Week 3 Thiết Bị IoT

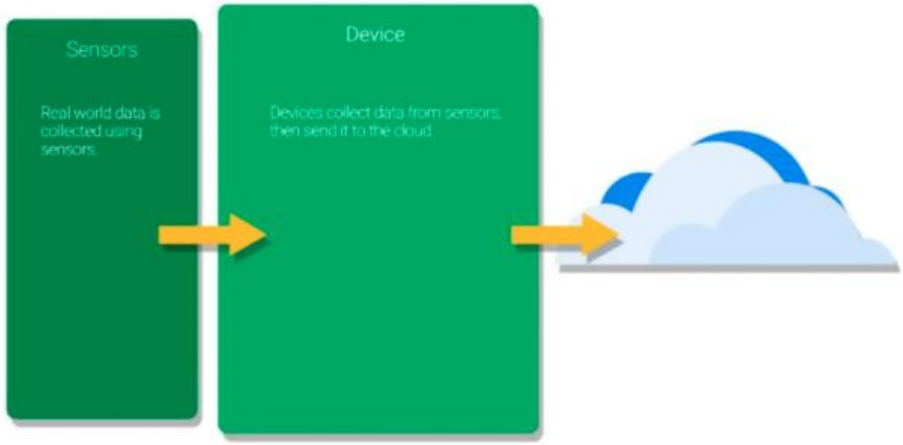
1. Khái Niệm Về Cảm Biến Và Thiết Bị IoT

- Các cảm biến và thiết bị IoT được gọi đơn giản là thiết bị (devices) với ngầm định bao gồm các cảm biến được ghép nối.

- Trong nhiều tình huống khi nói “thiết bị” có thể được hiểu bao gồm cả thiết bị và cảm biến.

1. Khái Niệm Về Cảm Biến Và Thiết Bị IoT

- Các cảm biến đo lường các thông tin về môi trường và gửi thông tin về thiết bị
- Các thiết bị thu thập dữ liệu và gửi về server/cloud
- Các thiết bị hạn chế về tài nguyên: CPU, memory…



1. Khái Niệm Về Cảm Biến Và Thiết Bị IoT

- Devices: một “thing” trong “internet of things” là một đơn vị xử lý có khả năng kết nối internet để trao đổi dữ liệu với server/cloud

- Các thiết bị được gọi là “smart-devices” hay “connected devices”

- Thiết bị giao tiếp với hai loại dữ liệu:
  - Telemetry (dữ liệu thu thập)
  - State (trạng thái của thiết bị)

1. Khái Niệm Về Cảm Biến Và Thiết Bị IoT

- Trong dự án IoT, định nghĩa về thiết bị có thể thay đổi phụ thuộc vào nhu cầu của dự án

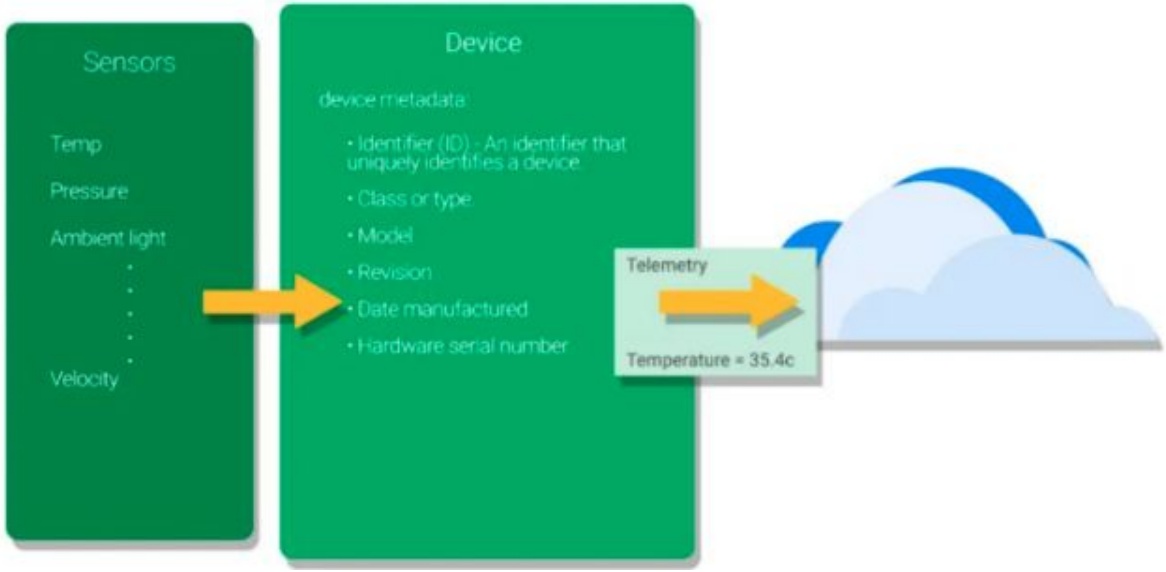
- Đôi khi mỗi thiết bị có thể được cân nhắc như một thực thể tách biệt, hoặc một thiết bị có thể chịu trách nhiệm điều khiển một nhóm nhiều cảm biến

2. Các loại thông tin của thiết bị

- Mỗi thiết bị có thể cung cấp hoặc sử dụng một vài loại thông tin phục vụ cho mỗi loại backend khác nhau.
- **Devices Metadata:**
  - Identity (ID): Định danh thiết bị
  - Class/Type of device: Lớp hoặc loại thiết bị
  - Model
  - Revision
  - Date manufactured
  - Hardware serial number

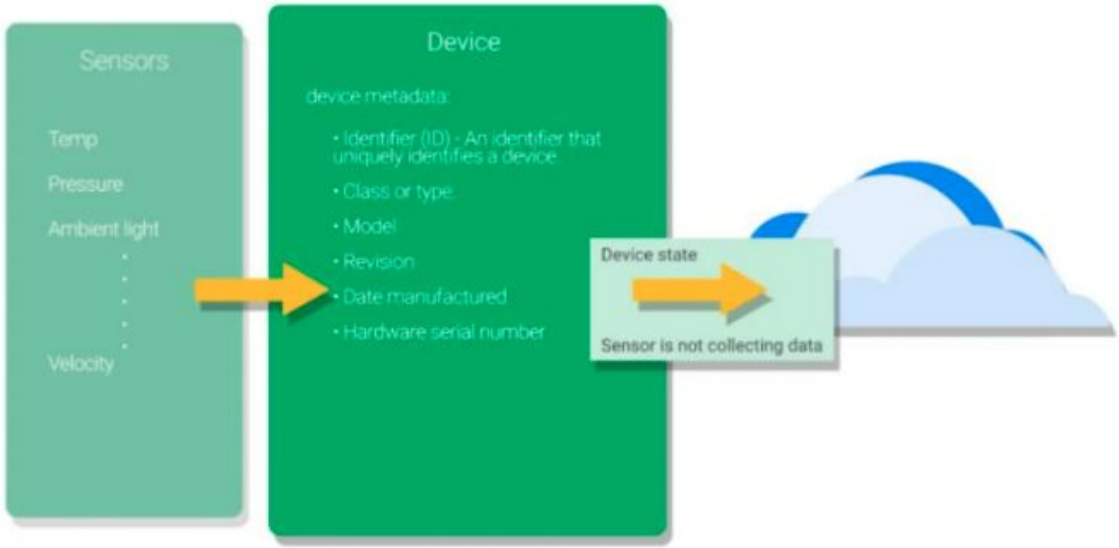
2. Các loại thông tin của thiết bị

- **Telemetry:**
  - Là các dữ liệu được thu thập bởi cảm biến.
  - Là dữ liệu chỉ đọc



