CÁC CHUẨN GIAO TIẾP

1. GPIO (General Purpose Input/Output)

- **Nguyên lý**: GPIO là các chân I/O có thể cấu hình linh hoạt giữa input và output. Điều này giúp vi điều khiển giao tiếp với các thiết bị ngoại vi như công tắc, đèn LED, cảm biến, …
- **Quá trình hoạt động**:
    - **Input**: Đọc trạng thái từ các thiết bị ngoại vi, như công tắc hoặc cảm biến, thông qua lệnh digitalRead(pin).
    - **Output**: Cung cấp tín hiệu để điều khiển các thiết bị ngoại vi, ví dụ như bật/tắt LED bằng digitalWrite(pin, HIGH) hoặc digitalWrite(pin, LOW).

Cú pháp cấu hình chân: pinMode(pin, mode); // mode có thể là INPUT hoặc OUTPUT

Cú pháp đọc và ghi tín hiệu:
digitalWrite(pin, value); // Ghi HIGH hoặc LOW (OUTPUT)
digitalRead(pin); // Đọc HIGH hoặc LOW (INPUT)

2. PWM (Pulse Width Modulation)

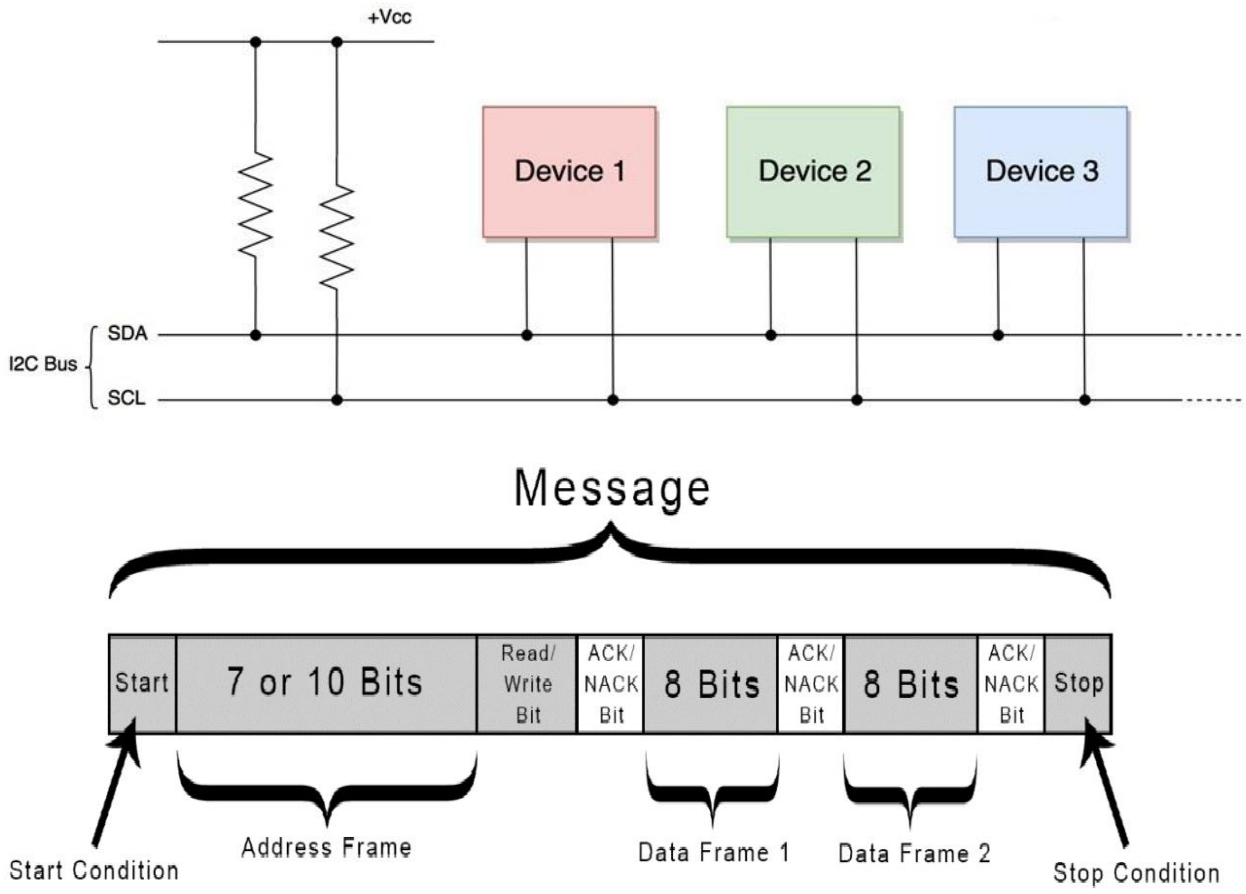
- **Nguyên lý**: PWM tạo ra một tín hiệu vuông với chu kỳ ON/OFF liên tục, và độ rộng xung (duty cycle) điều khiển mức điện áp trung bình mà thiết bị nhận được. PWM thường dùng để điều khiển tốc độ động cơ, độ sáng đèn LED, âm lượng buzzer, v.v.
- **Quá trình hoạt động**:
    - **Chuỗi xung vuông**: Tạo một chuỗi xung với thời gian HIGH và LOW thay đổi.
    - **Thay đổi duty cycle**: Thay đổi tỷ lệ thời gian HIGH so với tổng chu kỳ để điều chỉnh năng lượng trung bình.
    - **Điều khiển thiết bị**: Thiết bị như động cơ hoặc LED sẽ phản ứng với mức năng lượng mà nó nhận được từ tín hiệu PWM.

