## I2C

Dùng để kết nối các mạch tích hợp (IC), giao tiếp 1 master với nhiều slave chỉ cần 2 dây:  
SCL và SDA

- **Master:** Thiết bị khởi tạo giao tiếp, tạo ra xung đồng hồ (SCL) và điều khiển quá trình truyền dữ liệu. Trong hệ thống I2C, dữ liệu luôn được truyền từ master tới slave.

- **Slave:** Thiết bị nhận địa chỉ từ master, phản hồi theo yêu cầu. Mỗi slave có một địa chỉ riêng, giúp master nhận diện.

- **Transmitter & Receiver:**

* **Transmitter:** Thiết bị gửi dữ liệu lên bus.
* **Receiver:** Thiết bị nhận dữ liệu từ bus.

- **Arbitration (Đấu Giá):** Quá trình xác định master “điều khiển” bus khi có nhiều master cùng cố gắng giao tiếp đồng thời. I2C thực hiện việc này thông qua phần cứng, đảm bảo chỉ một master duy nhất chiếm quyền giao tiếp mà không làm hỏng dữ liệu.

- **Clock Stretching:** Kỹ thuật cho phép slave kéo đường SCL xuống mức thấp để báo hiệu với master rằng nó cần thêm thời gian xử lý dữ liệu.

* **SCL và SDA** là hai đường tín hiệu song công được kéo lên về mức high thông qua điện trở kéo lên.
* **Chế độ open-drain** đảm bảo rằng mỗi thiết bị chỉ có thể kéo tín hiệu xuống khi cần, giúp tránh xung đột khi nhiều thiết bị cùng kết nối.
* **Việc cấu hình đúng và kiểm tra điện áp** trên các chân này là bước quan trọng để đảm bảo giao tiếp I2C hoạt động ổn định.

## MODE

* **Standard Mode** thích hợp với các thiết bị cơ bản, không yêu cầu tốc độ cao.
* **Fast Mode** cung cấp tốc độ nhanh hơn và tương thích với thiết bị Standard Mode (về cơ chế downward compatible), nhưng không thể kết hợp ngược lại.
* **Fast Mode Plus** và **High Speed Mode** đòi hỏi phần cứng hỗ trợ chuyên biệt và thường dùng cho các ứng dụng yêu cầu tốc độ truyền cao.

QUY TRÌNH TRUYỀN DỮ LIỆU TRONG I2C

* Bước 1: Start condition
  + Start condition
  + Được master phát ra trên đường SDA (trong khi SCL đang ở mức high)
  + Đây là tín hiệu báo hiệu bắt đầu một phiên giao tiếp
  + Khi start được tạo ra, bus I2C được chiếm bởi master, và không thiết bị nào khác được phép truyền cho đến khi có STOP
* Bước 2: Address Phase
  + Cấu trúc 8 bit
    - **7 bit đầu tiên:** Là địa chỉ của slave mà master muốn giao tiếp.
    - **Bit thứ 8 (R/W bit):** Xác định chế độ giao tiếp:
      * Nếu **0**: Master sẽ gửi (write) dữ liệu đến slave.
      * Nếu **1**: Master sẽ đọc (read) dữ liệu từ slave.
* Bước 3: ACK sau mỗi byte
  + Mỗi byte truyền đi phải được theo sau bởi 1 bit ACK
    - ACK: Slave sẽ gửi bit ACK (thường kéo SDA xuống mức low) nếu đã nhận được byte dữ liệu (bao gồm cả byte địa chỉ) một cách chính xác.
  + Quy trình:
    - Sau khi master gửi xong byte địa chỉ, slave so sánh địa chỉ nhận được với địa chỉ của mình
    - Nếu khớp, slave gửi ACK để báo hiệu “ tôi nhận được và sẵn sàng giao tiếp”
* Bước 4: Data Phase
  + Trường hợp Write (R/W = 0):
    - Master**:** Gửi 1 byte dữ liệu sau byte địa chỉ đã được ACK.
    - **Slave:** Sau khi nhận byte dữ liệu, gửi ACK để xác nhận dữ liệu đã được nhận thành công.
  + Trường hợp Read (R/W = 1):
    - **Slave:** Sau khi nhận được địa chỉ và ACK, slave sẽ truyền 1 byte dữ liệu cho master.
    - **Master:** Sau khi nhận được byte dữ liệu, master gửi ACK để báo rằng dữ liệu đã được nhận.
    - Nếu master nhận đủ dữ liệu cần thiết, nó sẽ không gửi ACK cho byte cuối cùng (hoặc gửi NACK) để báo hiệu dừng nhận, rồi sau đó phát ra STOP condition.
* Bước 5: Stop condition
  + Stop condition
    - Được master tạo ra để kết thúc phiên giao tiếp.