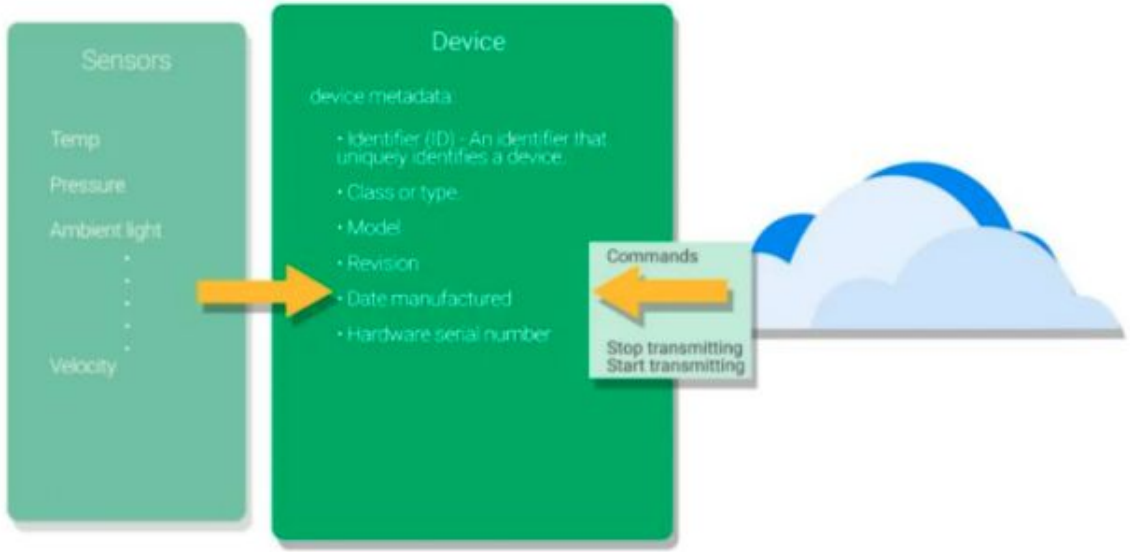
2. Các loại thông tin của thiết bị

- **Device's State:**
  - Mô tả trạng thái hiện tại của thiết bị
  - Có thể đọc/ghi/cập nhật (nhưng không thường xuyên)



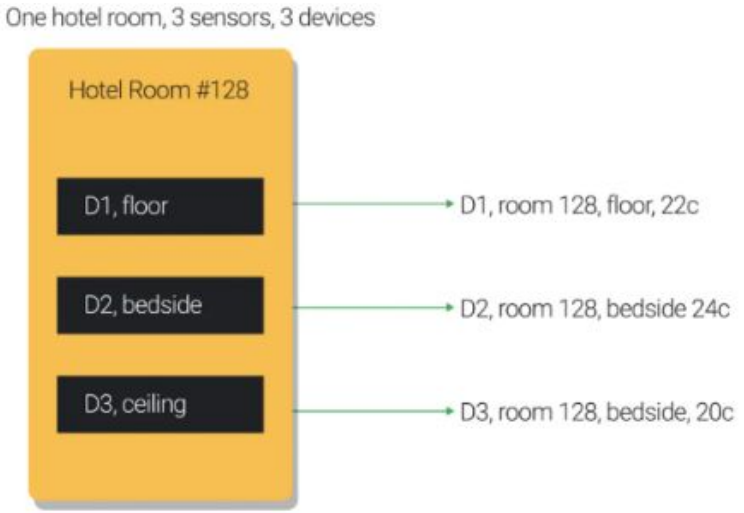
2. Các loại thông tin của thiết bị

- **Command:**
  - Là các hành vi được thực hiện bởi thiết bị
  - Ví dụ: Quay 360 độ sang phải, tắt/bật đèn, thay đổi công suất, v.v…



2. Các loại thông tin của thiết bị

- **<u>Ví dụ 1:</u>**
- Giám sát nhiệt độ phòng khách sạn
- **<u>Giải pháp 1:</u>**
  - Mỗi phòng có 3 cảm biến
  - Sàn, Trần, Giường.

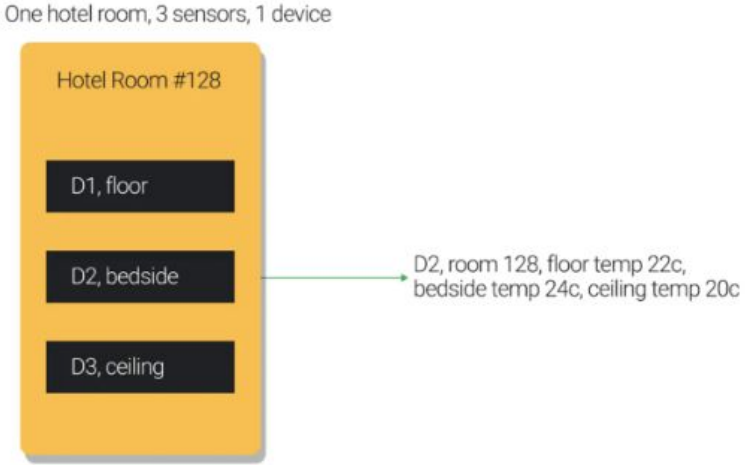


2. Các loại thông tin của thiết bị

- Dữ liệu được gửi đến server/cloud như là 3 thiết bị khác nhau.
  - {deviceID: "dh28dslkja", "location": "floor", "room": 128, "temp": 22 }
  - {deviceID: "8d3kiuhs8a", "location": "ceiling", "room": 128, "temp": 24 }
  - {deviceID: "kd8s8hh3o", "location": "bedside", "room": 128, "temp": 23 }

2. Các loại thông tin của thiết bị

- **Giải pháp 2:**
- Mỗi phòng sử dụng 1 thiết bị chịu trách nhiệm quản lý 3 cảm biến



- {deviceID: "dh28dslkja", "room": 128, "temp_floor": 22, "temp_ceiling": 24, "temp_bedside": 23, "average_temp": 23 }

3. Cảm biến

- **Phân loại cảm biến:**
  - **Theo nguồn cấp**
    - Passive
    - Active
  - **Theo loại tín hiệu**
    - Analog
    - Digital
  - **Theo loại thiết bị đo**
    - Chemical
    - Mechanical
    - Electrical

3. Cảm biến

- **Lựa chọn cảm biến**
  - Lựa chọn cảm biến cần cân nhắc đến tầm quan trọng của các yếu tố và mức độ ưu tiên trong thiết kế tổng thể
  - Durability - Tính bền bỉ:
    - Tính bền bỉ cần được cân nhắc dựa trên mỗi trường hoạt động của cảm biến
    - Đảm bảo thiết bị hoạt động lâu bền trong khoảng thời gian đáng kể không phải tốn chi phí sửa chữa hay thay thế.
    - Ví dụ: cảm biến nhiệt kháng nước cho trạm thời tiết

3. Cảm biến

- **Lựa chọn cảm biến**
  - Accuracy - Độ chính xác:
    - Độ chính xác cần đáp ứng đủ cho nhu cầu
    - Ví dụ:
      - Đo nhiệt độ nhà có thể chấp nhận sai số +/-2 độ
      - Đo nhiệt độ trong y tế cần sai số nhỏ hơn +/- 0.2 độ

