Phát triển ứng dụng IoT

(IoT APPLICATION DEVELOPING)

Chương 2: Cơ bản về lập trình Arduino

ThS. Lê Đức Quang

Bộ môn : Kỹ thuật hệ thống và mạng máy tính

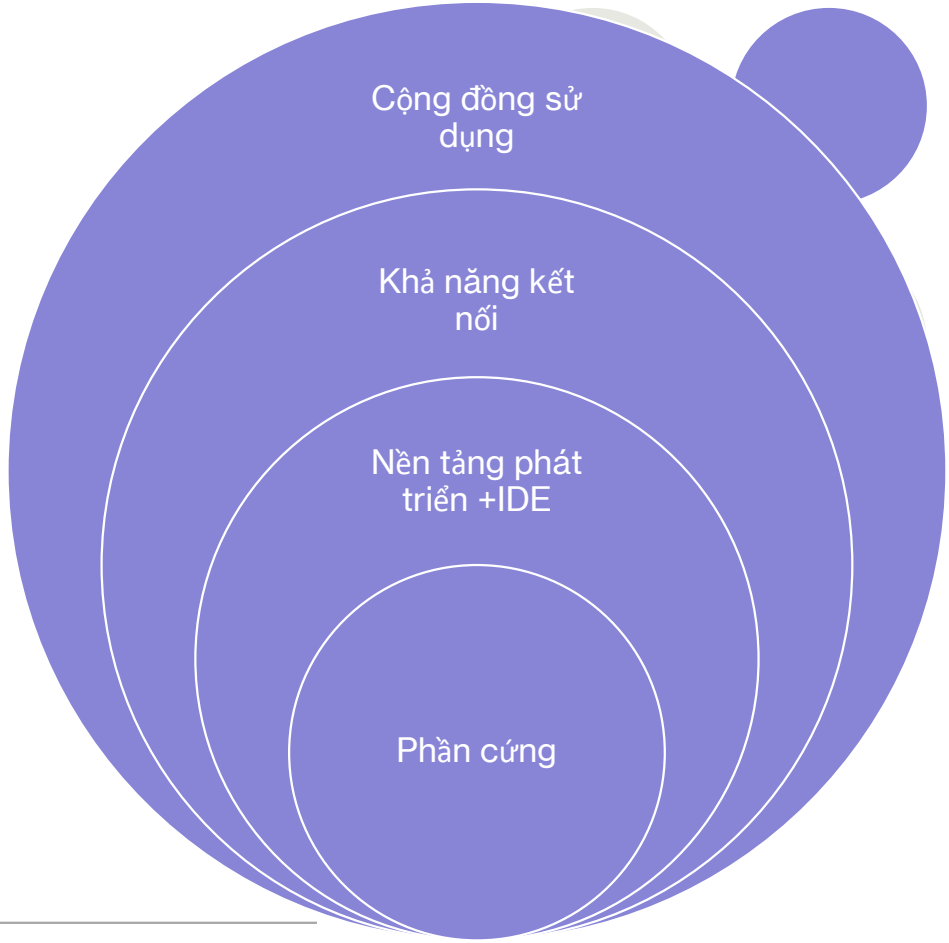
Khoa Công nghệ thông tin

Nội dung

- Giới thiệu về Arduino
- Lập trình cơ bản với Arduino

2.1 Giới thiệu về Arduino

- A) Arduino là gì?
  - Arduino là một nền tảng phần cứng và phần mềm mã nguồn mở được thiết kế để giúp đơn giản hóa việc phát triển các dự án điện tử.



2.1 Giới thiệu về Arduino

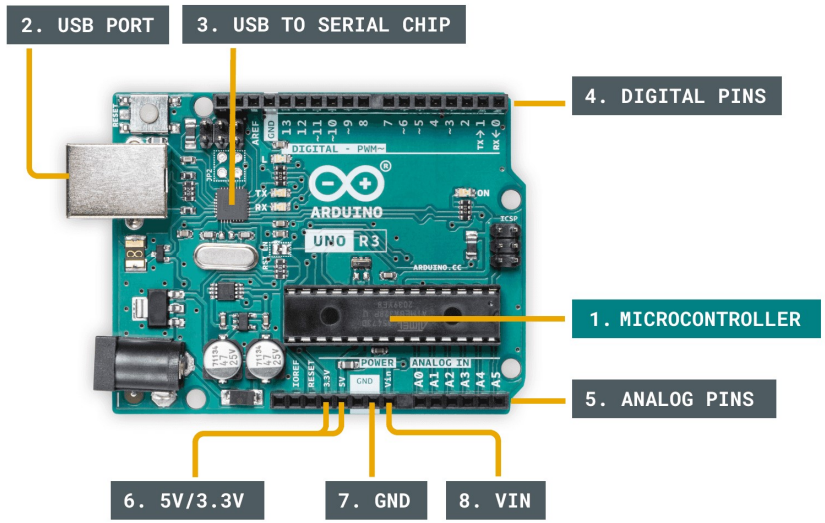
Tại sao lại chọn Arduino?

- Dễ sử dụng và phổ biến
  - Nhỏ gọn, tiêu thụ ít năng lượng
  - Kiến trúc tương tự như máy tính
  - Giá rẻ
- Môi trường lập trình thân thiện với người dùng
  - Ngôn ngữ bậc cao C
  - Môi trường phát triển tích hợp hoàn chỉnh
  - Việc điều khiển được đơn giản hóa
- Cộng đồng phát triển rộng lớn

2.1 Giới thiệu về Arduino

B) Các thành phần của Arduino

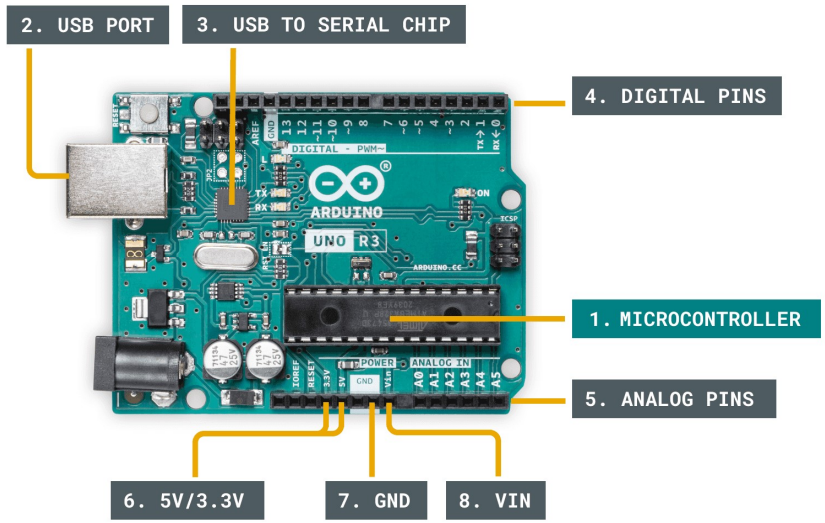
- Vi mạch (Microcontrollers – IC): bộ não của Arduino, trung tâm xử lý hoạt động của Arduino
  - Vi mạch AVR (như ATmega328P) hoặc SAMD (như ATSAMD21).
  - Vi mạch chứa bộ xử lý CPU, bộ nhớ lưu trữ chương trình (Flash Memory) và bộ nhớ tạm (SRAM).



2.1 Giới thiệu về Arduino

B) Các thành phần của Arduino

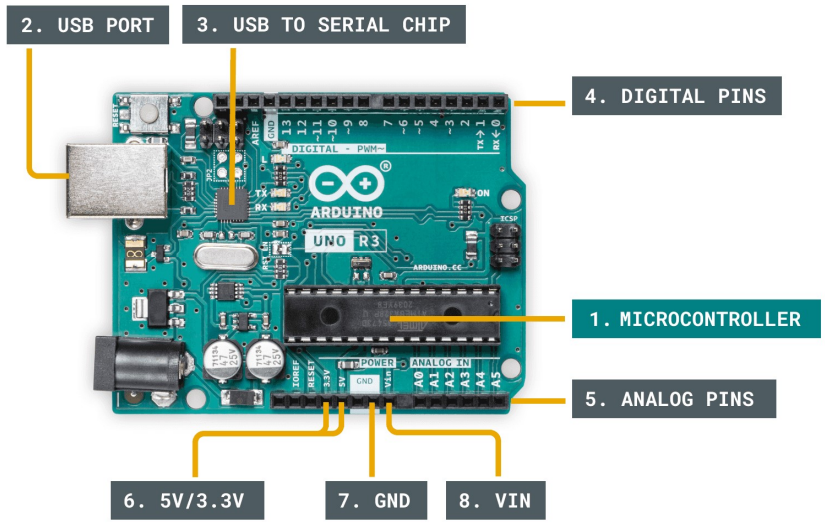
- Cổng Input/Output (I/O Pins)
  - Cổng I/O là các chân GPIO (General Purpose Input/Output) của Arduino.
  - Có nhiều chân I/O, cho phép kết nối và tương tác với các linh kiện ngoại vi.
  - Chân I/O có thể được đặt trong chế độ đầu vào hoặc đầu ra để đọc dữ liệu từ cảm biến hoặc điều khiển các thiết bị khác.



2.1 Giới thiệu về Arduino

B) Các thành phần của Arduino (tiếp)

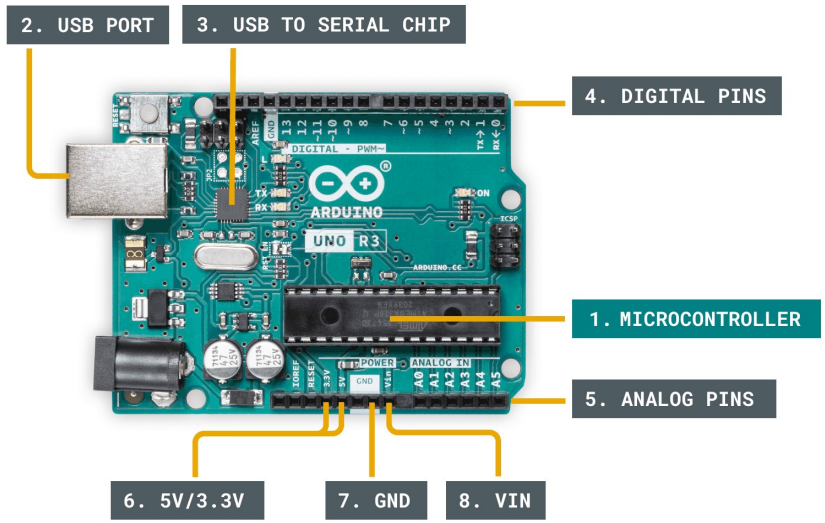
- Cổng Input/Output (I/O Pins)
  - Cổng I/O là các chân GPIO (General Purpose Input/Output) của Arduino.
  - Có nhiều chân I/O, cho phép kết nối và tương tác với các linh kiện ngoại vi.
  - Chân I/O có thể được đặt trong chế độ đầu vào hoặc đầu ra để đọc dữ liệu từ cảm biến hoặc điều khiển các thiết bị khác.