Cách master đọc dữ liệu từ EEPROM mà không dùng Repeated Start:

1. Master tạo **Start** và gửi địa chỉ slave với R/W = 0 (ghi).
2. Master gửi địa chỉ 0x45 tới EEPROM.
3. Thay vì Stop, master tạo **Repeated Start**.
4. Master gửi lại địa chỉ slave với R/W = 1 (đọc).
5. EEPROM gửi dữ liệu từ 0x45.
6. Master kết thúc bằng **Stop**.

* Quá trình này bao gồm hai giao dịch riêng biệt: **Ghi (write)** địa chỉ (0x45) và **Đọc (read)** dữ liệu.
* Điểm yếu: Khi master tạo Stop sau lần ghi đầu tiên, nó nhả bus, cho phép các master khác chiếm quyền điều khiển bus trong hệ thống đa master. Điều này có thể làm gián đoạn giao tiếp trước khi master hoàn thành mục tiêu đọc dữ liệu.

## Thanh ghi dữ liệu và giao tiếp Half-Duplex

* **Half-Duplex:** Chỉ truyền hoặc nhận dữ liệu tại một thời điểm (không đồng thời). Vì vậy, I2C chỉ cần một thanh ghi dữ liệu thay vì hai buffer riêng như full-duplex (ví dụ: UART).
* **Data Register:** Thanh ghi mà phần mềm (firmware) tương tác trực tiếp để ghi hoặc đọc dữ liệu.
* **Shift Register:** Thanh ghi dịch từng bit dữ liệu ra SDA theo nhịp SCL khi truyền, hoặc nhận từng bit khi đọc.
* Quy trình:
* **Truyền:** Firmware ghi byte vào data register → sao chép sang shift register → truyền qua SDA.
* **Nhận:** Dữ liệu từ SDA vào shift register → khi đủ 1 byte, sao chép sang data register → firmware đọc.

## Chế độ Master và Slave

* **Nội dung:** Ngoại vi I2C có thể hoạt động ở chế độ master hoặc slave. Khi ở chế độ slave, địa chỉ của slave được lưu trong thanh ghi “Own Address Register”.
* **Giải thích:**
* **Master Mode:** I2C điều khiển bus, tạo tín hiệu SCL và giao tiếp với slave.
* **Slave Mode:** I2C nhận lệnh từ master khác và phản hồi dựa trên địa chỉ của nó.
* **Own Address Register:** Thanh ghi lưu địa chỉ duy nhất của thiết bị khi ở chế độ slave (ví dụ: 0x45 trong bài trước). Master dùng địa chỉ này để liên lạc với slave cụ thể.

## Viết driver

**YÊU CẦU API CHO DRIVER I2C**

* **API:** Giao diện lập trình ứng dụng, là các hàm mà ứng dụng người dùng gọi để điều khiển I2C.
* **Master TX/RX:** Master khởi tạo giao tiếp (Start, địa chỉ, Stop), nên cần API riêng để truyền dữ liệu (Master TX) và nhận dữ liệu (Master RX).
* **Slave TX/RX:** Slave chỉ phản hồi khi được Master gọi, nên cũng cần API riêng.
* **Interrupt Handling:** Xử lý ngắt giúp giao tiếp không chặn (non-blocking), sẽ được giải thích sau

**CÁC MỤC CONFIG:**

1. **SCL Speed:** Quyết định tốc độ giao tiếp, lưu trong thanh ghi CCR (Clock Control Register).
2. **Device Address:** Địa chỉ Slave (ví dụ: 0x45), lưu trong thanh ghi OAR (Own Address Register).
3. **ACK Enable/Disable:** Xác nhận (ACK) là tín hiệu Slave gửi để báo đã nhận dữ liệu. Mặc định tắt, cần bật nếu muốn dùng.
4. **Fast Mode Duty Cycle:** Ở tốc độ cao (>100 kHz), có thể điều chỉnh tỷ lệ thời gian mức cao/thấp của SCL (ví dụ: 1:1 hoặc 2:1), ảnh hưởng đến hiệu suất.

A black background with white text

AI-generated content may be incorrect.

## Tạo ra tần số SCL mong muốn từ ngoại vi I2C trên STM32F4x

Tần số SCL là yếu tố quan trọng trong giao tiếp I2C, quyết định tốc độ truyền dữ liệu (Standard Mode: 100 kHz, Fast Mode: ≤400 kHz).

Để điều khiển tần số SCL trên STM32F4x, cần cấu hình hai thanh ghi: **CR2** và **CCR**. Trong CR2, trường **FREQ** cần được thiết lập bằng tần số bus APB mà I2C kết nối.