3. Cảm biến

- **Lựa chọn cảm biến**
  - Versatility - Tính linh hoạt:
    - Cảm biến có thể vận hành với nhiều biến động của môi trường xung quanh
    - Ví dụ:
      - Cảm biến cho trạm thời tiết có thể hoạt động trong điều kiện mùa hè, mùa đông. Không khả thi với các loại cảm biến chỉ hoạt động trong môi trường trong nhà

3. Cảm biến

- **Lựa chọn cảm biến**
  - Power consumption - Mức tiêu thụ điện năng:
    - Tùy theo tính huống, có thể đòi hỏi các thiết bị tiêu thụ ít điện năng, sử dụng các đặc tính tiết kiệm năng lượng.
    - Ví dụ:
      - Cảm biến hoặc thiết bị sử dụng pin năng lượng mặt trời cần thiết kế chế độ ngủ (sleep mode) tiết kiệm năng lượng, thức dậy (wake-up) khi cần thu thập và truyền dữ liệu

3. Cảm biến

- **Lựa chọn cảm biến**
  - Cân nhắc môi trường đặc biệt
    - Ví dụ:
      - Cảm biến trong giám sát chất lượng nước trong nuôi trồng thủy sản đòi hỏi sensor đặt trong môi trường nước (pH, DO, TDS,...) khác với cảm biến dựa trên lấy mẫu phân tích.

3. Cảm biến

- **Lựa chọn cảm biến**
  - Cost - Giá Thành
    - Các mạng IoT thường bao gồm hàng trăm, hàng ngàn thiết bị và cảm biến.
    - Thiết kế cần cân nhắc đến chi phí lắp đặt, bảo trì, thay thế …

4. Lập trình cho thiết bị IoT

- Các thiết bị dùng vi điều khiển:
  - Kiến trúc 8bit, hiệu năng thấp, không dùng hệ điều hành
  - Lập trình firmware:
    - Giao tiếp vào ra cơ bản.
    - Ghép nối ngoại vi theo các chuẩn truyền thông UART, I2C, SPI
  - Ví dụ: Arduino, ATmega, PIC, AVR, …

4. Lập trình cho thiết bị IoT

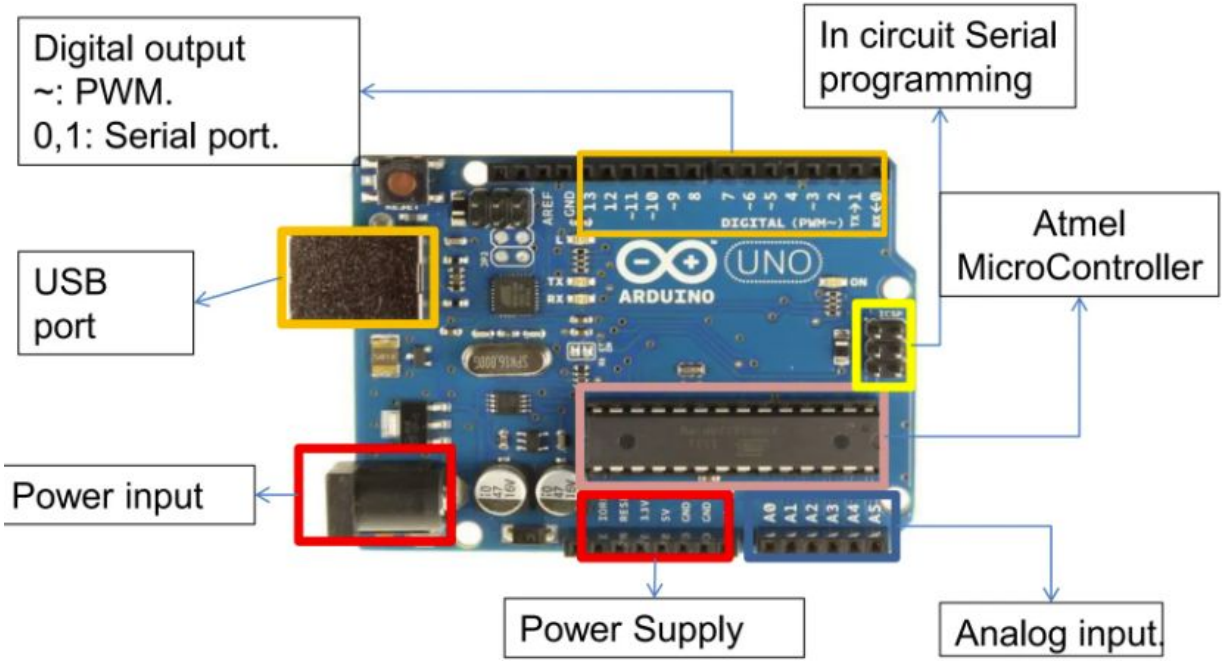
- Các thiết bị dùng vi xử lý 32 bit, hiệu năng trung bình:
- Có thể dùng hệ điều hành nhúng (RTOS, TinyOS, uCLinux,...)
- Giao tiếp vào ra cơ bản.
- Ghép nối ngoại vi theo các chuẩn truyền thông UART, I2C, SPI
- Có thêm khả năng kết nối Wifi, Bluetooth, GSM module
- Ví dụ: ESP8266, Arduino Uno Wifi

4. Lập trình cho thiết bị IoT

- Các thiết bị dùng vi xử lý 32/64 bit, hiệu năng cao:
- Có thể dùng hệ điều hành nhúng (Embedded Linux, Windows Embedded, Raspbian, Android, Ubuntu, …)
- Kết nối Wifi, Ethernet
- Giao tiếp ngoại vi theo cơ chế driver trên hệ điều hành.
- Ví dụ: Raspberry Pi, Jetson Nano, …

