểm nào của vi điều khiển khi có một sự kiện ngắt xảy ra.
- Interrupts cho phép những công việc quan trọng được thực hiện ngầm (mặc định là được cho phép). Một số hàm sẽ không hoạt động khi tắt interrupt và các dữ liệu truyền đến vi điều khiển có thể bị bỏ qua

2. Một số hàm thường dùng

a. `interrupts()`

- Chức năng: cho phép ngắt hoạt động khi chúng ta đã tắt interrupt.
- Cú pháp:

```
interrupts();
```

- Kết quả trả về: không.
- Ví dụ:

```
void setup() {}
void loop() {
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```

b. `noInterrupts()`

- Chức năng: tắt ngắt. Chúng ta tắt ngắt trong trường hợp có những công việc nghiêm khắc về thời gian vì ngắt có thể ảnh hưởng một phần nhỏ đến thời gian thực hiện các công việc khác. Lưu ý khi tắt ngắt thì một số hàm sẽ không hoạt động.

- Cú pháp:

```
noInterrupts();
```

- Kết quả trả về: không.

c. `attachInterrupt()`

- Chức năng:
  - Đăng ký chương trình con mà chương trình này được gọi khi có một sự thay đổi từ một chân digital bên ngoài.
  - Các chân hỗ trợ interrupt:

Board	Digital Pins Usable For Interrupts
Uno, Nano, Mini, other 328-based	2, 3
Uno WiFi Rev.2	all digital pins
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7
Zero	all digital pins, except 4
MKR Family boards	0, 1, 4, 5, 6, 7, 8, 9, A1, A2
Due	all digital pins
101	all digital pins (Only pins 2, 5, 7, 8, 10, 11, 12, 13 work with **CHANGE**)

- Cú pháp:

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
```

  - `pin`: chân sẽ xét điều kiện để vào chương trình con.
  - `ISR`: hàm được gọi khi có ngắt xảy ra.
  - `mode`: định nghĩa các chế độ để gọi vào ngắt.
    - `LOW`: xảy ra ngắt khi chân có mức điện áp.
    - `CHANGE`: xảy ra ngắt khi tại chân vi điều khiển có sự thay đổi giá trị logic.
    - `RISING`: xảy ra khi có sự chuyển mức điện áp từ thấp lên cao.
    - `FALLING`: xảy ra khi có sự thay đổi mức điện áp từ cao xuống thấp.

- Kết quả trả về: không.

- Ví dụ:

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin),
blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

- Lưu ý:

  - Khi gọi hàm ngắt thì hàm `delay()` sẽ không hoạt động, giá trị trả về `millis()` sẽ không thay đổi, dữ liệu nhận về từ các giao tiếp có thể bị mất. Vì vậy, ta dùng hàm này khi thật sự cần thiết và cực kỳ quan trọng.

  - Nên khai báo các biến dạng `volatile` trong chương trình ngắt này.

  - Phải lập trình hàm được đính kèm càng ngắn càng tốt và thời gian thực hiện phải ngắn

  d. `detachInterrupt()`

- Chức năng: tắt ngắt được đính kèm.

- Cú pháp:

```
detachInterrupt(digitalPinToInterrupt(pin));
```

- `pin`: chân được đính kèm interrupt cần tắt.

- Kết quả trả về: không.

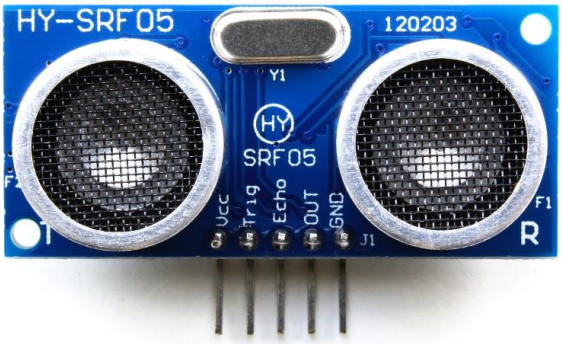
3. Lời kết

- Interrupt là phần khá quan trọng nhưng cũng rất thận trọng khi sử dụng. Nếu bạn cảm thấy không tự tin thì có thể tìm cách giải quyết khác không cần dùng interrupt.