2.1 Giới thiệu về Arduino

B) Các thành phần của Arduino (tiếp)

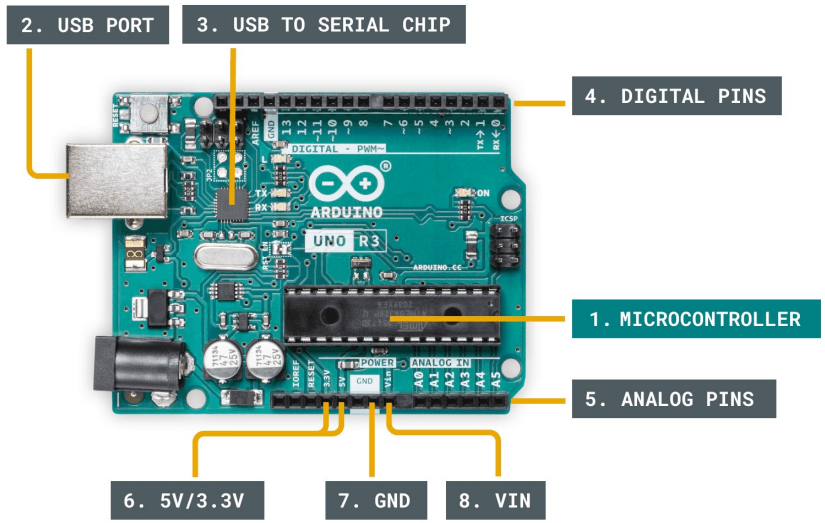
- Cổng Input/Output (I/O Pins)
  - Cổng I/O là các chân GPIO (General Purpose Input/C của Arduino.
    - Analog pin
    - Digital pin
  - Có nhiều chân I/O, cho phép kết nối và tương tác vớ kiện ngoại vi.
  - Chân I/O có thể được đặt trong chế độ đầu vào hoặ để đọc dữ liệu từ cảm biến hoặc điều khiển các thiế khác.
- Giao tiếp: Arduino hỗ trợ nhiều giao thức để kết nố các thiết bị ngoại vi.
  - UART, I2C, SPI, USB



2.1 Giới thiệu về Arduino

C) Các dòng board Arduino:

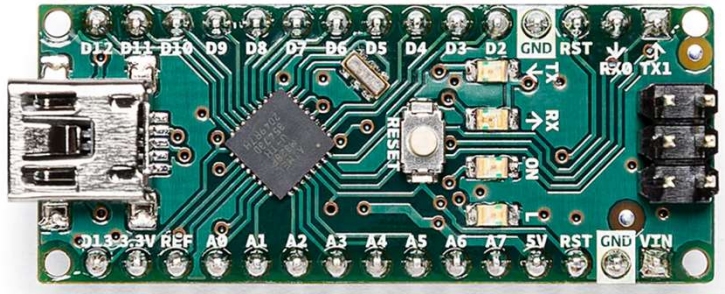
- Arduino Uno:
  - Board Arduino phổ biến nhất
  - Vi mạch ATmega328P, SRAM 2KB, Memory 32KB
  - 14 chân I/O, 6 chân ADC (Analog to Digital Converter), cổng USB
  - Phù hợp với người mới bắt đầu



2.1 Giới thiệu về Arduino

C) Các dòng board Arduino (tiếp):

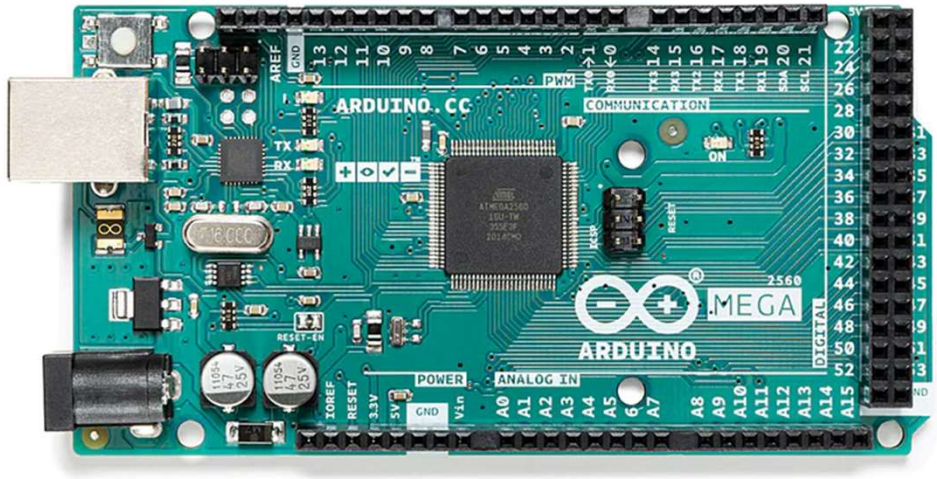
- Arduino Nano:
  - Phiên bản nhỏ gọn hơn của Arduino Uno,
  - Vi mạch ATmega328,
  - 14 chân I/O, 8 chân ADC (Analog to Digital Converter), cổng mini-B USB
  - Thích hợp cho các ứng dụng có hạn chế không gian.



2.1 Giới thiệu về Arduino

C) Các dòng board Arduino (tiếp):

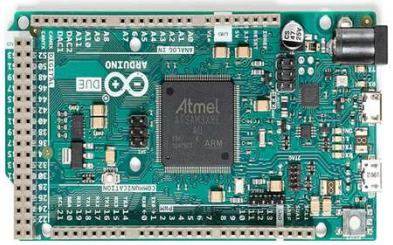
- Arduino Mega:
  - Vi mạch ATmega2560, SRAM 8KB, Memory 256 KB
  - 54 chân I/O, 8 chân ADC (Analog to Digital Converter),
  - Phù hợp cho các dự án phức tạp và yêu cầu nhiều kết nối.



2.1 Giới thiệu về Arduino

C) Các dòng board Arduino (tiếp):

- Các dòng Arduino khác:
  - Arduino MKR Series: Loạt board MKR (Maker) nhỏ gọn và tiêu thụ ít năng lượng, được thiết kế cho ứng dụng IoT và các thiết bị di động.
  - Các biến thể khác từ dòng Uno, Nano, Mega: Micro, Due, …
  - Mỗi dòng Arduino có thể có các shield hỗ trợ, là các module thực hiện các chức năng chuyên biệt hỗ trợ: Ethenet, điều khiển động cơ, GPS, relay,…



2.2 Lập trình với Arduino

2.2.1 Môi trường phát triển Arduino:

- Arduino IDE (Integrated Development Environment)
  - Soạn thảo mã nguồn, biên dịch và nạp chương trình vào board Arduino
  - Miễn phí
  - Hỗ trợ đa nền tảng

ARDUINO IDE


VERIFY/UPLOAD
SELECT BOARD & PORT
OPEN SERIAL MONITOR
AnalogReadSerial | Arduino IDE 2.0.0-rc9
Arduino MKR WiFi 1010
SKETCHBOOK
OPEN SERIAL PLOTTER
BOARD MANAGER
LIBRARY MANAGER
DEBUGGER
SEARCH
/*
AnalogReadSerial
Reads an analog input on pin 0, prints the result to the Serial Monitor.
Graphical representation is available using Serial Plotter (Tools > Serial Plotter menu).
in of a potentiometer to pin A0, and the outside pins to +5V and ground.
This example code is in the public domain.
https://www.arduino.cc/en/Tutorial/BuiltInExamples/AnalogReadSerial
*/
// the setup routine runs once when you press reset:
void setup() {
// initialize serial communication at 9600 bits per second:
Serial.begin(9600);
}
// the loop routine runs over and over again forever:
void loop() {
// read the input on analog pin 0:
int sensorValue = analogRead(A0);
// print out the value you read:

2.2 Lập trình với Arduino

2.2.1 Môi trường phát triển Arduino:

- Các chức năng chính của Arduino IDE 2
  - Sketchbook: chứa các sketches, là các dự án do người dùng tạo ra
    - Mỗi Sketches chứa các file mã nguồn .ino của dự án Arduino
  - Boards Manager: quản lý các "core" của các board Arduino, cần thiết để biên dịch dự án Arduino cho mỗi loại board
  - Library Manager: quản lý các thư viện hỗ trợ cho dự án Arduino
  - Serial Monitor: công cụ hiển thị dữ liệu truyền từ board Arduino sang máy tính qua cổng Serial
  - Serial Plotter: hiển thị dữ liệu từ Arduino dưới dạng đồ thị
  - Debugging: Công cụ gỡ lỗi
  - Autocompletion: tính năng tự động hoàn thành code
  - Remote Sketchbook: làm việc với Arduio Cloud

2.2 Lập trình với Arduino

2.2.2 Tạo chương trình đầu tiên với Arduino

- Chuẩn bị: Cài đặt đúng gói thư viên “core” với board đang sử dụng nếu chưa cài đặt
- Bước 1: Tạo một sketch mới hoặc chọn một sketch có sẵn. VD: Blink
- Bước 2: Viết mã nguồn cho file ino. VD: Blink.ino
- Bước 3: Biên dịch sketch
- Bước 4: Upload chương trình lên board Arduino
- Bước 5 (tùy chọn): Theo dõi Serial Monitor

2.2 Lập trình với Arduino

2.2.2 Tạo chương trình đầu tiên với Arduino

- Mã nguồn chương trình Blink.ino

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

2.2 Lập trình với Arduino

2.2.3 Các khái niệm cơ bản của chương trình Arduino

- Hàm **setup() :** hàm chỉ chạy duy nhất 1 lần khi board Arduino được bật hoặc khi vừa nạp xong chương trình. Thường được sử dụng để thiết lập các cài đặt ban đầu cho chương trình.
- Hàm **loop() :** hàm được chạy lặp đi lặp lại chừng nào Arduino còn được bật, thực hiện các chức năng điều khiển chính của Arduino
- Hàm **delay():** hàm dừng chương trình lại một khoảng thời gian tính bằng milisecond

2.2 Lập trình với Arduino

2.2.3 Các khái niệm cơ bản của chương trình Arduino

- Hàm **millis() :** hàm trả về nhãn thời gian hiện tại tính từ khi chương trình bắt đầu.
- Giao tiếp với máy tính qua Serial : cách cơ bản nhất để biết chương trình đang chạy trên board như thế nào.
  - **Serial.begin(<baud rate>):** bắt đầu giao tiếp với máy tính qua cổng serial ở tốc độ <baud rate>
  - **Serial.print(<msg>):** in giá trị <msg> qua cổng Serial, có thể xem bằng Serial Monitor
  - **Serial.read():** đọc giá trị từ máy tính vào board