4.1 Lập trình Arduino

- Arduino Uno: Microchip ATmega328 không dùng hệ điều hành



4.1 Lập trình Arduino

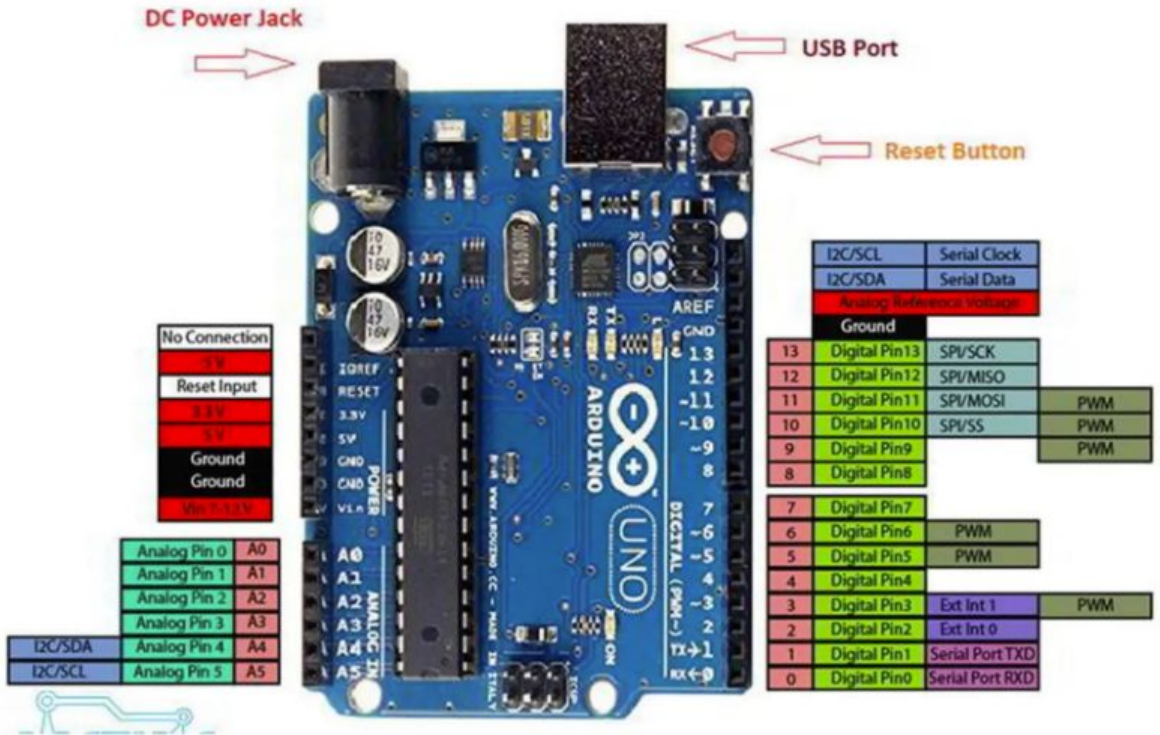
- Cài đặt Arduino IDE
  - https://www.arduino.cc/en/guide/windows
- Cấu trúc code:

```
void setup()
{
        //Used to indicate the initial values of system on starting.
}

void loop()
{
        Contains the statements that will run whenever the system
is powered after setup.
}
```

4.2 Giao tiếp ngoại vi

- Basic I/O
- UART
- I2C
- SPI
- ADC
- PWM



4.2 Giao tiếp ngoại vi

- **Giao tiếp vào ra cơ bản**
  - PIN configuration (cấu hình chân):

pinMode() Function Syntax

```
Void setup () {
  pinMode (pin , mode);
}
```

- **pin** – the number of the pin whose mode you wish to set
- **mode** – INPUT, OUTPUT, or INPUT_PULLUP.

4.2 Giao tiếp ngoại vi

- **Giao tiếp vào ra cơ bản**
  - Xuất tín hiệu ra các chân

digitalWrite() Function Syntax

```
Void loop() {
  digitalWrite (pin ,value);
}
```

- **pin** – the number of the pin whose mode you wish to set
- **value** – HIGH, or LOW.

Example

```
int LED = 6; // LED connected to pin 6

void setup () {
   pinMode(LED, OUTPUT); // set the digital pin as output
}

void setup () {
   digitalWrite(LED,HIGH); // turn on led
   delay(500); // delay for 500 ms
   digitalWrite(LED,LOW); // turn off led
   delay(500); // delay for 500 ms
}
```

4.2 Giao tiếp ngoại vi

```
const int buttonPin = 2;  // the number of the pushbutton pin
const int ledPin =  13;   // the number of the LED pin
int buttonState = 0;
void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(buttonPin, INPUT);
}
void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);
if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
```

4.2 Giao tiếp ngoại vi

- **Đọc tín hiệu từ chân Analog (ADC)**

analogRead() function Syntax

```
analogRead(pin);
```

- **pin** – the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

This function returns a number between 0 and 1023, which represents voltages between 0 and 5 volts.