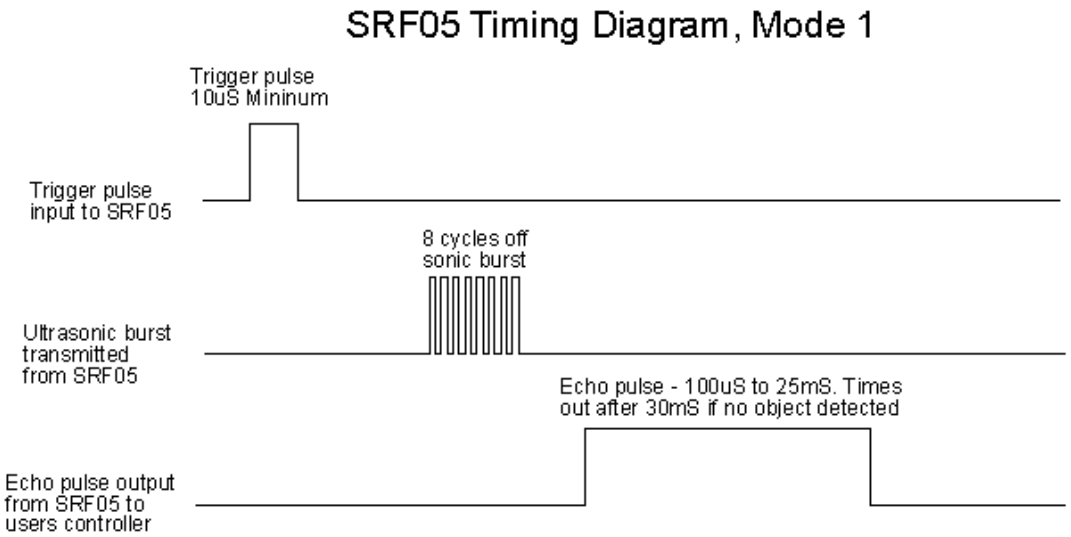
Bài 7 MỘT SỐ MODULE THÔNG DỤNG KHÁC

1. Module đo khoảng cách SRF-05

- Chức năng: đo khoảng cách dùng sóng siêu âm.

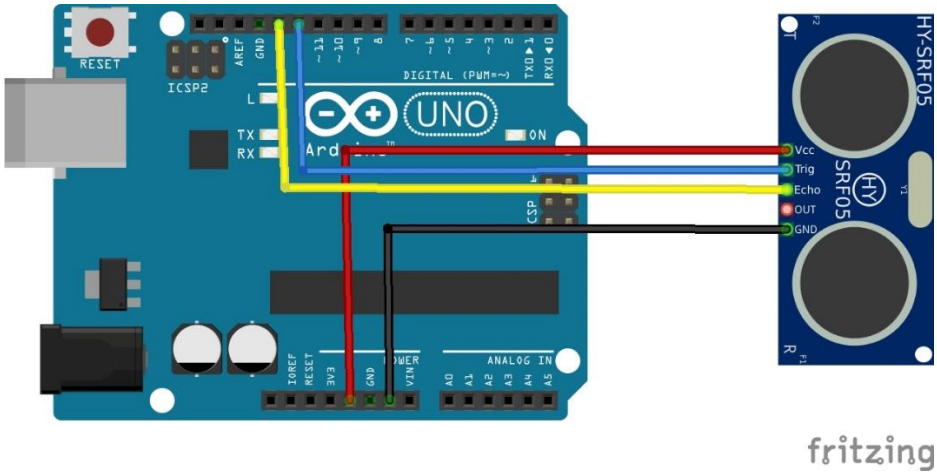


- Nguyên lý hoạt động: ta kích chân Trig một xung lớn hơn 10us thì module sẽ phát ra môi trường 8 xung sóng siêu âm. Sau đó chân ECHO sẽ chuyển trạng thái từ thấp lên cao, chân ECHO xuống thấp lại khi có xung phản xạ lại (gặp vật cản thì xung sẽ phản xạ lại) hoặc sau 30ms nếu không có vật cản. Thời gian của xung ECHO ở mức cao là thời gian để sóng âm đi từ module đến vật cản cộng với thời gian sóng phản xạ từ vật cản về module.



- Sơ đồ chân:
  - Hai chân VCC-GND được nối vào nguồn 5V.

- Chân Trig nối với chân 12, chân Echo nối vào chân 13.