Cú pháp cấu hình PWM:

```
analogWrite(pin, value);  // value từ 0 đến 255
```

3. I2C (Inter-Integrated Circuit)

- **Nguyên lý**: I2C là một giao thức giao tiếp chuỗi đồng bộ sử dụng hai dây (SDA - dữ liệu, SCL - xung nhịp). Một thiết bị Master điều khiển một hoặc nhiều thiết bị Slave. Dữ liệu được truyền theo byte và mỗi byte đều được xác nhận bằng tín hiệu ACK từ thiết bị nhận.



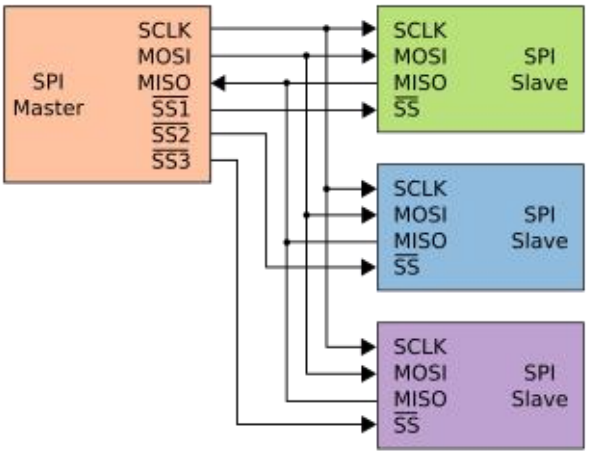
- **Quá trình hoạt động**:

    1. Master gửi tín hiệu START.
    2. Master gửi địa chỉ thiết bị Slave và tín hiệu đọc/ghi.
    3. Nếu Slave nhận đúng địa chỉ, nó phản hồi bằng tín hiệu ACK.

4. Dữ liệu được truyền qua SDA, đồng bộ với tín hiệu xung nhịp từ SCL.
5. Giao tiếp kết thúc bằng tín hiệu STOP.

4. SPI (Serial Peripheral Interface)

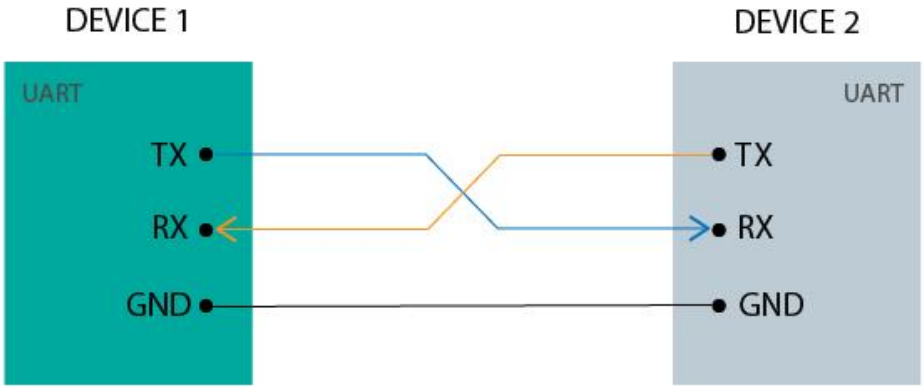
- **Nguyên lý**: SPI là giao thức giao tiếp đồng bộ tốc độ cao, sử dụng 4 chân: MOSI (Master Out Slave In), MISO (Master In Slave Out), SCLK (Clock), và SS (Slave Select). Dữ liệu được truyền song song với tốc độ cao, mỗi lần gửi là 1 byte.