2.2 Lập trình với Arduino

2.2.3 Các khái niệm cơ bản của chương trình Arduino

- Các hàm thao tác với các chân GPIO:
  - pinMode(): thiết lập chân GPIO output hay input
  - digitalRead(): đọc giá trị từ chân digital (0 hoặc 1)
  - digitalWrite(): thiết lập giá trị cho 1 chân digital (HIGH hoặc LOW)
  - analogRead(): đọc giá trị từ chân analog (0 đến 1023, độ phân giải 10 bit)
  - analogWrite(): ghi giá trị ra chân analog

2.2 Lập trình với Arduino

2.2.3 Các khái niệm cơ bản của chương trình Arduino

- Đặc điểm ngôn ngữ :khai báo biến, hàm, lệnh có cấu trúc, OOP,… tương tự C++

2.2 Lập trình với Arduino

2.2.4 Công cụ mô phỏng Arduino

- **Tinkercad**: ứng dụng web cho phép thiết lập mô phỏng nhiều loại dự án kỹ thuật, trong đó có Arduino
- Wokwi: ứng dụng web mô phỏng mạch với Arduino với đầy đủ các thư viện
- Proteus: công cụ mạnh mẽ cho thiết kế mạch

2.2 Lập trình với Arduino

2.2.5 Nguyên tắc cần nhớ khi phát triển ứng dụng Arduino:

- Không chạy Arduino khi chưa chắc chắn về thiết kế
  - Cần thử nghiệm thiết kế trước trên công cụ mô phỏng
  - Kiểm tra kỹ lưỡng mạch trước khi chạy
- Không lắp ráp mạch Arduino khi Arduino đang hoạt động
- Cấp đúng và đủ nguồn cho mạch

2.2 Lập trình với Arduino

2.2.5 Nguyên tắc cần nhớ khi phát triển ứng dụng Arduino:

- Các lỗi lắp ráp có thể làm hỏng Arduino và linh kiện:
  - Nối trực tiếp chân có điện áp cao xuống chân GND
  - Nối trực tiếp chân có điện áp cao xuống chân có điện áp thấp
  - Nối ngược chiều dòng điện : cấp nguồn vào chân 5v và lấy ra ở chân Vin, nối ngược GND và Vin với nguồn, …
  - Cấp điện áp cao vượt quy định cho Arduino
  - Bắt vi điều khiển Arduino cấp nguồn quá 200mA cho các thiết bị khác.

2.3 Làm việc với module ngoại vi

2.3.1. GPIO (General Purpose Input/Output)

2.3.1.1 Khái niệm

- GPIO là một thành phần quan trọng trong các vi mạch điện tử, bao gồm cả các board Arduino
- GPIO cho phép vi mạch tương tác với thế giới ngoài thông qua các chân điều khiển.
- Có 2 loại chân GPIO:
  - Digital pin
  - Analog pin

2.3 Làm việc với module ngoại vi

2.3.1. GPIO (General Purpose Input/Output)

2.3.1.2 Chế độ làm việc của các chân GPIO

- INPUT: chế độ làm việc mặc định của chân GPIO
  - Trở kháng rất cao (100MΩ)
  - Sử dụng 1 điện trở để phân biệt trạng thái của chân khi không có tín hiệu input từ ngoài vào (pullup resistor, pulldown resistor)
  - Dùng để đọc dữ liệu từ cảm biến và các module khác (ETC e.g…)
- INPUT_PULLUP: tương tự INPUT nhưng có sẵn điện trở pullup 20KΩ

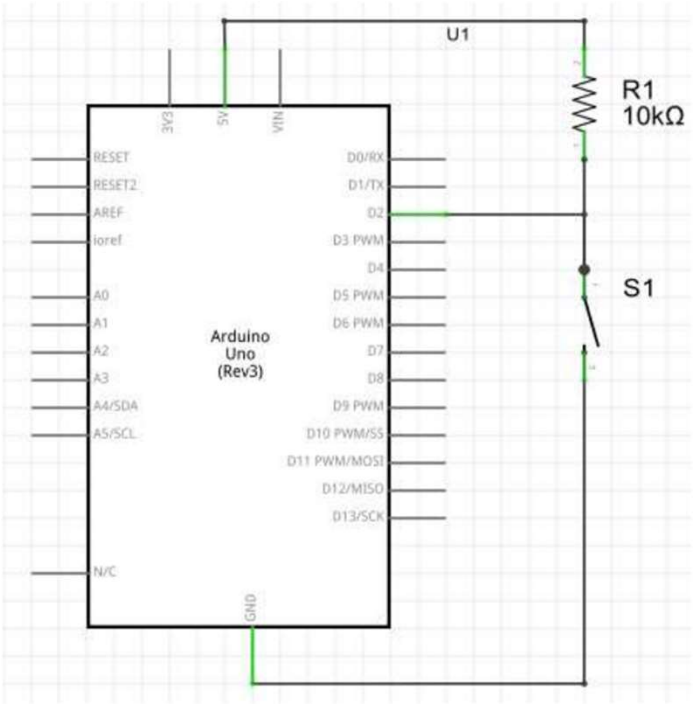
2.3 Làm việc với module ngoại vi

2.3.1. GPIO (General Purpose Input/Output)

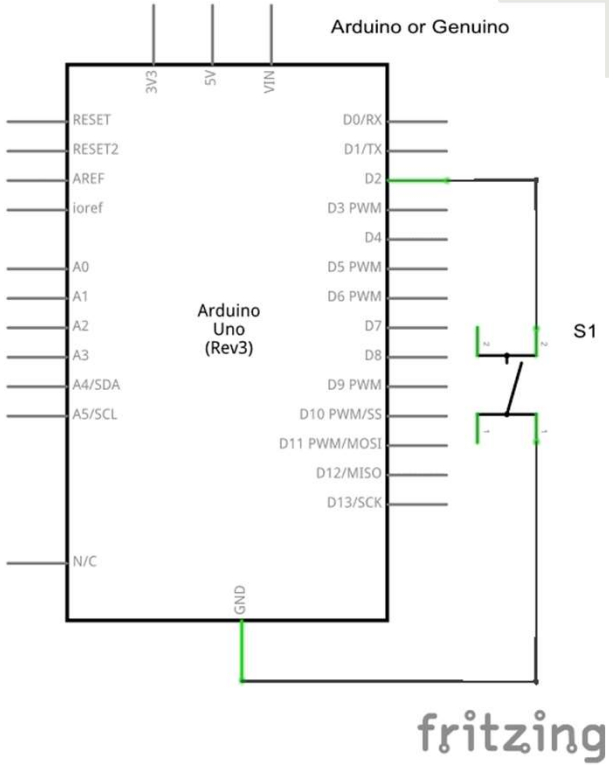
2.3.1.2 Chế độ làm việc của các chân GPIO

- INPUT: chế độ làm việc mặc định của chân GPIO
  - Trở kháng rất cao (100MΩ)
  - Sử dụng 1 điện trở để phân biệt trạng thái của chân khi không có tín hiệu input từ ngoài vào (pullup resistor, pulldown resistor)
  - Dùng để đọc dữ liệu từ cảm biến và các module khác (ETC e.g…)
- INPUT_PULLUP: tương tự INPUT nhưng có sẵn điện trở pullup 20KΩ

2.3 Làm việc với module ngoại vi



Chế độ INPUT, sử dụng trở 10KΩ làm pullup



Chế độ INPUT_PULLUP, sử dụng trở có sẵn của Arduino làm pullup

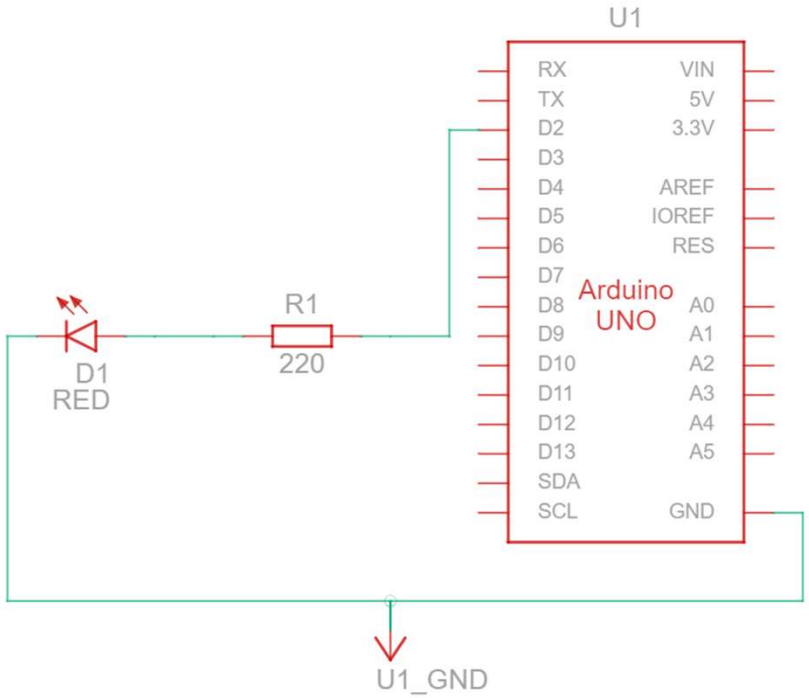
2.3 Làm việc với module ngoại vi

2.3.1. GPIO (General Purpose Input/Output)

2.3.1.2 Chế độ làm việc của các chân GPIO

- OUTPUT:
  - Trở kháng thấp, cho phép cấp dòng điện qua chân GPIO ra mạch ngoài
  - Dòng tối đa chân vi điều khiển có thể cấp là 40mA
    - Chỉ đủ cho 1 đèn LED 20mA hoặc vài loại cảm biến
    - Nếu dòng cao hơn (ví dụ 2 LED song song) có thể gây hỏng chân hoặc hỏng vi điều khiên
    - Nên lắp thêm trở 470Ω hoặc 1KΩ để an toàn hơn

2.3 Làm việc với module ngoại vi



2.3 Làm việc với module ngoại vi

2.3.1. GPIO (General Purpose Input/Output)

2.3.1.2 Các hàm làm việc với chân GPIO:

- pinMode(<chân GPIO>, <Chế độ>)
  - Hàm thiết lập chế độ cho chân GPIO (INPUT, INPUT_PULLUP, OUTPUT)
  - Không trả về giá trị
  - Sử dụng trong hàm setup()
  - Chân GPIO:
    - Chân digital: 2,3,4,5,…
    - Chân analog: A0,A1,…
  - Chế độ:
    - INPUT,INPUT_PULLUP
    - OUTPUT

