TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
KHOA ĐIỆN TỬ - VIỄN THÔNG

----- ๛ 🕮 ๛ -----

Bài tập C4.1
Ghép nối hệ thống đo lường, điều khiển số với máy tính

Giảng viên hướng dẫn: TS. Phạm Duy Hưng

Thành viên thực hiện: Nhóm 3

22022118 Phạm Văn Duy

22022103 Ngô Đức Hiếu

22022175 Nguyễn Quốc Toản

22022145 Tạ Đình Kiên

Hà Nội, tháng 3 năm 2025

MỤC LỤC

Bài tập 1: Thiết kế và thực thi một bộ DAQ đơn giản sử dụng VĐK (vào/ra số; vào/ra tương tự)

Yêu cầu bài tập

- **Thu thập dữ liệu từ cảm biến:** Hệ thống cần có khả năng thu thập dữ liệu từ cảm biến ánh sáng . Vi điều khiển sẽ đọc dữ liệu tương tự thông qua bộ chuyển đổi ADC và đọc tín hiệu số qua các chân GPIO. Việc lấy mẫu dữ liệu phải đảm bảo tốc độ phù hợp để thu thập thông tin một cách chính xác, phản ánh đúng sự thay đổi của tín hiệu đầu vào theo thời gian.

- **Xử lý và truyền dữ liệu:** Sau khi thu thập được dữ liệu từ cảm biến, vi điều khiển cần xử lý tín hiệu trước khi truyền đi. Đối với tín hiệu tương tự, có thể sử dụng mạch lọc RC để loại bỏ nhiễu hoặc thực hiện xử lý số để làm mượt tín hiệu. Sau khi xử lý, dữ liệu sẽ được truyền đến máy tính thông qua giao tiếp UART (Serial). Tốc độ truyền cần được lựa chọn hợp lý để đảm bảo dữ liệu không bị mất hoặc sai lệch trong quá trình truyền.

- **Hiển thị và phân tích dữ liệu trên máy tính:** Dữ liệu sau khi được truyền đến máy tính sẽ được hiển thị trên phần mềm như LabVIEW

1. Nguyên tắc hoạt động

- Vi điều khiển thu thập dữ liệu từ cảm biến qua:
  + Đầu vào tương tự (ADC) để lấy mẫu tín hiệu liên tục.
  + Đầu vào số (GPIO) để nhận tín hiệu mức cao/thấp.
- Xử lý dữ liệu trước khi truyền
  + Có thể sử dụng mạch lọc RC để giảm nhiễu.
  + Xử lý số trên vi điều khiển nếu cần thiết.
- Truyền dữ liệu từ vi điều khiển đến máy tính qua UART.
- Phần mềm trên máy tính (LabVIEW hoặc tương tự) hiển thị dữ liệu dưới dạng đồ thị hoặc bảng số liệu theo thời gian thực.

2. Phân tích nguyên tắc hoạt động của hệ thống

- Hệ thống thu thập dữ liệu (DAQ) từ cảm biến ánh sáng hoạt động theo nguyên tắc chuyển đổi tín hiệu từ môi trường vật lý thành dữ liệu số để xử lý và hiển thị trên máy tính. Hệ thống bao gồm các khối chính sau:

a. Hệ thống vật lý (Physical System)

- Đây là môi trường thực tế chứa nguồn sáng, có thể là ánh sáng mặt trời, đèn điện hoặc nguồn sáng nhân tạo khác.

- Cường độ ánh sáng thay đổi theo thời gian và vị trí, tạo ra tín hiệu cần đo lường.

b. Cảm biến chuyển đổi (Transducer Sensor)

- Cảm biến ánh sáng chuyển đổi cường độ ánh sáng thành tín hiệu điện.

- Đầu ra analog: Xuất tín hiệu điện áp thay đổi liên tục theo cường độ ánh sáng.

- Đầu ra số: Chuyển mức tín hiệu HIGH/LOW khi ánh sáng vượt qua một ngưỡng nhất định.

c. Xử lý tín hiệu (Signal Conditioning)

- Lọc tín hiệu: Dùng mạch RC để làm mượt tín hiệu analog, giảm nhiễu.
- Khuếch đại: Để đảm bảo tín hiệu nằm trong dải ADC của Arduino.
- Chuyển đổi mức logic: Nếu đầu ra số không tương thích với điện áp logic của Arduino, cần điều chỉnh để tránh lỗi đọc tín hiệu.

d. Chuyển đổi A/D (Analog to Digital Conversion - ADC)

- Đối với tín hiệu analog, Arduino sử dụng bộ ADC để chuyển đổi thành giá trị số.

- Độ phân giải của ADC: Arduino Uno: 10-bit (0 - 1023) với dải điện áp 0 - 5V.

- Đối với tín hiệu số, Arduino đọc trực tiếp giá trị HIGH hoặc LOW từ cảm biến.

e. Truyền dữ liệu và hiển thị trên LabVIEW

- Dữ liệu từ Arduino được truyền qua Serial (UART) đến máy tính.
- LabVIEW nhận dữ liệu, hiển thị đồng thời cả tín hiệu số và tương tự dưới dạng đồ thị thời gian thực.