- **Quá trình hoạt động**:
    1. Master kéo chân CS xuống LOW để chọn thiết bị Slave.
    2. Dữ liệu được truyền qua MOSI, đồng bộ với xung nhịp từ SCLK.
    3. Nếu cần thiết, Slave có thể trả dữ liệu về qua MISO.
    4. Sau khi truyền xong, Master kéo chân SS lên HIGH để kết thúc giao tiếp.

5. UART (Universal Asynchronous Receiver/Transmitter)

• **Chức năng**: UART là **giao tiếp nối tiếp không đồng bộ**, dùng để truyền dữ liệu giữa 2 thiết bị (thường là vi điều khiển và module ngoài như máy tính, module Bluetooth, GPS,...).



• **Đặc điểm nổi bật**:

- Chỉ cần **2 dây**:
  - **TX** (Transmit): chân truyền dữ liệu.
  - **RX** (Receive): chân nhận dữ liệu.
- **Không cần clock** chung (asynchronous).
- Truyền theo dạng byte, với cấu trúc gồm:
  Start bit → Data bits → Optional Parity bit → Stop bit
- **Chỉ 1 đối một**: mỗi UART chỉ nói chuyện được với **1 thiết bị** tại một thời điểm.

CÁC LINH KIỆN THƯỜNG GẶP

1. LED (Light Emitting Diode)

- **Chức năng**: LED là một loại diode phát quang, có thể phát sáng khi dòng điện chạy qua. Thường được sử dụng trong các ứng dụng như báo hiệu, chiếu sáng, hoặc chỉ thị trạng thái.
- **Giao thức sử dụng**:
    - **GPIO** (General Purpose Input/Output): Để điều khiển LED, bạn sẽ thường sử dụng chân GPIO của vi điều khiển để bật hoặc tắt LED (chế độ Output).

Cú pháp:

digitalWrite(pin, HIGH) để bật LED

digitalWrite(pin, LOW) để tắt LED.

    - **PWM (Pulse Width Modulation)**: Nếu bạn cần điều chỉnh độ sáng của LED (thay đổi độ sáng theo mức trung bình), bạn có thể sử dụng PWM. PWM giúp điều khiển độ sáng của LED bằng cách thay đổi **duty cycle**.

Cú pháp:

analogWrite(pin, value) (với value từ 0 đến 255).

- **Ví dụ**:

```
int ledPinGPIO = 13;    // Chân LED sử dụng GPIO (bật/tắt)
int ledPinPWM = 9;      // Chân LED sử dụng PWM (điều chỉnh
độ sáng)

void setup() {
pinMode(ledPinGPIO, OUTPUT);  // Cấu hình chân GPIO cho
LED là output
pinMode(ledPinPWM, OUTPUT);   // Cấu hình chân PWM cho
LED là output
}

void loop() {
// Bật LED bằng GPIO
digitalWrite(ledPinGPIO, HIGH);  // Bật LED sử dụng GPIO
delay(1000);                     // Chờ 1 giây

// Tắt LED bằng GPIO
digitalWrite(ledPinGPIO, LOW);   // Tắt LED sử dụng GPIO
delay(1000);                     // Chờ 1 giây

// Điều chỉnh độ sáng LED với PWM
for (int brightness = 0; brightness <= 255; brightness++) {
```

```
    analogWrite(ledPinPWM, brightness);   // Điều chỉnh độ sáng
    LED (PWM)
    delay(10);                    // Chờ 10ms để thay đổi mượt mà
    }

     // Giảm độ sáng LED với PWM
     for (int brightness = 255; brightness >= 0; brightness--) {
        analogWrite(ledPinPWM, brightness);  // Điều chỉnh độ sáng
    LED (PWM)
       delay(10);                    // Chờ 10ms để thay đổi mượt mà
     }
    }
```

2. Button (Nút bấm)

- **Chức năng**: Nút bấm là một linh kiện đầu vào, dùng để nhận tín hiệu từ người dùng (bấm nút) và chuyển tín hiệu này đến vi điều khiển.
- **Giao thức sử dụng**:
    - **GPIO**: Nút bấm thường được kết nối với chân GPIO của vi điều khiển, cấu hình như **Input**. Khi nút được bấm, tín hiệu sẽ thay đổi giữa HIGH và LOW.

Cú pháp:

digitalRead(pin) để đọc trạng thái của nút (HIGH nếu bấm, LOW nếu không bấm).

- **Pull-up/Pull-down Resistor**: Để đảm bảo tín hiệu ổn định, bạn có thể sử dụng **Pull-up** hoặc **Pull-down resistor**.