- Chương trình mẫu:

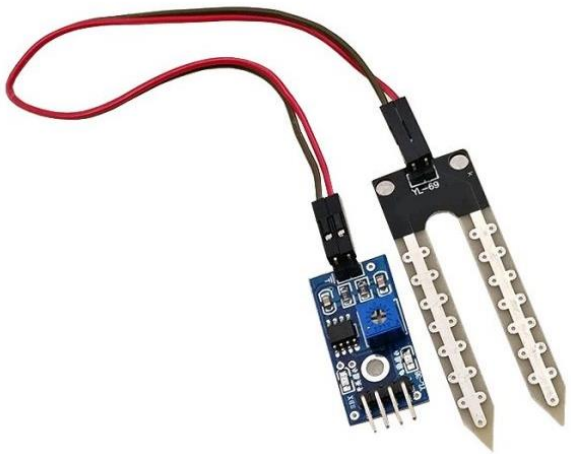
```
/*
VCC to +5V
GND to ground
TRIG to digital pin 12
ECHO to digital pin 13
*/

const int TRIG_PIN = 12;
const int ECHO_PIN = 13;

void setup()
{
  // initialize serial communication:
  Serial.begin(9600);
  pinMode(TRIG_PIN,OUTPUT);
  pinMode(ECHO_PIN,INPUT);
}

void loop()
{
  long duration, distanceCm, distanceIn;
```

```
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  duration = pulseIn(ECHO_PIN,HIGH);

  // convert the time into a distance
  distanceCm = duration / 29.1 / 2 ;
  distanceIn = duration / 74 / 2;

  if (distanceCm <= 0)
  {
    Serial.println("Out of range");
  }
  else
  {
    Serial.print(distanceIn);
    Serial.print("in: ");
    Serial.print(distanceCm);
    Serial.print("cm");
    Serial.println();
  }
  delay(1000);
}
```

- Lưu ý: kết quả đo khoảng cách phụ thuộc rất nhiều vào bề mặt vật thể.

2. Cảm biến độ ẩm đất

- Chức năng: đo độ ẩm đất nhưng không đưa ra giá trị cụ thể. Ta cần phải dùng phương pháp thực nghiệm để điều chỉnh cũng như lấy thông số cho việc lập trình cảm biến.



- Sơ đồ chân:
  - Hai chân VCC-GND nối với nguồn 5V.
  - Chân D0 cho kết quả trả về dạng digital (HIGH/LOW). Trên module có biến trở điều chỉnh giá trị so sánh.
  - Chân A0 trả về kết quả dạng analog (dao động từ 0V - VCC).

- Chương trình mẫu:

```
int sensor_pin = A0;
int output_value ;


void setup()
```

```
{
  Serial.begin(9600);
  Serial.println("Reading From the Sensor ...");
  delay(2000);
}

void loop()
{
  output_value= analogRead(sensor_pin);
  output_value = map(output_value,550,0,0,100);
  Serial.print("Mositure : ");
  Serial.print(output_value);
  Serial.println("%");
  delay(1000);
}
```

Bài 8 MỘT SỐ HÀM KHÁC

1. Hàm I/O nâng cao

- `pulseIn(pin, value[, timeout])`: trả về độ dài của xung ở mức `HIGH` hoặc `LOW` (`value`) tại chân `pin`. Giá trị trả về là 0 nếu không có xung mức `value` trước thời gian `timeout`.
- `shiftIn(dataPin, clockPin, bitOrder)`: dịch chuyển từng bit khi có sự thay đổi tại chân `clockPin` thành một byte tại chân `dataPin`, thứ tự của các bit trong một byte được sắp xếp theo `bitOrder` (**`MSBFIRST/LSBFIRST`**).
- `shiftOut(dataPin, clockPin, bitOrder, value)`: xuất giá trị từng bit ra chân `dataPin` khi có sự thay đổi tại chân `clockPin` theo thứ `bitOrder` (`MSBFIRST/LSBFIRST`) với giá trị `value`.

2. Hàm liên quan đến toán

- `abs(x)`: trả về trị tuyệt đối của `x`.
- `constrain(x, a, b)`: trả về giá trị của `x` bị chặn bởi cân dưới `a` và cận trên `b`.
- `map(value, fromLow, fromHigh, toLow, toHigh)`: chuyển đổi giá trị value từ phạm vi giá trị này (`fromLow - fromHigh`) thành phạm vi giá trị khác (`toLow - toHigh`).
- `max(x, y)`: trả về lớn hơn trong 2 số `x` và `y`.
- `min(x, y)`: trả về nhỏ hơn trong 2 số `x` và `y`.
- `pow(base, exponent)`: tính lũy thừa `base` mũ `exponent`.
- `sq(x)`: bình phương `x`.
- `sqrt(x)`: căn bậc 2 của `x`.
- `sin(rad)`: sin của `rad`.

- `cos()`: cos của `rad`.
- `tan()`: tan của `rad`.