2.3 Làm việc với module ngoại vi

2.3.1. GPIO (General Purpose Input/Output)

2.3.1.2 Các hàm làm việc với chân GPIO:

- digitalRead(<chân GPIO>)
  - Đọc mức logic HIGH (cao) hoặc LOW (thấp) từ chân GPIO
  - Giá trị trả về : LOW hoặc HIGH
  - Chân GPIO:
    - Chân digital: 2,3,4,5,…
    - Chân analog: A0,A1,…, trừ A6, A7
  - Lưu ý: nếu chân GPIO không nối vào đâu thì giá trị nhận được có thể là HIGH hoặc LOW

2.3 Làm việc với module ngoại vi

2.3.1. GPIO (General Purpose Input/Output)

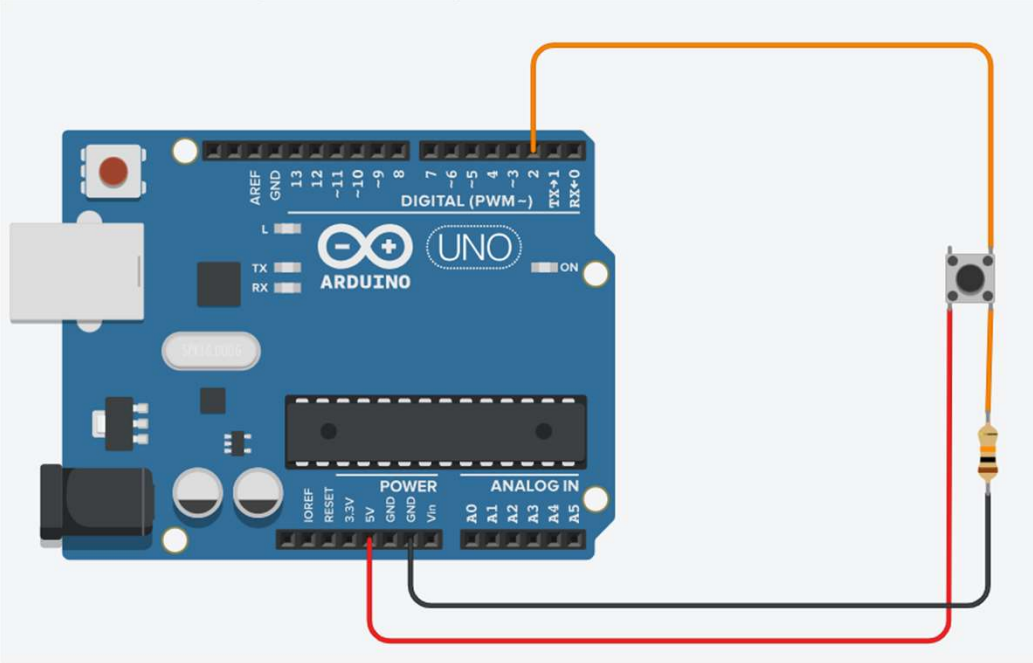
2.3.1.2 Các hàm làm việc với chân GPIO:

- digitalWrite(<chân GPIO>, <mức logic>)
  - Ghi mức logic HIGH (cao) hoặc LOW (thấp) ra chân GPIO
  - Không trả về giá trị
  - Chân GPIO:
    - Chân digital: 2,3,4,5,…
    - Chân analog: A0,A1,…, trừ A6, A7
  - Mức logic: HIGH, LOW
  - Lưu ý: đẩy mức HIGH thì chân GPIO phải được thiết lập chế độ OUTPUT trước

2.3 Làm việc với module ngoại vi

2.3.1. GPIO (General Purpose Input/Output)

2.3.1.3 Một số ví dụ:

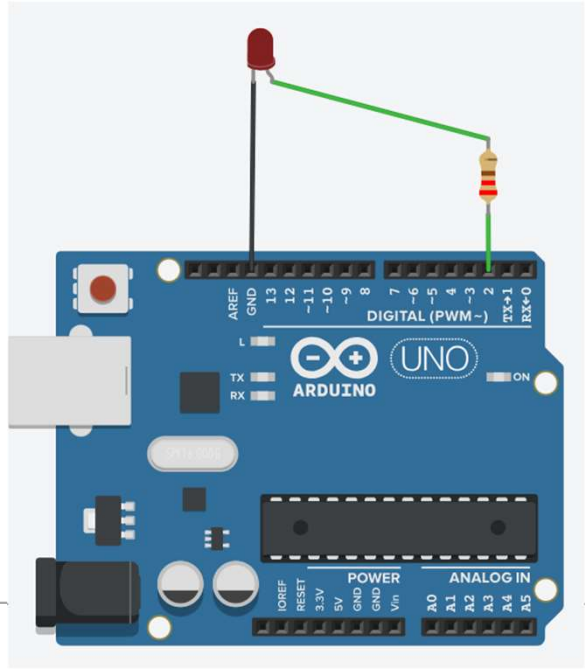



```
int pushButton = 2;
void setup() {
  Serial.begin(9600);
  pinMode(pushButton, INPUT);
}

void loop() {
  int buttonState = digitalRead(pushButton);
  Serial.println(buttonState);
  delay(1);
}
```

2.3 Làm việc với module ngoại vi

2.3.1. GPIO (General Purpose Input/Output)

2.3.1.3 Một số ví dụ:




```
int LED_PIN=2;
void setup()
{
  pinMode(LED_PIN, OUTPUT);
}

void loop()
{
  digitalWrite(LED_PIN, HIGH);
  delay(1000);
  digitalWrite(LED_PIN, LOW);
  delay(1000);
}
```

2.3 Làm việc với module ngoại vi

2.3.1. GPIO (General Purpose Input/Output)

2.3.1.3 Một số ví dụ:

Bài tập tại lớp: thiết kế mạch Arduino và code điều khiển để bật tắt một đèn LED bằng một nút bấm 4 chân. Yêu cầu đèn LED phải được điều khiển qua chân D2.

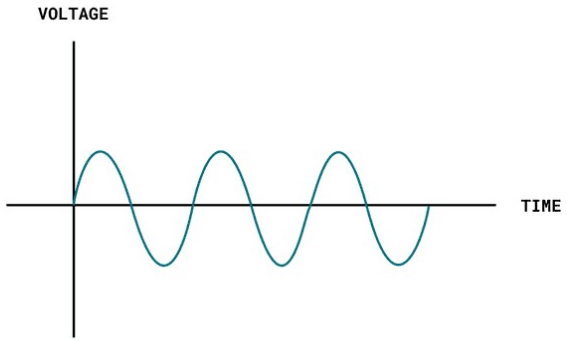
2.3 Làm việc với module ngoại vi

2.3.2. Analog

2.3.2.1 Tín hiệu Analog:

- Tín hiệu tương tự (analog signal) là tín hiệu có độ lớn dao động trong một giới hạn
  - Khác với tín hiệu số (digital signal) chỉ có thể có 2 giá trị là 0 hoặc 1
  - Trong Arduino, giá trị dao động là điện áp từ 0-VCC (0-5v hoặc 0-3.3v)

- Sử dụng để đọc dữ liệu từ cảm biến, biến trở hoặc điều khiển động cơ, đèn,…



2.3 Làm việc với module ngoại vi

2.3.2. Analog

2.3.2.2 Đọc tín hiệu Analog:

- ADC (Analog Digital converter): chuyển đổi tín hiệu tương tự thành số
  - Chip Atmega có 6 kênh (8 trên Nano, 16 trên Mega) ADC, tương ứng các chân A0,A1,A1,A3,A4,A5
  - Chuyển đổi giá trị điện áp 0-VCC thành khoảng 0-1023 (độ phân giải 10bit)
- Hàm analogRead(<GPIO pin>): đọc giá trị analog từ một chân được chỉ định trước
  - Chân GPIO sử dụng phải là các chân Analog IN (A0, A1…)
  - Giá trị trả về trong khoảng 0-1023
  - Chế độ của chân được sử dụng phải là INPUT
    - Nếu trước đó sử dụng chân với mode là OUTPUT thì sau khi chuyển sang INPUT cần có thời gian trễ để cho kết quả đọc đúng

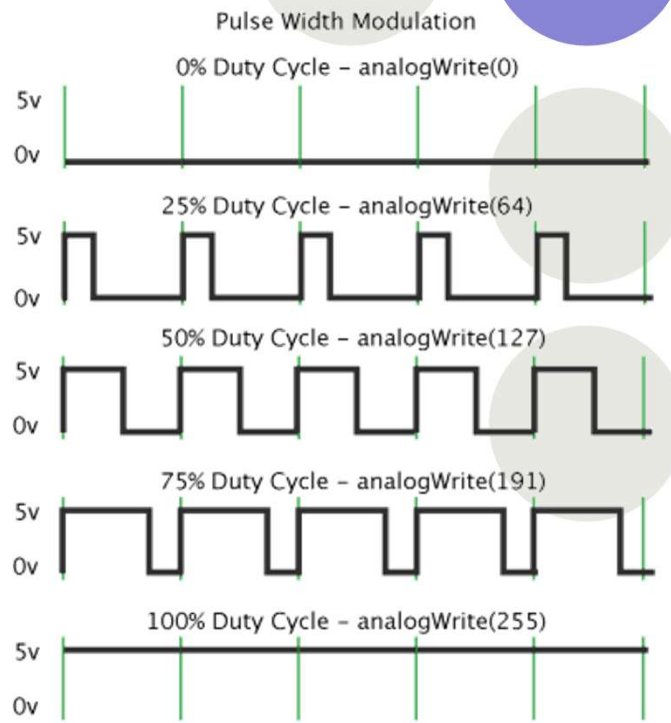
2.3 Làm việc với module ngoại vi

2.3.2. Analog

2.3.2.3 Ghi tín hiệu Analog:

- PWM (Pulse Width Modulation): kỹ thuật cho giá trị tương tự (analog value) bằng tín hiệu số
  - Tín hiệu xung vuông on-off (0 và Vcc) xen kẽ với tỷ lệ on và off sao cho tính trung bình theo thời gian sẽ cho ra giá trị analog cần.

$$V = \frac{T_{ON}}{T_{cycle}} VCC$$

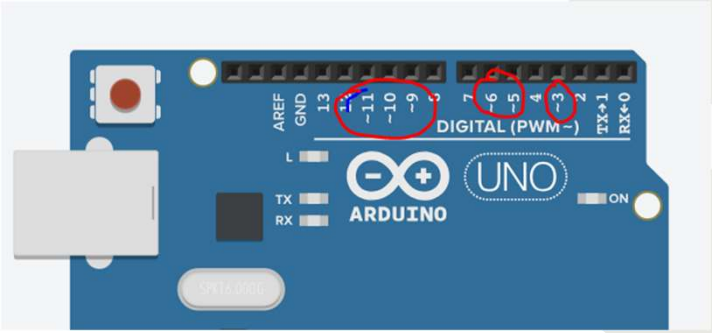


2.3 Làm việc với module ngoại vi

2.3.2. Analog

2.3.2.3 Ghi tín hiệu Analog (tiếp):

- Hàm analogWrite(<PWM pin>, <value>): ghi dữ liệu analog ở dạng điện áp (0-VCC) ra chân PWD theo giá trị value (0-255)
  - PWD pin : các chân được chú thích là PWD (~) trên Arduino. (vd chân 3,5,6,10,11 trên Arduino UNO)
  - Tần số xung : 490Hz (trừ 5,6 có tần số 980Hz)