Example

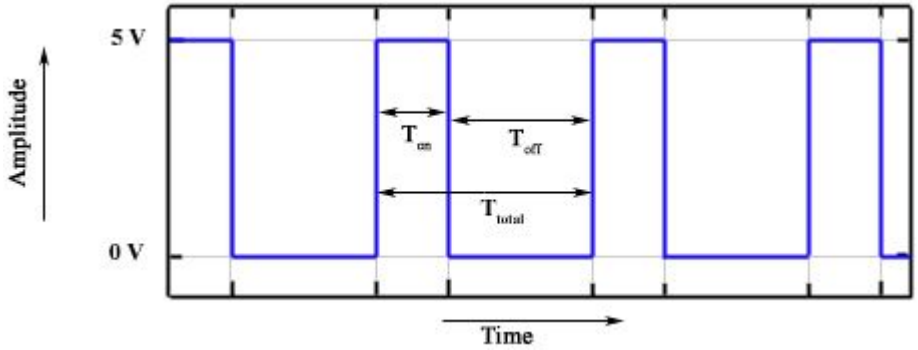
```
int analogPin = 3;//potentiometer wiper (middle terminal)
   // connected to analog pin 3
int val = 0; // variable to store the value read

void setup() {
   Serial.begin(9600); // setup serial
}

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

4.2 Giao tiếp ngoại vi

- **PWM:**



analogWrite() Function Syntax

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

value – the duty cycle: between 0 (always off) and 255 (always on).

- On-time:
- Off-time:
- Period:
- Duty cycle:

Example

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

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

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

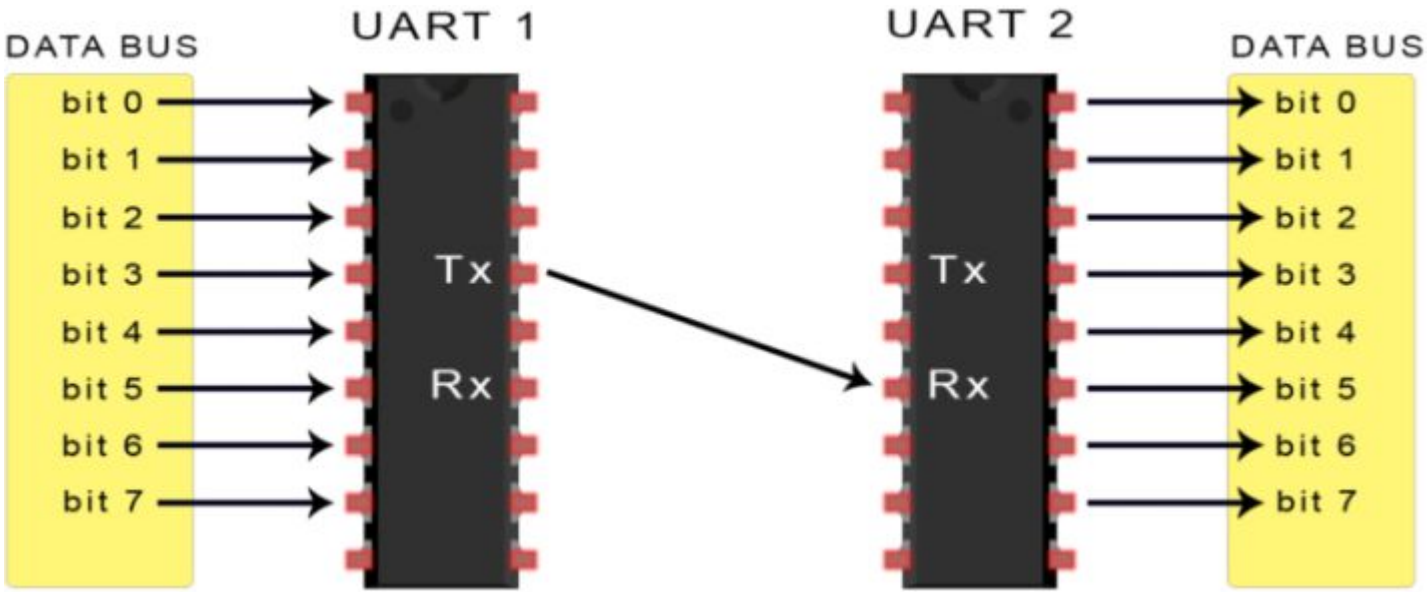
4.2 Giao tiếp ngoại vi

UART: Universal Asynchronous Receiver-Transmitter

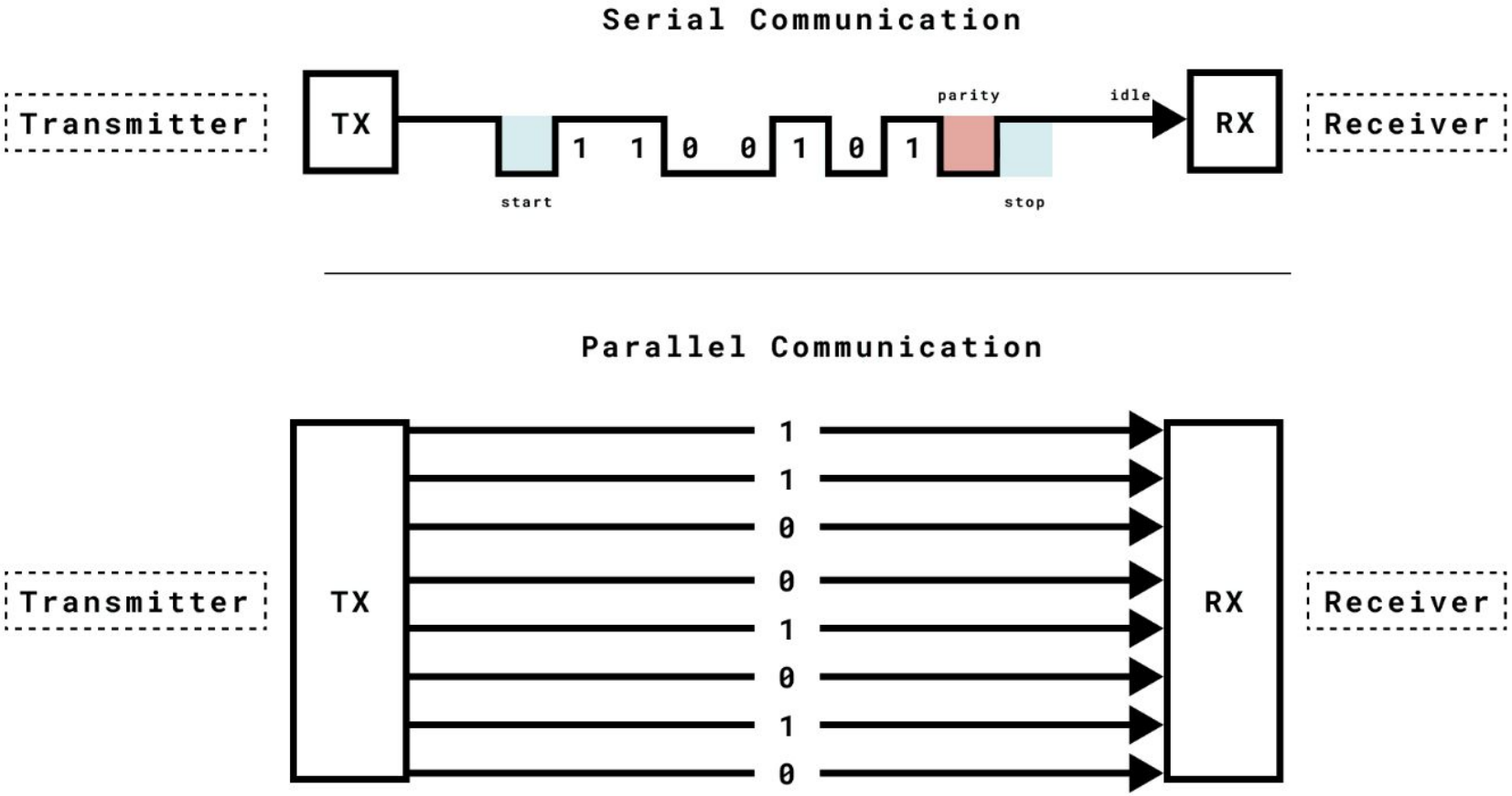
- Là một giao thức truyền thông nối tiếp.

- Sử dụng rộng rãi trong các hệ thống nhúng và thiết bị điện tử.

- Nó cho phép truyền và nhận dữ liệu giữa hai thiết bị mà không cần đồng hồ chung (asynchronous).



https://bkaii.com.vn/tin-tuc/1099-tim-hieu-ve-truyen-thong-uart-khai-niem-nguyen-ly-va-ung-dung



https://docs.arduino.cc/learn/communication/uart/

- Baud Rate: Tốc độ truyền tin (bits per second / bps)
  - 4800, 9600, 11520, 57600, …
- Start bit: logic level = LOW
- Data bits: LSE - MSE (6,7 or 8 bits)
- Parity: even/odd check
- Stop bit: logic level = HIGH

4.2 Giao tiếp ngoại vi

UART: Universal Asynchronous Receiver-Transmitter

- Cơ chế hoạt động của UART bao gồm việc truyền dữ liệu dưới dạng các bit theo thứ tự tuần tự từ bit ít quan trọng nhất (LSB) đến bit quan trọng nhất (MSB).

- Giao tiếp UART sử dụng hai đường chính: **Tx** (Transmit) để truyền dữ liệu và **Rx** (Receive) để nhận dữ liệu.

- Tốc độ truyền dữ liệu được đồng bộ bằng cách thiết lập trước tốc độ baud (baud rate) trên cả hai thiết bị.

4.2 Giao tiếp ngoại vi

UART: Universal Asynchronous Receiver-Transmitter

Một số đặc điểm của UART:

- Dễ sử dụng, không yêu cầu đồng hồ chung.
- Thường truyền dữ liệu ở tốc độ thấp và khoảng cách ngắn.
- Được sử dụng trong các ứng dụng như giao tiếp giữa các vi điều khiển và ngoại vi.

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

void setup() {
  Serial.begin(9600); //initialize serial communication at a 9600 baud rate
}

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);
  }
}
```

```
String sendMessage;
String receivedMessage;

void setup() {
  Serial.begin(9600);    // Initialize the Serial monitor for debugging
  Serial1.begin(9600);   // Initialize Serial1 for sending data
}

void loop() {
  while (Serial1.available() > 0) {
    char receivedChar = Serial1.read();
    if (receivedChar == '\n') {
      Serial.println(receivedMessage);  // Print the received message in the Serial
      receivedMessage = "";  // Reset the received message
    } else {
      receivedMessage += receivedChar;  // Append characters to the received message
    }
  }

  if (Serial.available() > 0) {
    char inputChar = Serial.read();
    if (inputChar == '\n') {
      Serial1.println(sendMessage);  // Send the message through Serial1 with a newl
      sendMessage = "";  // Reset the message
    } else {
      sendMessage += inputChar;  // Append characters to the message
    }
  }
}
```

4.2 Giao tiếp ngoại vi

SPI - Serial Peripheral Interface:

- Là một giao thức truyền thông nối tiếp đồng bộ, thường được sử dụng để kết nối vi điều khiển với các thiết bị ngoại vi.