3. Hàm làm việc với chuỗi

- `char *strcpy(char *dich, char *nguon)`: sao chép chuỗi nguồn vào chuỗi đích.
- `char *strncpy(char *dich, char *nguon, int n)`: sao chép `n` ký tự đầu tiên của chuỗi nguồn sang chuỗi đích.
- `int strlen(char *s)`: trả về độ dài của chuỗi `s`.
- `char *strcat(char *dich,char *nguon)`: ghép chuỗi nguồn vào sau chuỗi đích.
- `char *strncat(char *dich,char *nguon,int n)`: ghép `n` kí tự đầu tiên của chuỗi nguồn vào chuỗi đích.
- `int strcmp(char *s1,char *s2)`: so sánh hai chuỗi `s1` và `s2`.
- `char *strlwr(char *s)`: chuyển tất cả các ký tự về chữ thường.
- `char *strupr(char *s)`: chuyển tất cả các ký tự về chữ hoa.
- `void *memset (void *ptr, int value, size_t num)`: ghi `num` bytes của bộ nhớ bắt đầu từ địa chỉ ptr thành giá trị `value`.
- `void *memcpy (void *destination, const void *source, size_t num)`: sao chép `num` bytes từ địa chỉ `source` đến địa chỉ `destination`.

LỜI KẾT

Chúng ta đã đi qua hầu hết các tính năng căn bản của Arduino và đã thực hiện một số ví dụ đơn giản, nhóm tác giả hy vọng các bạn có thể nắm được các kiến thức của Arduino để bạn có thể tạo ra những sản phẩm như mong muốn.

Trong quyển sách này, tác giả không hướng dẫn bạn đọc làm bất kỳ một dự án nào hoàn chỉnh mà chỉ cung cấp cho bạn những kiến thức riêng lẻ vì số lượng dự án thì nhiều vô số kể nên các tác giả không thể hướng dẫn chi tiết từng dự án một. Vì thế, nội dung sách chỉ cho bạn những kiến thức cốt lõi nhất để bạn có thể tự phát triển ứng dụng bạn mong muốn.

Cho đến thời điểm hiện tại, Arduino được xem là một nền tảng vi điều khiển dân dụng phổ biến nhất trên thế giới nên số tượng tài liệu trên mạng nhiều vô số kể và lượng kiến thức trong quyển sách này chỉ là một phần rất nhỏ nhoi trong lượng kiến thức ấy. Vì thế, cá nhân các bạn cần phải **chủ động** tìm hiểu thêm để có những kiến thức rộng hơn và hiểu được bản chất của tư duy vi điều khiển.

Nếu bạn đọc có bất cứ góp ý/thắc mắc liên quan đến nội dung quyển sách thì đừng ngần ngại gửi mail về nhóm tác giả theo thông tin tác giả bên dưới.

Quyển sách được tổng hợp từ rất nhiều tài liệu tham khảo trên mạng nên đây chỉ được xem là quyển sách ghi chú lại các thông tin cần thiết về Arduino. Quyển sách này chỉ mang tính chất tham khảo và không có bản quyền tác giả (các vấn đề bản quyền mang tính chất pháp lý không được các tác giả biên soạn chịu trách nhiệm).

THÔNG TIN TÁC GIẢ

Tác giả 1

Họ và tên: Phan Minh Trí

Email: minhtri1996vl@gmail.com

Chuyên ngành học tập: chuyên ngành điều khiển và tự động hóa, khoa Điện – Điện tử trường đại học Bách Khoa thành phố Hồ Chí Minh.

Lĩnh vực nghiên cứu: kỹ sư phần mềm nhúng (embedded software engineeing), lập trình vi điều khiển (embedded engineeing), rô bốt (robotic).

Tác giả 2

Họ và tên: Lâm Phước An

Email: 1551002@hcmut.edu.vn

Chuyên ngành học tập: chuyên ngành điện tử, khoa Điện – Điện tử trường đại học Bách Khoa thành phố Hồ Chí Minh.

Lĩnh vực nghiên cứu: kỹ sư phần mềm nhúng (embedded software engineeing), lập trình vi điều khiển (embedded engineeing).

ÁP DỤNG GIẢNG DẠY

Xin cảm ơn ***câu lạc bộ GenS-P*** (trường Trung học Phổ thông chuyên Nguyễn Bỉnh Khiêm, Vĩnh Long) đã dùng tài liệu này như một tài liệu tham khảo cho chương trình giảng dạy Arduino tại câu lạc bộ.