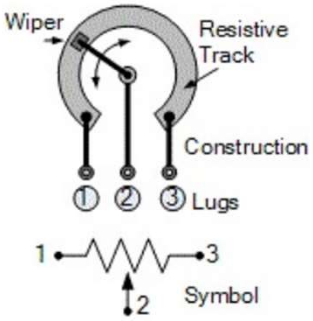


2.3 Làm việc với module ngoại vi

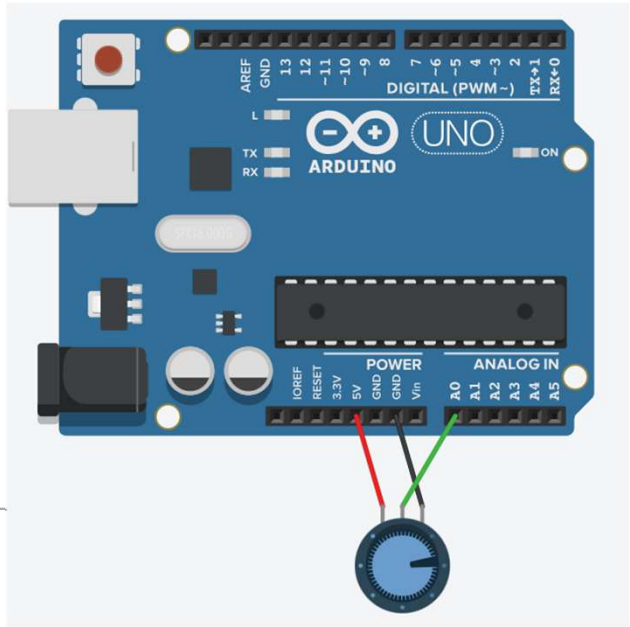
2.3.2. Analog

2.3.2.4 Ví dụ:

Đọc giá trị biến trở và in giá trị ra serial



Biến trở: thiết bị cho phép thay đổi giá trị điện trở bằng núm vặn
- Biến trở 3 chân:
R13=R12+R23
- Mạch chia áp: Nối chân 1, 3 vào nguồn điện 5v, chân 2 có điện áp từ 0 đến 5v theo vị trí núm vặn



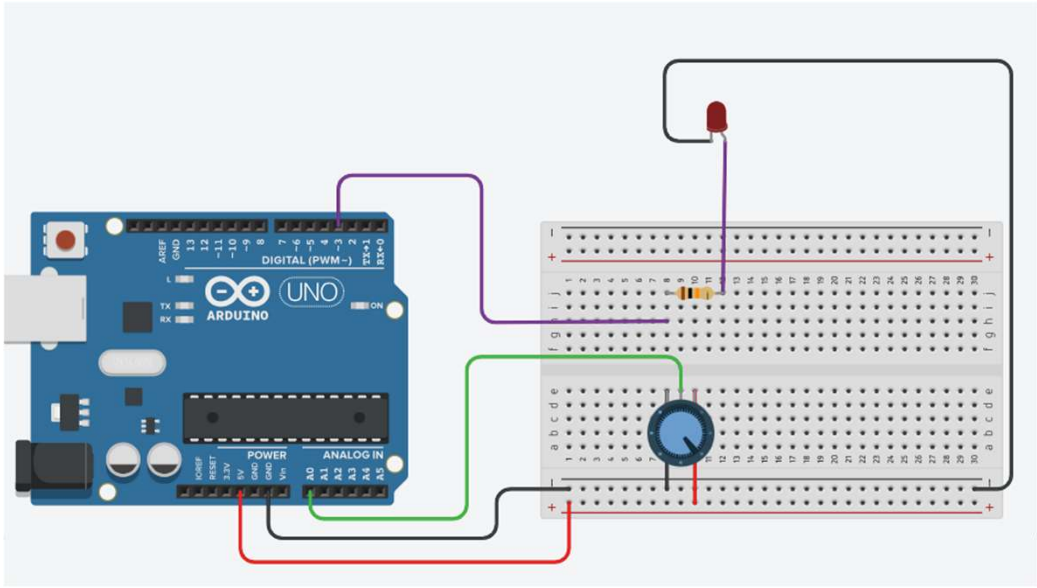

```
int sensorValue = 0;
void setup()
{
  pinMode(A0, INPUT);
  Serial.begin(9600);
}

void loop()
{
sensorValue =
analogRead(A0);
  Serial.println(sensorValue);
  delay(10);
}
```

2.3 Làm việc với module ngoại vi

2.3.2. Analog

2.3.2.4 Ví dụ:

Đọc giá trị biến trở để điều khiển độ sáng đèn LED




```
int ANALOG_PIN=A0;
int LED_PIN=3;
void setup() {
  Serial.begin(9600);
  pinMode(ANALOG_PIN,INPUT);
  pinMode(LED_PIN,OUTPUT);
}
void loop() {
  int sensorValue = analogRead(ANALOG_PIN);
  int out_value=sensorValue/1023.0*255;
  analogWrite(LED_PIN,out_value);
  Serial.println(out_value);
  delay(10);  delay(10);
}
```

2.3 Làm việc với module ngoại vi

2.3.2. Analog

2.3.2.4 Ví dụ:

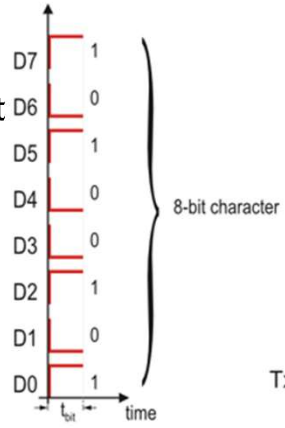
Bài tập tại lớp: thiết kế mạch và viết chương trình điều khiển 1 đèn LED sáng dần và tối dần theo chu kỳ 5s

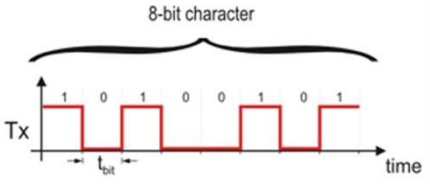
2.3 Làm việc với module ngoại vi

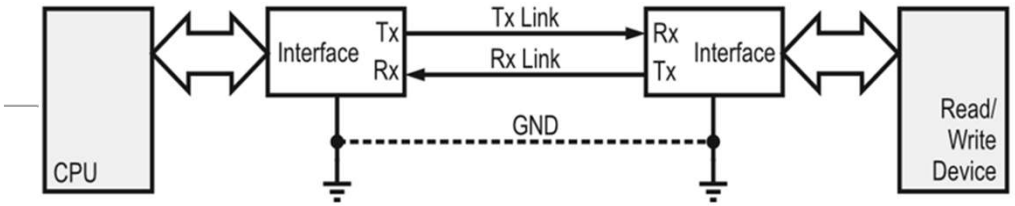
2.3.3. UART

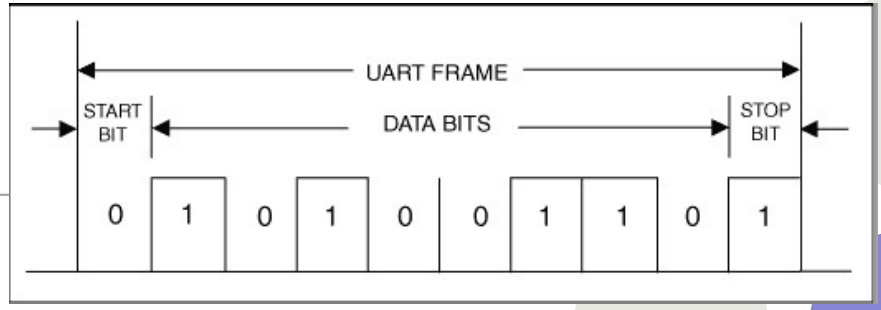
2.3.3.1 Khái niệm:

- Truyền thông nối tiếp (Serial Communication): phương pháp truyền dữ liệu lần lượt từng bit trên một đường truyền.
  - Đồng bộ
  - Không đồng bộ
- Giao thức Serial: chuẩn giao tiếp truyền thông không đồng bộ
- UART (Universal Asynchronous Receiver-Transmitter): phần cứng hỗ trợ truyền dữ liệu theo giao thức Serial
  - 2 dây kết nối Tx và Rx
  - Baudrate: số bit truyền trong mỗi giây
  - Khung truyền: quy định số bit trong mỗi lần truyền

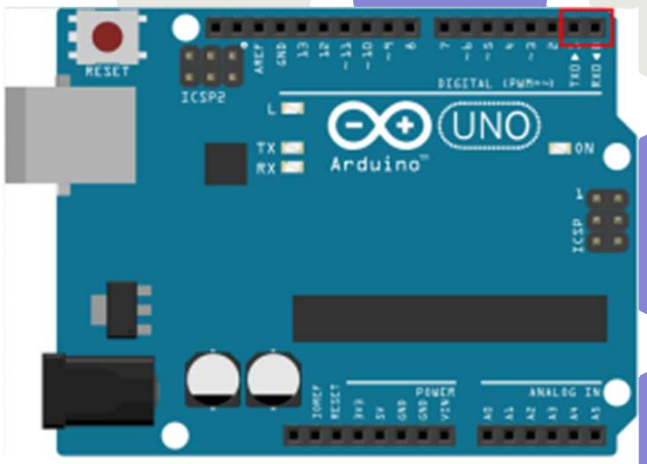








2.3 Làm việc với module ngoại vi



2.3.3. UART

2.3.3.1 Khái niệm:

- UART được sử dụng để truyền thông giữa Arduino với các thiết bị khác: máy tính, module Bluetooth, SIM800, thiết bị Arduino khác…
- Mỗi loại Arduino có số cổng Serial (số kết nối UART) khác nhau:
  - Uno: 1 (pin 0 và 1)
  - Mega: 4
- Sử dụng đối tượng cổng nối tiếp Serial để lập trình kết nối UART trên Arduino
- Khi kết nối với máy tính qua USB là đã sử dụng 1 cổng Serial (USB-to-Serial CH430E)
  - Không thể sử dụng cổng Serial đó để kết nối thiết bị khác