- SPI cho phép truyền dữ liệu với tốc độ cao hơn so với UART và sử dụng đồng hồ chung để đồng bộ dữ liệu giữa các thiết bị.

4.2 Giao tiếp ngoại vi

SPI sử dụng 4 đường tín hiệu chính:

- **SCLK** (Serial Clock): Tín hiệu đồng hồ do master điều khiển.

- **MOSI** (Master Out Slave In): Đường truyền dữ liệu từ master đến slave.

- **MISO** (Master In Slave Out): Đường truyền dữ liệu từ slave đến master.

- **SS/CS** (Slave Select/Chip Select): Đường chọn thiết bị slave.

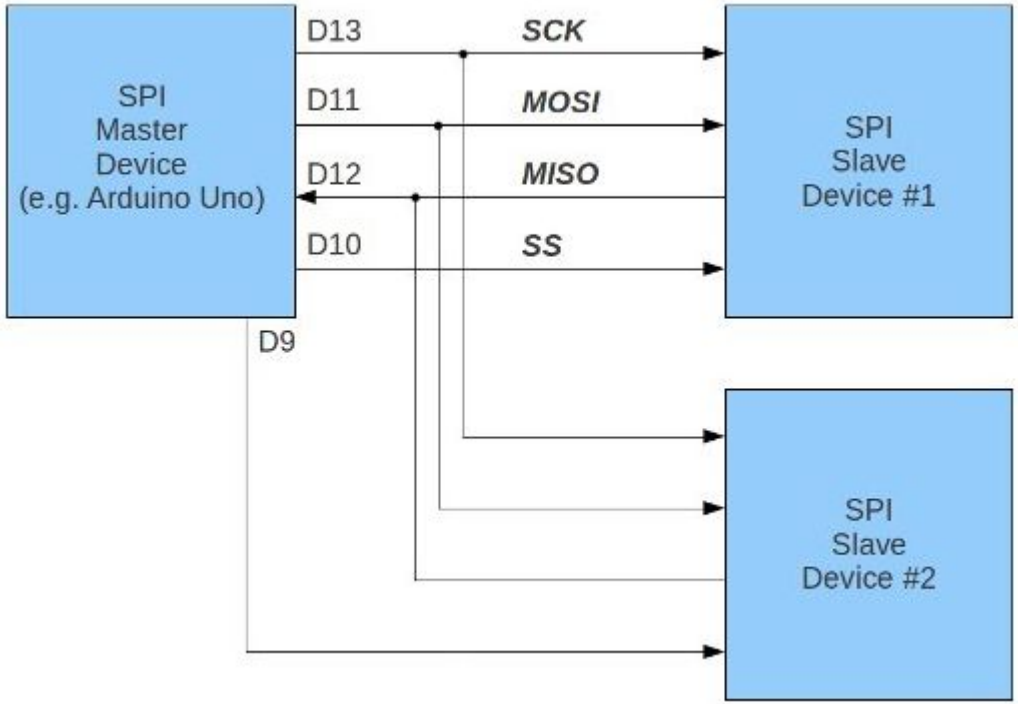
- Dữ liệu được truyền qua lại giữa 2 đường MISO và MOSI.

- Chân SS được thiết lập ở mức thấp LOW

http://arduino.vn/bai-viet/1081-arduino-va-giao-tiep-spi

- D9 và D10 là 2 chân SS

- Muốn giao tiếp với slave nào set chân đó ở mức LOW



http://arduino.vn/bai-viet/1081-arduino-va-giao-tiep-spi

- Sử dụng thư viện SPI:

```
#include "SPI.h"
```

- Khai báo chân SS:

```
pinMode(ss, OUTPUT);
```

- Kích hoạt giao tiếp SPI:

```
SPI.begin();
```

- Xác định cách để gửi dữ liệu, MSB hay LSB trước bằng cách sử dụng:

```
SPI.setBitOrder(MSBFIRST);
```

- Set chân SS xuống mức LOW, truyền dữ liệu, và set SS lên mức HIGH để dừng.

```
digitalWrite(SS, LOW);
SPI.transfer(value);
digitalWrite(ss, HIGH);
```

SPI.transfer()

Description

SPI transfer is based on a simultaneous send and receive: the received data is returned in receivedVal (or receivedVal16). In case of buffer transfers the received data is stored in the buffer in-place (the old data is replaced with the data received).

Syntax

```
receivedVal = SPI.transfer(val)

receivedVal16 = SPI.transfer16(val16)

SPI.transfer(buffer, size)
```

Parameters

- val: the byte to send out over the bus
- val16: the two bytes variable to send out over the bus
- buffer: the array of data to be transferred

Returns

The received data.

https://www.arduino.cc/reference/en/language/functions/communication/spi/transfer/

4.2 Giao tiếp ngoại vi

Ưu điểm của SPI:

- Không có bit bắt đầu và dừng, vì vậy dữ liệu có thể được truyền liên tục mà không bị gián đoạn.

- Không có hệ thống định địa chỉ slave phức tạp như I2C.

- Tốc độ truyền dữ liệu cao hơn I2C.

- Các đường MISO và MOSI riêng biệt, vì vậy dữ liệu có thể được gửi và nhận cùng một lúc.

4.2 Giao tiếp ngoại vi

Nhược điểm SPI:

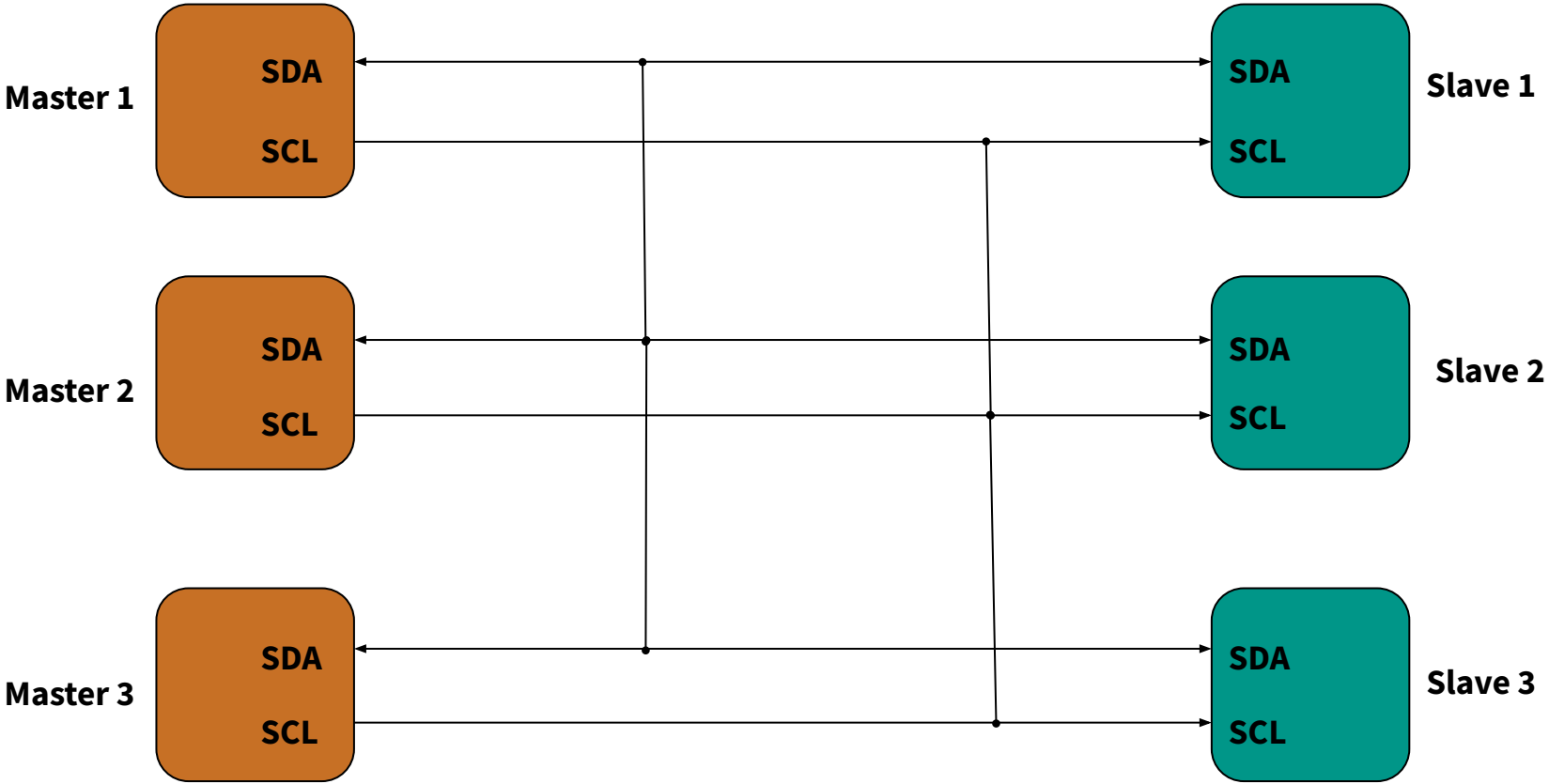
- Sử dụng bốn dây (I2C và UART sử dụng hai)
- Không xác nhận dữ liệu đã được nhận thành công (I2C có điều này)
- Không có hình thức kiểm tra lỗi như bit chẵn lẻ trong UART
- Chỉ cho phép một master duy nhất