* **CR2 (Control Register 2):** Chứa trường FREQ (bit 5:0), xác định tần số clock nguồn (APB bus) mà I2C sử dụng.
* **CCR (Clock Control Register):** Điều khiển tần số SCL đầu ra dựa trên FREQ và các cài đặt bổ sung.
* **APB Bus:** Trong STM32F4x, I2C thường nối với APB1 (tốc độ thấp), cần biết tần số APB1 để cấu hình chính xác.
* **Clock Stretching là gì?** Slave giữ SCL ở mức thấp để tạm dừng giao tiếp I2C.
* **Mục đích:** Giúp Slave làm chậm giao tiếp khi không theo kịp tốc độ Master.
* **Ví dụ:**
* Không Clock Stretching: Slave gửi ACK ngay chu kỳ 9.
* Có Clock Stretching: Slave giữ SCL thấp sau chu kỳ 8, gửi ACK khi sẵn sàng, tránh NACK sai.
* **Tự động:** Phần cứng I2C xử lý, chỉ cần bật tính năng qua CR1.NOSTRETCH.
* **Ứng dụng:** Đảm bảo giao tiếp ổn định khi Slave chậm, sẽ được tích hợp trong driver.

**Quy trình Master gửi dữ liệu (như một câu chuyện)**

1. **Bắt đầu cuộc gọi (START Condition - EV5)**:
   * Master nói: "Alo, tôi muốn nói chuyện!" bằng cách tạo tín hiệu START trên bus I2C (giống như bấm nút gọi điện).
   * Sau đó, Master kiểm tra: "Cuộc gọi đã kết nối chưa?" bằng cách đợi cờ **SB** (Start Bit) trong thanh ghi trạng thái bật lên (SB = 1). Đây là sự kiện **EV5**.
   * Nếu không làm bước tiếp theo ngay (gửi số điện thoại Slave), SCL (dòng đồng hồ) sẽ bị "treo" (kéo thấp), giống như cuộc gọi bị giữ chờ.
2. **Gọi đúng số Slave (Address Phase - EV6)**:
   * Master gửi "số điện thoại" của Slave: một byte gồm 7 bit địa chỉ Slave + 1 bit "Tôi muốn gửi tin nhắn" (R/W = 0).
   * Nếu Slave nghe thấy và trả lời "OK, tôi đây!" (ACK), cờ **ADDR** bật lên (ADDR = 1), đây là sự kiện **EV6**.
   * Master phải "nhấc máy" bằng cách đọc trạng thái và xóa cờ **ADDR** (đọc SR1, SR2). Nếu không có Slave nào trả lời (NACK), cuộc gọi thất bại.
3. **Gửi tin nhắn từng chữ (Data Phase - EV8\_1, EV8)**:
   * Master bắt đầu gửi "Tin nhắn: Xin chào!" từng chữ một (mỗi byte dữ liệu).
   * Đầu tiên, Master kiểm tra: "Hộp thư (DR) trống chưa?" qua cờ **TxE** (Transmit Data Empty). Khi **TxE = 1** (EV8\_1), Master bỏ chữ đầu tiên vào hộp thư (**DR**).
   * Hộp thư chuyển chữ đó sang "loa" (shift register) để phát ra ngoài, Slave nghe và nói "Đã nhận!" (ACK).
   * Hộp thư lại trống ngay (**TxE = 1**, EV8), Master tiếp tục bỏ chữ tiếp theo vào. Cứ thế lặp lại cho đến hết tin nhắn.
4. **Kết thúc cuộc gọi (STOP Condition - EV8\_2)**:
   * Khi gửi xong chữ cuối (ví dụ: "Tạm biệt"), Slave trả lời "OK" (ACK).
   * Master đợi cả "loa" xong việc (**BTF = 1**) và "hộp thư" trống (**TxE = 1**), đây là sự kiện **EV8\_2**.
   * Lúc này, Master nói: "Xong, cúp máy đây!" bằng cách tạo tín hiệu STOP. Phần cứng tự động dọn dẹp (**TxE**, **BTF** về 0).

**Clock Stretching là gì?**

* Giống như khi Slave bận nghe điện thoại mà chưa kịp trả lời, nó sẽ giữ dây điện thoại (SCL) để Master không nói tiếp.
* Trong STM32, nếu Master không xử lý kịp (ví dụ: không xóa **SB** hay đợi **BTF**), SCL sẽ bị kéo thấp, giống như "Chờ chút, tôi chưa sẵn sàng!".

Dưới đây là các bước viết driver Master send data

A screenshot of a computer error

AI-generated content may be incorrect.