2.3 Làm việc với module ngoại vi

2.3.3. UART

2.3.3.2 Các hàm của đối tượng Serial:

- Serial.begin(baudrate, [config]): thiết lập tốc độ baudrate và cấu hình cho cổng nối tiếp
  - Cấu hình: mặc định là SERIAL_8N1 (8 data bits, no parity và 1 stop bit)
- Serial.end(): dừng giao tiếp
- Serial.print(value,[format]): in dữ liệu text (ký tự ASCI)qua cổng nối tiếp
  - Value: bất kỳ dữ liệu gì, mặc định là text
  - Format: BIN, HEX, OCT, DEC, số chữ số sau dấu phẩy (với value là số thực)
- Serial.println(value,[format]): tương tự như Serial.print(), in thêm ký tự xuống dòng

2.3 Làm việc với module ngoại vi

2.3.3. UART

2.3.3.2 Các hàm của đối tượng Serial:

- Serial.write(value): gửi dữ liệu nhị phân
  - Value: dữ liệu kích 1 byte (byte, char), hoặc mảng ký tự (string)
  - Trả về số byte đã gửi
- Serial.write(arr, len): gửi một mảng bytes
  - arr: mảng bytes
  - Len: số phần tử gửi
  - Trả về số byte đã gửi

2.3 Làm việc với module ngoại vi

2.3.3. UART

2.3.3.2 Các hàm của đối tượng Serial:

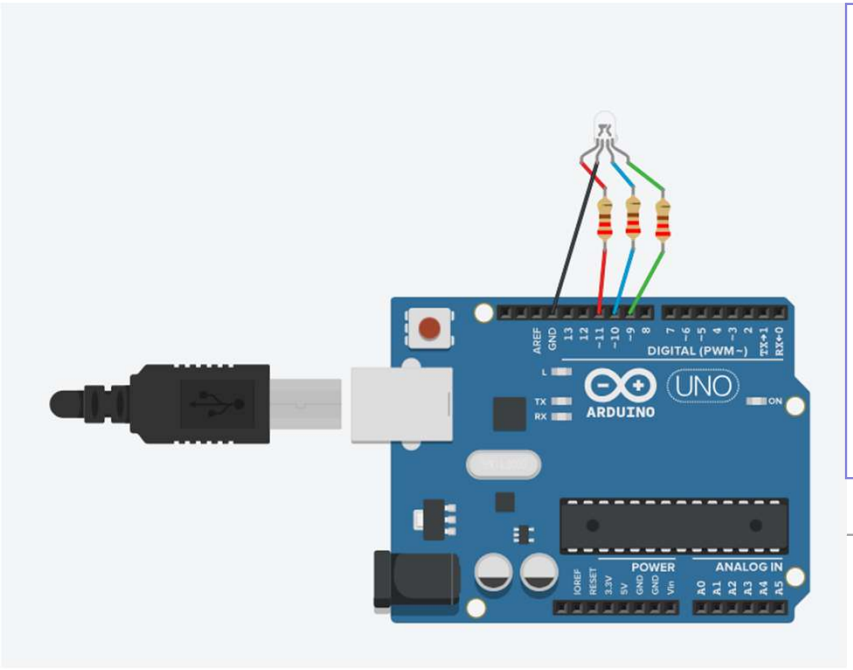
- Serial. available(): trả về số byte có trong bộ đệm nhận
- Serial.read(): đọc 1 byte dữ liệu đến, trả về byte đọc được
- Các hàm khác:
  - Serial. availableForWrite(): trả về số byte đang nằm trong bộ đệm ghi
  - Serial. parseFloat(): trả về giá trị float đầu tiên trong bộ đệm
  - Serial.parseInt(): trả về giá trị int đầu tiên trong bộ đệm
  - Serial. readBytes(buffer, length): đọc mảng byte độ dài length vào bộ đệm buffer

```
int incomingByte = 0;
void setup() {
 Serial.begin(9600);
 }
void loop() {
 if (Serial.available() > 0) {
  incomingByte = Serial.read();
  Serial.print("I received: ");
  Serial.println(incomingByte,
DEC);
}
```

2.3 Làm việc với module ngoại vi

2.3.3. UART

2.3.3.3 Ví dụ: điều khiển Arduino bằng Serial Monitor




```
int RED_PIN=11;
int BLUE_PIN=10;
int GREEN_PIN=9;
int signal=0;
void setup()
{
  Serial.begin(9600);
  pinMode(RED_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
}
```

```
void loop()
{
  if(Serial.available()>0){
        signal=Serial.read();
    Serial.print(signal);
    if(signal=='1'){

digitalWrite(RED_PIN,HIGH);
    }
    if(signal=='2'){

digitalWrite(BLUE_PIN,HIGH);
    }
    if(signal=='3'){

digitalWrite(GREEN_PIN,HIGH);
    }
    delay(100);
    digitalWrite(RED_PIN,LOW);
    digitalWrite(BLUE_PIN,LOW);
    digitalWrite(GREEN_PIN,LOW);
  }
  delay(10);
}
```

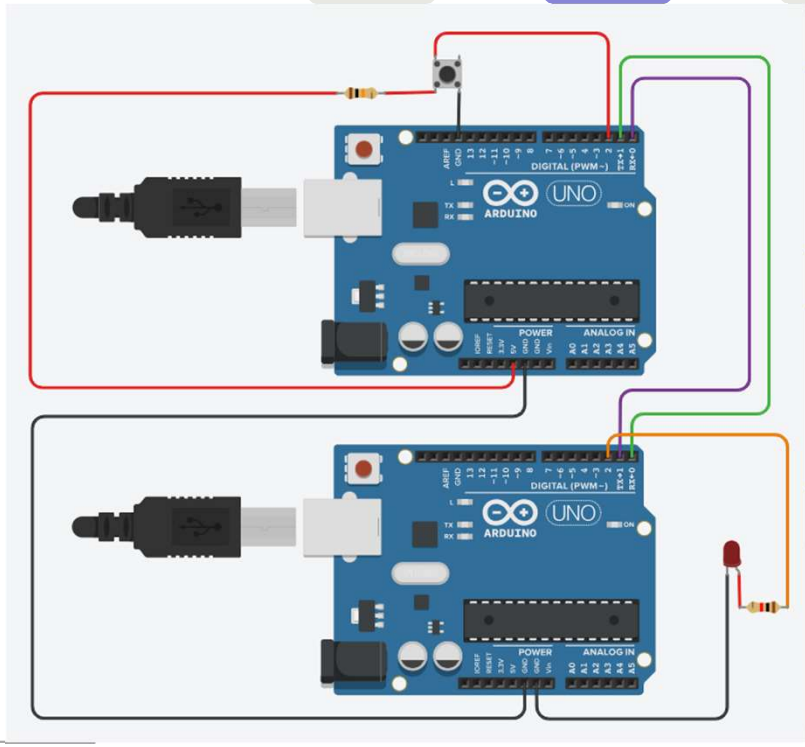
2.3 Làm việc với module ngoại vi

2.3.3. UART

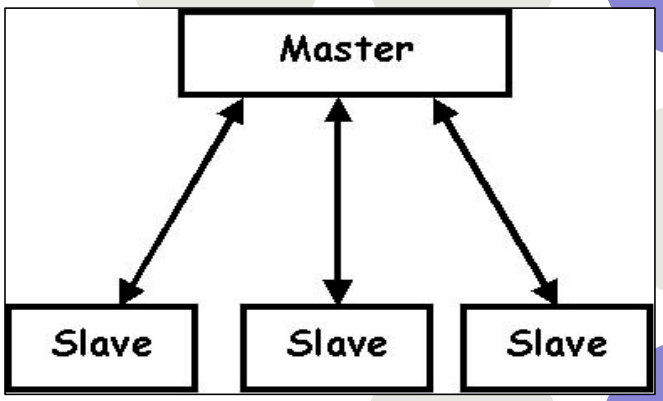
2.3.3.3 Ví dụ: Kết nối truyền dữ liệu giữa hai thiết bị Arduino

```
int buttonState = 0;
void setup() {
 pinMode(2, INPUT);
 Serial.begin(9600);
}
void loop() {
 buttonState = digitalRead(2);
 if (buttonState == HIGH) {
  Serial.println('1');
 }
 else
 {
  Serial.println('2');
 }
 delay(10);
}
```

```
char str;
void setup() {
 pinMode(2, OUTPUT);
 Serial.begin(9600);
}
void loop() {
 str = Serial.read();
 if (str == '1') {
   digitalWrite(2, LOW);
 }
 else if (str == '2') {
  digitalWrite(2, HIGH);
 }
}
```



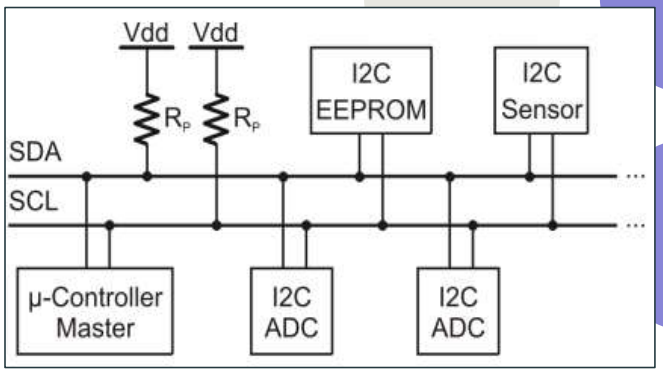
2.3 Làm việc với module ngoại vi

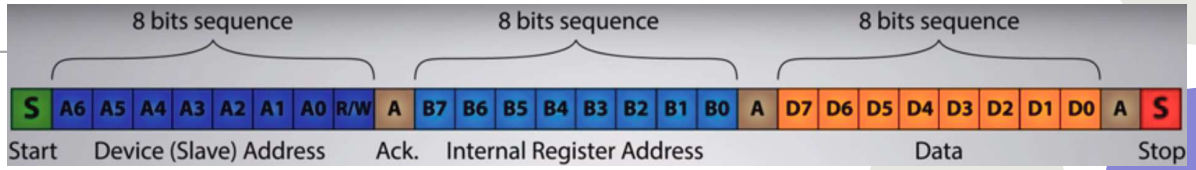


2.3.4. I2C

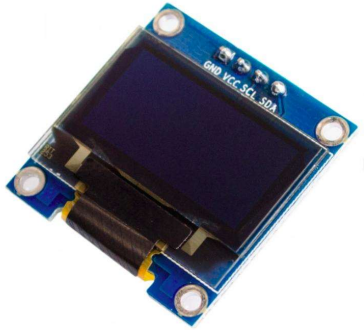
2.3.4.1 Khái niệm:

- Mô hình Master-Slave: mô hình giao tiếp mà một thiết bị có thể điều khiển ít nhất một thiết bị khác.
    - Master: thiết bị điều khiển (vi điều khiển)
    - Slave: thiết bị bị điều khiển (cảm biến, module, vi điều khiển khác)
- I2C: chuẩn giao tiếp đồng bộ kiểu Master/slave
    - Sử dụng 2 dây SCL (Serial Clock Data) và SDA (Serial Data) để giao tiếp
    - Kết nối tối đa 127 thiết bị (địa chỉ 7bit) hoặc 1024 thiết bị (địa chỉ 10bit)
        - Mỗi thiết bị có một địa chỉ trên mạng I2C
    - Dữ liệu được truyền theo mỗi 8bit:
        - Địa chỉ thiết bị trên mạng I2C
        - Địa chỉ thanh ghi nội của thiết bị
        - Dữ liệu truyền



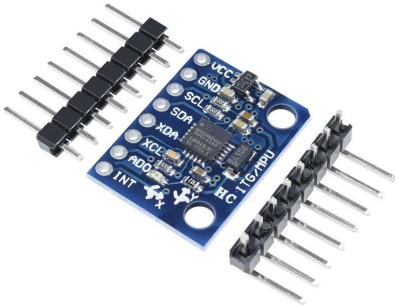


2.3 Làm việc với module ngoại vi



2.3.4. I2C

2.3.4.2 Làm việc với I2C trên Arduino:



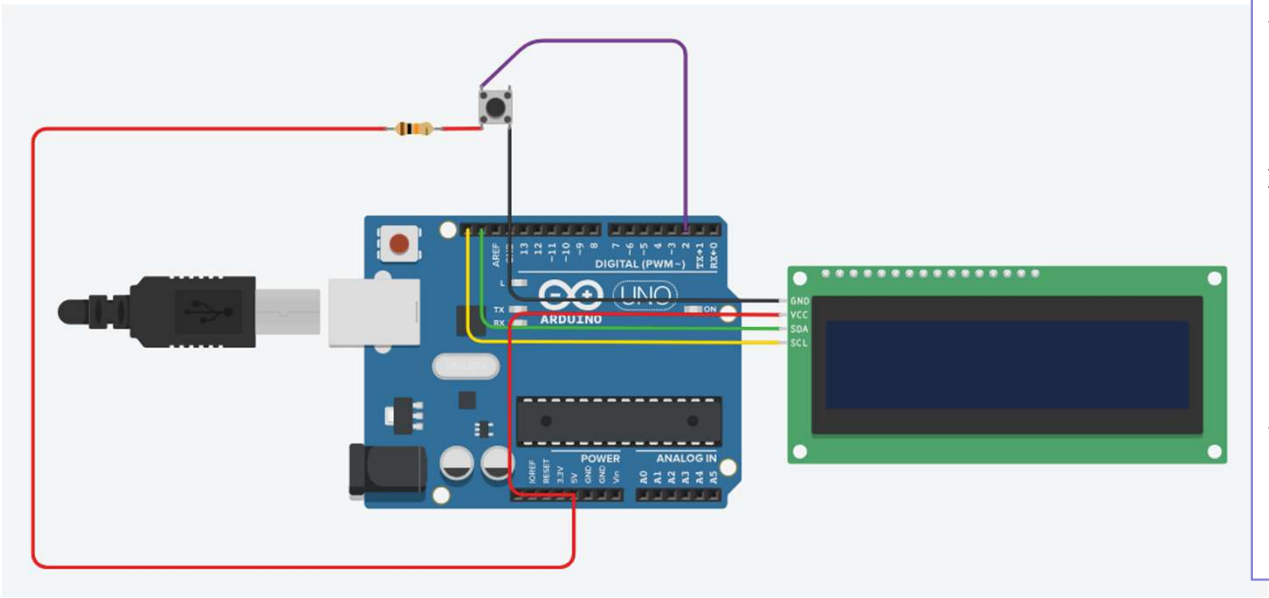
- Kết nối:
  - Thiết bị hỗ trợ kết nối I2C luôn có 2 chân SCL và SDA
  - Arduino : chân có ghi ký hiệu SDA và SCL, hầu hết là A4 và A5
    - UNO: D18,D18
    - Mega: D20,D21
- Thư viện hỗ trợ
  - Wire.h: thư viện tích hợp sẵn trên Arduino IDE, cung cấp các hàm làm việc chung với I2C
  - Các thư viện riêng dành cho từng loại thiết bị: LCD,cảm biến, RTC,…

2.3 Làm việc với module ngoại vi

2.3.4. I2C

2.3.4.3 Ví dụ:

Điều khiển màn hình LCD hỗ trợ giao tiếp I2C




```
#include <Adafruit_LiquidCrystal.h>
int counter = 0;
int BUTTON=2;
int prev_state=-1;
int current_state=-1;
Adafruit_LiquidCrystal lcd_1(0);
void setup()
{
  pinMode(BUTTON,INPUT);
  lcd_1.begin(16, 2);

  lcd_1.print("Number press");
  lcd_1.setBacklight(1);
}
void loop()
{
  current_state=digitalRead(BUTTON);
  if(current_state==HIGH &&
prev_state==LOW){
    counter++;
  }
  display(counter);
  prev_state=current_state;
  delay(100);
}
void display(int value){
        lcd_1.setCursor(0, 1);
        lcd_1.print(value);
}
```

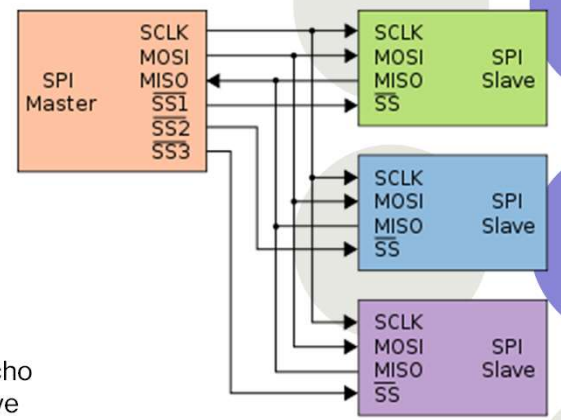
2.3 Làm việc với module ngoại vi

2.3.5. SPI

2.3.5.1 Khái niệm chuẩn giao tiếp SPI:

- SPI (Serial Peripheral Interface): chuẩn giao tiếp đồng bộ tốc độ cao
  - Kiểu master/slave: cho phép 1 master làm việc với nhiều slave
  - Khả năng truyền song công (full duplex)
  - Ứng dụng trong các giao tiếp cần tốc độ cao: SD card, LCD,…
- SPI sử dụng 4 đường truyền:
  - SCK (Serial Clock): xung giữ nhịp của master
  - MISO (Master IN/Slave OUT): đường truyền dữ liệu từ Slave đến Master
  - MOSI (Master OUT/ Slave IN): đường truyền dữ liệu từ Master đến Slave
  - SS (Slave Select): chân chọn Slave để giao tiếp, nối với từng slave riêng biệt

Chân chung cho các slave

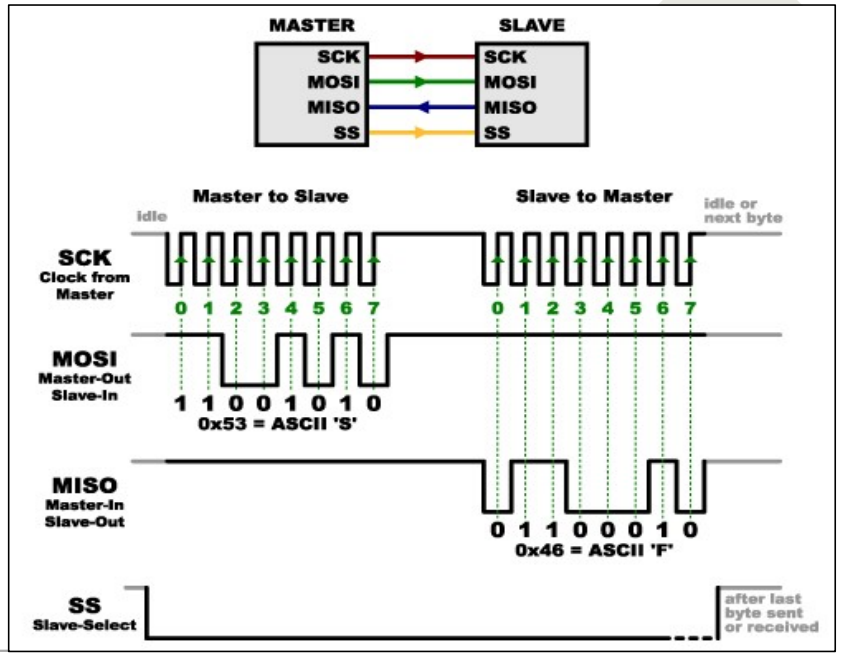


2.3 Làm việc với module ngoại vi

2.3.5. SPI

2.3.5.1 Khái niệm chuẩn giao tiếp SPI:

- Truyền dữ liệu với SPI:
  - Chân SS kéo xuống thấp để chọn slave giao tiếp với master
  - Mỗi xung nhịp chân SCK tạo ra cho phép truyền 1 bit dữ liệu từ master đến slave và ngược lại.



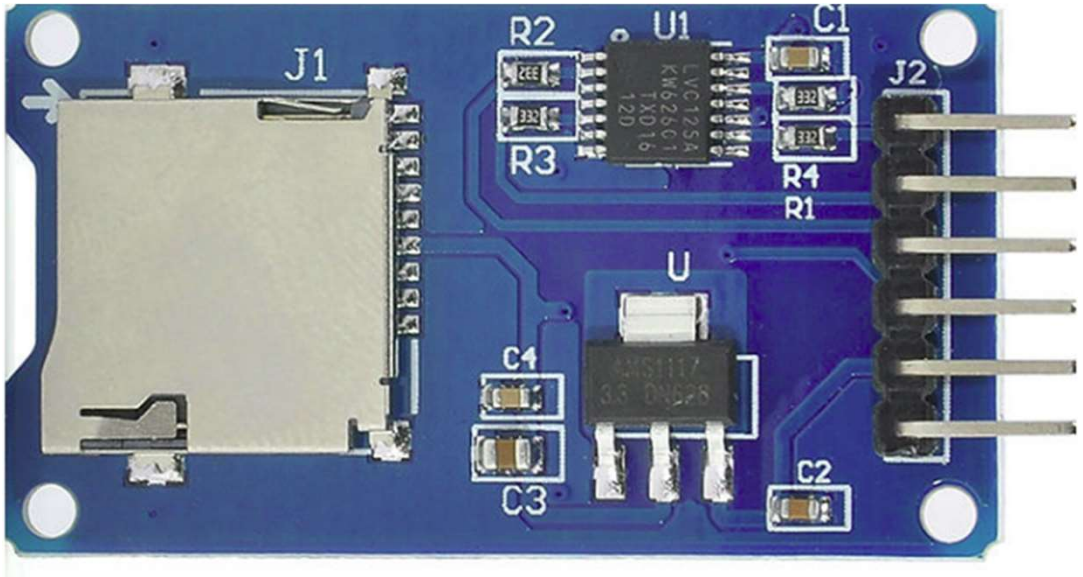
2.3 Làm việc với module ngoại vi

2.3.5. SPI

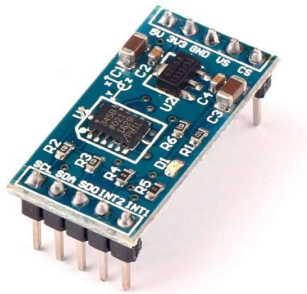
2.3.5.2 Làm việc với SPI trên Arduino:

- Thư viện SPI.h
- Thư viện riêng dành cho từng loại thiết bị
  - Modul Thẻ nhớ
  - Màn hình LCD, OLED
  - Cảm biến gia tốc

  …



TFT 1.44 SPI



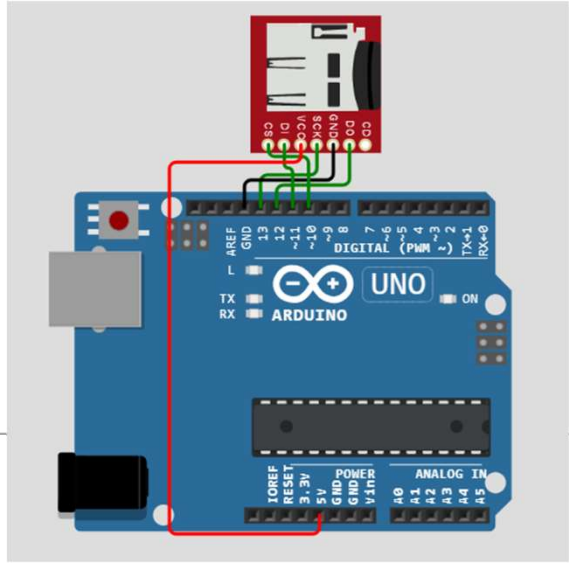
2.3 Làm việc với module ngoại vi

https://wokwi.com/projects/310692660849410626

2.3.5. SPI

2.3.5.3 Ví dụ:

Đọc ghi dữ liệu module thẻ nhớ micro SD (Wokwi)




```
#include "SdFat.h"
#define SPI_SPEED SD_SCK_MHZ(4)
#define CS_PIN 10
SdFat sd;

void setup() {
  Serial.begin(115200);
  if (!sd.begin(CS_PIN, SPI_SPEED)) {
    if (sd.card()->errorCode()) {
      Serial.println("SD initialization
failed.");
    } else if (sd.vol()->fatType() == 0) {
      Serial.println("Can't find a valid
FAT16/FAT32 partition.");
    } else {
      Serial.println("Can't determine
error type");
    }
    return;
  }
  Serial.println("Files on card:");
  Serial.println("   Size    Name");

  sd.ls(LS_R | LS_SIZE);
}
void loop() {
}
```

2.3 Làm việc với module ngoại vi

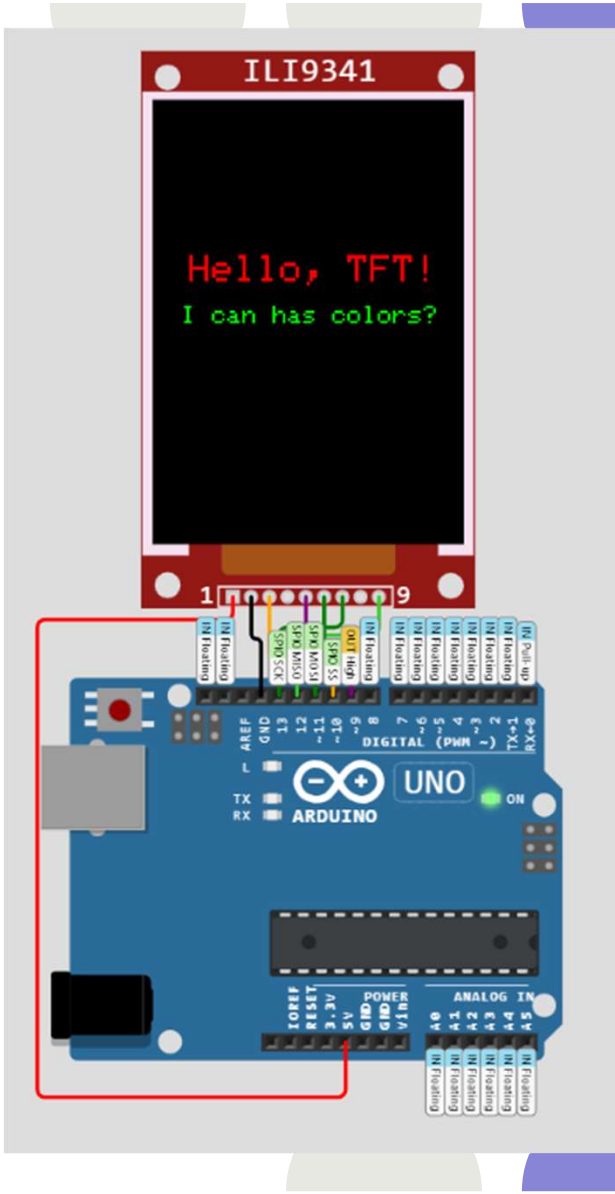
2.3.5. SPI

2.3.5.3 Ví dụ:

Làm việc với màn hình LCD (Wokwi)

```
#include "SPI.h"
#include "Adafruit_GFX.h"
#include
"Adafruit_ILI9341.h"
#define TFT_DC 9
#define TFT_CS 10
Adafruit_ILI9341 tft =
Adafruit_ILI9341(TFT_CS,
TFT_DC);
```

```
void setup() {
  tft.begin();

  tft.setCursor(26, 120);
  tft.setTextColor(ILI9341_RE
D);
  tft.setTextSize(3);
  tft.println("Hello, TFT!");

  tft.setCursor(20, 160);
  tft.setTextColor(ILI9341_GR
EEN);
  tft.setTextSize(2);
  tft.println("I can has
colors?");

}

void loop() { }
```



https://wokwi.com/projects/308024602434470466

2.3 Làm việc với module ngoại vi

2.3.6. Ngắt (interrupt)

2.3.6.1 Khái niệm:

- Làm thế nào để phát hiện và xử lý một sự kiện xảy ra với Arduino?

  - Gắn sự kiện với một biến trạng thái, tại mỗi lần chạy hàm loop, kiểm tra trạng thái xem có thay đổi hay không.

  - Sử dụng ngắt (interrupt)

- Ngắt: một phản ứng của bộ xử lý đối với một sự kiện cần được phần mềm chú ý

  - Khi có sự thay đổi ở thiết bị vật lý, vi xử lý tạm dừng hoạt động hiện tại, lưu trạng thái, thực thi một chức năng (hàm ISR) để xử lý. Sau khi hàm ISR thực thi xong, vi xử lý tiếp tục công việc hiện tại.

2.3 Làm việc với module ngoại vi

2.3.6. Ngắt (interrupt)

2.3.6.2 Các hàm làm việc với ngắt trong Arduino:

- **attachInterrupt(Interrupt ID, ISR, mode):** đăng ký một chương trình con được gọi khi có thay đổi ở một chân digital
    - **Interrupt ID** : số thứ tự của chân ngắt mà khi thay đổi trạng thái sẽ kích hoạt ngắt
        - UNO: có 2 ngắt 0 và 1 (tương ứng chân 2,3)
        - Mega: có 6 ngắt (0->5) (chân 2,3,21,20,19,18)
    - ISR: tên hàm được gọi khi phát sinh ngắt
    - mode: định nghĩa kiểu thay đổi của chân digital làm phát sinh ngắt:
        - LOW: phát sinh ngắt khi chân có điện áp thấp
        - HIGH: phát sinh ngắt khi chân có điện áp cao
        - RISING: phát sinh ngắt khi chân có điện áp thấp chuyển lên điện áp cao
        - FALLING: phát sinh ngắt khi chân có điện áp cao chuyển xuống điện áp thấp

digitalPinToInterrupt(pin): hàm chuyển đổi chân digital thành số thứ tự của ngắt

2.3 Làm việc với module ngoại vi

2.3.6. Ngắt (interrupt)

2.3.6.2 Các hàm làm việc với ngắt trong Arduino:

- **noInterrupts():** ngừng cho phép ngắt hoạt động, khi gọi hàm này thì đoạn lệnh sau đó các hàm ngắt không được gọi, có tác dung bảo vệ các đoạn lệnh nhạy cảm với thời gian.

- **interrupt():** kích hoạt lại ngắt, khi gọi hàm này thì cho phép các hàm ngắt có thể được gọi

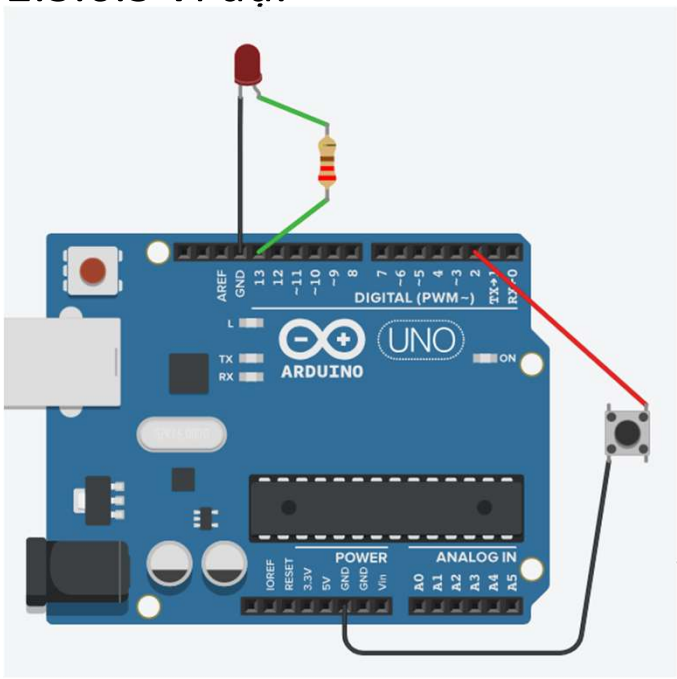
- **detachInterrupt**(pin): tắt ngắt được đính kèm với chân pin

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

2.3 Làm việc với module ngoại vi

2.3.6. Ngắt (interrupt)

2.3.6.3 Ví dụ:




```
int ledPin = 13;
int state=LOW;
void turnOff()
{
    state=!state;
}

void setup()
{
   pinMode(ledPin, OUTPUT);
   pinMode(2, INPUT_PULLUP);
   attachInterrupt(digitalPinToInterrupt(2), turnOff, FALLING);
   Serial.begin(9600);
}

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