4.2 Giao tiếp ngoại vi

I2C - Inter-Integrated Circuit:

- Là một giao thức truyền thông nối tiếp đồng bộ, được thiết kế để kết nối các vi điều khiển và các thiết bị ngoại vi với nhau trên cùng một bus.

- I2C cho phép nhiều thiết bị (cả master và slave) chia sẻ chung hai dây tín hiệu:
  - **SDA** (Serial Data): Đường truyền dữ liệu.
  - **SCL** (Serial Clock): Đường truyền tín hiệu đồng hồ do master điều khiển.


Master 1
SDA
SCL
Master 2
SDA
SCL
Master 3
SDA
SCL
SDA
SCL
Slave 1
SDA
SCL
Slave 2
SDA
SCL
Slave 3

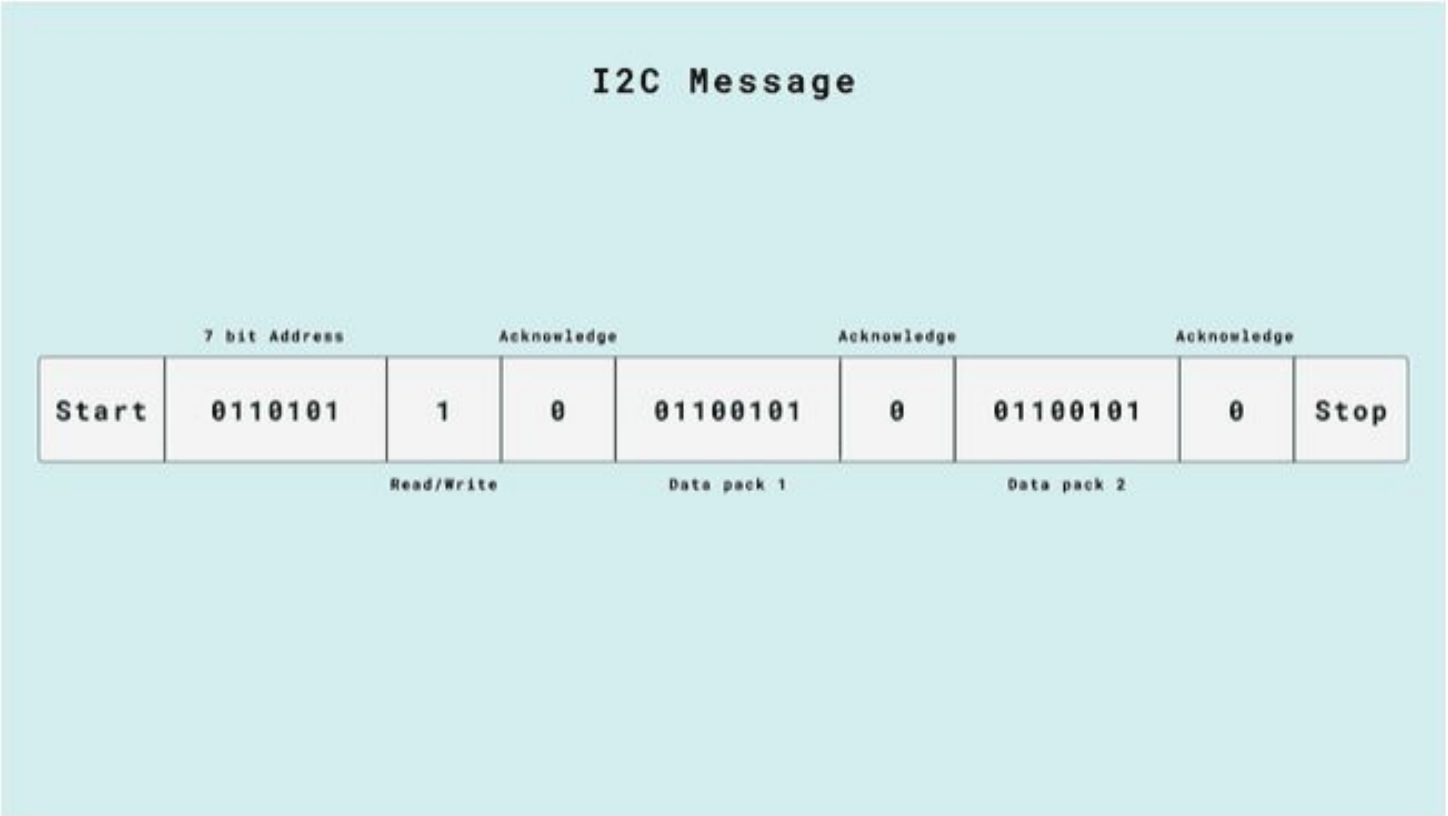
Cách thức hoạt động của I2C:

- Với I2C, dữ liệu được truyền trong các tin nhắn

- Tin nhắn được chia thành các khung dữ liệu.

- Mỗi tin nhắn có một khung địa chỉ chứa địa chỉ nhị phân của slave và một hoặc nhiều khung dữ liệu chứa dữ liệu đang được truyền.

- Thông điệp cũng bao gồm điều kiện khởi động và điều kiện dừng, các bit đọc / ghi và các bit ACK / NACK giữa mỗi khung dữ liệu



An I2C Message

https://docs.arduino.cc/learn/communication/wire/

- Start bit: LOW

- Stop bit: HIGH

- Định địa chỉ: Địa chỉ được xác định trong tin nhắn (7-10 bits).