Cú pháp:

pinMode(pin, INPUT_PULLUP) nếu sử dụng resistor kéo lên (pull-up).

- **Ví dụ**:

```
int buttonPin = 2;  // Địa chỉ chân button
int ledPin = 13;    // Địa chỉ chân LED


void setup() {
```

```
    pinMode(buttonPin, INPUT_PULLUP);   // Chân button với
pull-up resistor
  pinMode(ledPin, OUTPUT);          // Chân LED là output
}

void loop() {
   if (digitalRead(buttonPin) == LOW) {   // Nếu nút được bấm
(LOW)
    digitalWrite(ledPin, HIGH);  // Bật LED
  } else {
    digitalWrite(ledPin, LOW);   // Tắt LED
  }
}
```

3. LCD (Liquid Crystal Display)

- **Chức năng**: LCD là màn hình hiển thị dùng để hiển thị thông tin dưới dạng chữ hoặc hình ảnh. Chúng có thể có nhiều kích thước khác nhau và sử dụng để hiển thị các thông tin, trạng thái từ vi điều khiển.
- **Giao thức sử dụng**:
  - **I2C**: Màn hình LCD có thể giao tiếp qua giao thức I2C, giúp tiết kiệm số lượng chân kết nối (chỉ cần 2 chân: SDA và SCL).
  - **SPI**: Một số màn hình LCD có thể giao tiếp qua SPI.
  - **Parallel**: Các màn hình LCD thông thường (như 16x2) thường sử dụng giao thức parallel, với nhiều chân dữ liệu (8 hoặc 4 chân) để truyền thông.
- **Ví dụ (I2C)**:

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2); // I2C address 0x27, 16 column and 2 rows

void setup() {
  lcd.init(); // initialize the lcd
```

```
  lcd.backlight();
}

void loop() {
  lcd.clear();            // clear display
  lcd.setCursor(6, 0);     // move cursor to   (0, 0)
  lcd.print("IOT");       // print message at (0, 0)
  lcd.setCursor(5, 1);     // move cursor to   (2, 1)
  lcd.print("EC Dept"); // print message at (2, 1)
  delay(2000);              // display the above for two seconds

  lcd.clear();              // clear display
  lcd.setCursor(2, 0);         // move cursor to   (3, 0)
  lcd.print("LCD I2C MODE");       // print message at (3, 0)
  lcd.setCursor(6, 1);         // move cursor to   (0, 1)
  lcd.print("16x2"); // print message at (0, 1)
  delay(2000);                  // display the above for two
seconds
}
```

4. RTC (Real-Time Clock)

- **Chức năng**: RTC là một đồng hồ thời gian thực, dùng để theo dõi thời gian thực ngay cả khi vi điều khiển không được cấp nguồn. Thường được sử dụng trong các hệ thống yêu cầu giữ đồng hồ chính xác khi không có nguồn điện chính (như trong các thiết bị không có nguồn điện liên tục).
- **Giao thức sử dụng**:
  - **I2C**: Hầu hết các module RTC sử dụng giao thức I2C để giao tiếp với vi điều khiển (ví dụ: DS3231, DS1307).
  - **SPI**: Một số module RTC sử dụng giao thức SPI, nhưng I2C là phổ biến hơn.
- **Ví dụ (I2C)**:

```
#include <Wire.h>
#include <RTClib.h>

RTC_DS1307 rtc;

void setup() {
  Serial.begin(115200);

  // Initialize I2C communication
```

```
  Wire.begin();

  // Check if the RTC is connected properly
  if (!rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }

  // Check if the RTC lost power and if so, set the time
  if (!rtc.isrunning()) {
    Serial.println("RTC is NOT running!");
    // This line sets the RTC with an explicit date & time, for
example to set
    // January 1, 2024 at 00:00 you would call:
    rtc.adjust(DateTime(2024, 1, 1, 0, 0, 0));
  }
}

void loop() {
  DateTime now = rtc.now();

  // Print the current date and time to the serial monitor
  Serial.print(now.year(), DEC);
```

```
    Serial.print('/');
    Serial.print(now.month(), DEC);
    Serial.print('/');
    Serial.print(now.day(), DEC);
    Serial.print(" ");
    Serial.print(now.hour(), DEC);
    Serial.print(':');
    Serial.print(now.minute(), DEC);
    Serial.print(':');
    Serial.print(now.second(), DEC);
    Serial.println();

    delay(1000);  // Wait for 1 second
}
```

5.DHT22 (Digital Humidity and Temperature Sensor)

- **Chức năng**: DHT22 là cảm biến kỹ thuật số dùng để đo **nhiệt độ** và **độ ẩm** của môi trường xung quanh. Nó có độ chính xác cao hơn DHT11 và thích hợp cho các ứng dụng yêu cầu đo đạc chính xác môi trường.
- **Giao thức sử dụng**:**1-Wire (phi chuẩn)**: DHT22 sử dụng một giao thức truyền thông đặc biệt tương tự 1-Wire, chỉ cần một chân dữ liệu duy nhất để gửi tín hiệu từ cảm biến đến vi điều khiển.

  → Việc giao tiếp yêu cầu sử dụng thư viện (như DHT.h) để xử lý đúng định dạng tín hiệu.

  → Dữ liệu bao gồm: độ ẩm cao/thấp, nhiệt độ cao/thấp, và byte kiểm tra CRC.

- **Ví dụ:**

```
#include <DHT.h>
// Set up the DHT sensor
DHT dht(4, DHT22);
float temperature ;
float  humidity;
int counter;


void setup() {
  // put your setup code here, to run once:
```

```
  Serial.begin(9600);
  Serial.println("Hello, ESP32!");
}

void loop() {
  // put your main code here, to run repeatedly:
  temperature  = dht.readTemperature();
  humidity     = dht.readHumidity();
  Serial.println("Data: "+ String(counter));
  // Print the values of temperature in Celsus
  Serial.print("Temperatue:\t");
  Serial.print(dht.readTemperature(false));
  Serial.println("C");
   // Print the values of temperature in Fahrenheit
  Serial.print("Temperatue:\t");
  Serial.print(dht.readTemperature(true));
  Serial.println("F");
  // print Humidity in perscent
  Serial.println("Humidity: \t"+String(humidity)+ "%");
  // Print the values of the heat Index for both Units
  Serial.print("Heat Index In Celsus: ");
     Serial.println(dht.computeHeatIndex(temperature,   humidity,
false));
```

```
  Serial.print("Heat Index In Fahrenheit: ");
     Serial.println(dht.computeHeatIndex(temperature,  humidity,
true));
  Serial.println(" ");
  delay(10000); // this speeds up the simulation
  counter++;
}
```

6.Buzzer (Loa phát âm thanh loại điều khiển bằng tần số)

- **Chức năng**: Buzzer là một thiết bị phát ra âm thanh, thường được dùng để cảnh báo hoặc báo hiệu trạng thái. Loại điều khiển bằng tần số có thể phát ra âm cao/thấp khác nhau tùy theo tần số cung cấp, tạo nên các giai điệu hoặc tín hiệu âm thanh.
- **Giao thức sử dụng**: **PWM (tần số)**: Điều khiển buzzer bằng cách phát xung vuông với tần số xác định (thường dùng hàm tone() trong Arduino).

  → tone(pin, frequency) để phát âm thanh liên tục với tần số nhất định.

  → tone(pin, frequency, duration) để phát trong một khoảng thời gian.

  → noTone(pin) để dừng âm thanh.

- **Ví dụ:** Dùng kèm với thư viện pitches.h

```
#include "pitches.h"

#define SPEAKER_PIN 8

// Define the note frequencies for "Happy Birthday"
int melody[] = {
  NOTE_C4, NOTE_C4, NOTE_D4, NOTE_C4, NOTE_F4, NOTE_E4,
```

```
    NOTE_C4, NOTE_C4, NOTE_D4, NOTE_C4, NOTE_G4, NOTE_F4,
    NOTE_C4, NOTE_C4, NOTE_C5, NOTE_A4, NOTE_F4, NOTE_E4, NOTE_D4,
   NOTE_AS4, NOTE_AS4, NOTE_A4, NOTE_F4, NOTE_G4, NOTE_F4
};

int noteDurations[] = {
  4, 4, 4, 4, 4, 2,
  4, 4, 4, 4, 4, 2,
  4, 4, 4, 4, 4, 4, 2,
  4, 4, 4, 4, 4, 2
};

void setup() {
  Serial.begin(115200);
  Serial.println("Hello, ESP32-S3!");

}

void loop() {
  // Iterate through the melody
```

```
  for (int i = 0; i < sizeof(melody) / sizeof(melody[0]); i++) {
    int noteDuration = 1000 / noteDurations[i];
    tone(8, melody[i], noteDuration);

    // Pause between notes
    delay(noteDuration * 1.30);

    // Stop the tone
    noTone(8);
  }

  // Repeat the melody
  delay(2000);
}
```