- Bit đọc/Ghi:
  - Báo hiệu master muốn đọc hay ghi.
  - LOW: Master gửi dữ liệu
  - HIGH: Master yêu cầu dữ liệu

- Khung dữ liệu: 8bits (MSB FIRST)

- ACK/NACK: bit xác nhận

Wire Library

The Wire library is what Arduino uses to communicate with I2C devices. It is included in all board packages, so y
don't need to install it manually in order to use it.

To see the full API for the Wire library, visit its **documentation page**.

- `begin()` - Initialise the I2C bus
- `end()` - Close the I2C bus
- `requestFrom()` - Request bytes from a peripheral device
- `beginTransmission()` - Begins queueing up a transmission
- `endTransmission()` - Transmit the bytes that have been queued and end the transmission
- `write()` - Writes data from peripheral to controller or vice versa
- `available()` - returns the number of bytes available for retrieval
- `read()` - Reads a byte that was transmitted from a peripheral to a controller.
- `setClock()` - Modify the clock frequency
- `onReceive()` - Register a function to be called when a peripheral receives a transmission
- `onRequest()` - Register a function to be called when a controller requests data
- `setWireTimeout()` - Sets the timeout for transmissions in controller mode
- `clearWireTimeoutFlag()` - Clears the timeout flag
- `getWireTimeoutFlag()` - Checks whether a timeout has occurred since the last time the flag was cleared.

```
// Wire Controller Reader
// by Nicholas Zambetti [http://www.zambetti.com](http://www.zambetti.com)

// Demonstrates use of the Wire library
// Reads data from an I2C/TWI peripheral device
// Refer to the "Wire Peripheral Sender" example for use with this

// Created 29 March 2006

// This example code is in the public domain.


#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(8, 6);    // request 6 bytes from peripheral device #8

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

  delay(500);
}
```

Ưu điểm của I2C:

- Chỉ sử dụng hai dây
- Hỗ trợ nhiều master và nhiều slave
- Bit ACK / NACK xác nhận mỗi khung được chuyển thành công
- Phần cứng ít phức tạp hơn so với UART
- Giao thức nổi tiếng và được sử dụng rộng rãi

Nhược điểm của I2C:

- Tốc độ truyền dữ liệu chậm hơn SPI

- Kích thước của khung dữ liệu bị giới hạn ở 8 bit

- Cần phần cứng phức tạp hơn để triển khai so với SPI

4.2 Giao Tiếp Ngoại Vi

Cơ Chế Ngắt - Interrupts:

- Ngắt là một cơ chế cho phép vi điều khiển dừng công việc hiện tại để thực hiện một công việc khác.

- Chương trình chính đang chạy và thực hiện một chức năng trong mạch. Tuy nhiên, khi có một ngắt xảy ra, chương trình chính tạm dừng trong khi một quy trình khác được thực hiện. Khi quy trình này hoàn tất, bộ xử lý lại quay trở lại chương trình chính.

- Ngắt có thể đến từ nhiều nguồn khác nhau.

- Hầu hết các thiết kế của Arduino có hai ngắt phần cứng (được gọi là "interrupt0" và "interrupt1") được gán cứng với các chân digital I/O 2 và 3, tương ứng.

- Có thể định nghĩa một quy trình bằng cách sử dụng một hàm đặc biệt gọi là “Interrupt Service Routine” (thường được gọi là ISR).

- Có thể định nghĩa quy trình và chỉ định điều kiện xảy ra ngắt ở cạnh lên, cạnh xuống hoặc cả hai. Tại các điều kiện cụ thể này, ngắt sẽ được xử lý.

- Có thể để hàm này tự động thực thi mỗi khi một sự kiện xảy ra trên chân đầu vào.

Có hai loại ngắt:

- Ngắt phần cứng – Chúng xảy ra để phản ứng với một sự kiện bên ngoài, chẳng hạn như khi một chân ngắt bên ngoài chuyển từ mức cao sang thấp hoặc ngược lại.

- Ngắt phần mềm – Chúng xảy ra để phản ứng với một lệnh được gửi trong phần mềm. Loại ngắt duy nhất mà "ngôn ngữ Arduino" hỗ trợ là hàm **attachInterrupt()**.

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

  - pin : Chân được gán ngắt.
  - ISR: hàm xử lý ngắt - không nhận tham số truyền vào, không trả lại giá trị
  - mode: điều kiện ngắt
    - LOW, CHANGE, RAISING, FALLING

```
int pin = 2; //define interrupt pin to 2
volatile int state = LOW; // To make sure variables shared between an ISR
//the main program are updated correctly,declare them as volatile.

void setup() {
   pinMode(13, OUTPUT); //set pin 13 as output
   attachInterrupt(digitalPinToInterrupt(pin), blink, CHANGE);
   //interrupt at pin 2 blink ISR when pin to change the value
}
void loop() {
   digitalWrite(13, state); //pin 13 equal the state value
}

void blink() {
   //ISR function
   state = !state; //toggle the state when the interrupt occurs
}
```

```
// Wire Peripheral Sender
// by Nicholas Zambetti [http://www.zambetti.com](http://www.zambetti.com)

// Demonstrates use of the Wire library
// Sends data as an I2C/TWI peripheral device
// Refer to the "Wire Master Reader" example for use with this

// Created 29 March 2006

// This example code is in the public domain.


#include <Wire.h>

void setup() {
  Wire.begin(8);                // join i2c bus with address #8
  Wire.onRequest(requestEvent); // register event
}

void loop() {
  delay(100);
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent() {
  Wire.write("hello "); // respond with message of 6 bytes
  // as expected by master
}
```

```
// Wire Master Writer
// by Nicholas Zambetti [http://www.zambetti.com](http://www.zambetti.com)

// Demonstrates use of the Wire library
// Writes data to an I2C/TWI Peripheral device
// Refer to the "Wire Peripheral Receiver" example for use w

// Created 29 March 2006

// This example code is in the public domain.


#include <Wire.h>

void setup()
{
  Wire.begin(); // join i2c bus (address optional for maste
}

byte x = 0;

void loop()
{
  Wire.beginTransmission(4); // transmit to device #4
  Wire.write("x is ");        // sends five bytes
  Wire.write(x);              // sends one byte
  Wire.endTransmission();    // stop transmitting

  x++;
  delay(500);
}
```

```
// This example code is in the public domain.


#include <Wire.h>

void setup()
{
  Wire.begin(4);                // join i2c bus with address #4
  Wire.onReceive(receiveEvent); // register event
  Serial.begin(9600);           // start serial for output
}

void loop()
{
  delay(100);
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
  while(1 < Wire.available()) // loop through all but the last
  {
    char c = Wire.read(); // receive byte as a character
    Serial.print(c);         // print the character
  }
  int x = Wire.read();    // receive byte as an integer
  Serial.println(x);         // print the integer
}
```

```ino
#include <Wire.h>
#include <Adafruit_BMP280.h>

//Create an instance of the BMP280 sensor
Adafruit_BMP280 bmp;

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

  // Start the sensor, and verify that it was found
  if (!bmp.begin()) {
    Serial.println("Sensor not found");
    while (1){}
  }

}

void loop() {
  // Read the values
  float temperature = bmp.readTemperature();

  // Print to the Serial Monitor
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" C");

  Serial.println();
  delay(2000);
}
```