3. Các bước thực hiện

- Chuẩn bị linh kiện:
  + Arduino Uno
  + Cảm biến ánh sáng
  + Điện trở
  + Dây nối, breadboard

a. Nối dây cảm biến với Arduino

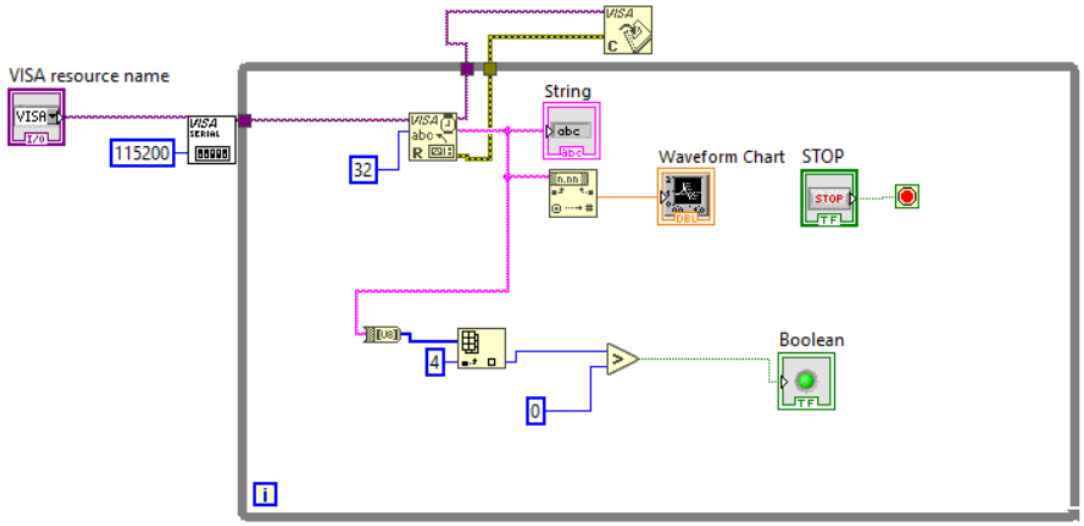
- VCC → 5V Arduino
- GND → GND Arduino
- AO (Analog Output - Đầu ra tương tự) → Chân A0 trên Arduino
- DOUT (Digital Output - Đầu ra số) → Chân D2 trên Arduino

b. Lập trình Arduino để đọc dữ liệu

- Arduino sẽ thực hiện các chức năng sau:
  + Đọc giá trị analog từ cảm biến (độ sáng dưới dạng điện áp).
  + Đọc giá trị digital từ cảm biến (0 hoặc 1, ngưỡng đã cài đặt sẵn).
  + Áp dụng bộ lọc nhiễu : lọc thông thấp.
  + Gửi dữ liệu qua UART (Serial) để hiển thị trên LabVIEW.
- Code thực hiện:

```c
#include <avr/io.h>
#include <util/delay.h>

#define ANALOG_PIN 0
#define DIGITAL_PIN PD2
#define ALPHA 0.1

float filtered_voltage = 0.0;

float lowPassFilter(float newValue, float oldValue) {
    return ALPHA * newValue + (1 - ALPHA) * oldValue;
}

void ADC_Init() {
    ADMUX = (1 << REFS0) | (ANALOG_PIN & 0x07);
    ADCSRA = (1 << ADEN) | (1 << ADSC) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
}

uint16_t ADC_Read() {
    ADCSRA |= (1 << ADSC);
    while (ADCSRA & (1 << ADSC));
    return ADC;
}

void UART_Init() {
    UBRR0H = 0;
    UBRR0L = 8;
    UCSR0B = (1 << TXEN0);
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}

void UART_Transmit(char data) {
    while (!(UCSR0A & (1 << UDRE0)));
    UDR0 = data;
}

void UART_SendString(const char* str) {
    while (*str) {
        UART_Transmit(*str++);
    }
}

void UART_SendFloat(float value) {
    char buffer[10];
    dtostrf(value, 6, 2, buffer);
    UART_SendString(buffer);
}

int main() {
    ADC_Init();
    UART_Init();
    DDRD &= ~(1 << DIGITAL_PIN);

    while (1) {
        uint16_t adc_value = ADC_Read();
        float voltage = (adc_value / 1023.0) * 5.0;
        filtered_voltage = lowPassFilter(voltage, filtered_voltage);
        uint8_t digital_value = (PIND & (1 << DIGITAL_PIN)) ? 1 : 0;

        UART_SendString("ADC:");
        UART_SendFloat(adc_value);
        UART_SendString(", Voltage:");
        UART_SendFloat(voltage);
        UART_SendString(", Filtered:");
        UART_SendFloat(filtered_voltage);
        UART_SendString(", Digital:");
        UART_Transmit(digital_value + '0');
        UART_SendString("\n");

        _delay_ms(50);
    }
}
```

c. Cấu hình Labview



❖ **Hiển thị Tín hiệu Số**

Chức năng:

- Đèn Boolean hoạt động như một tín hiệu số, chỉ có hai trạng thái:
  + Bật (ON - màu xanh, logic 1)
  + Tắt (OFF - màu xám, logic 0)

Hoạt động của mạch:

+ Nhận dữ liệu từ Serial: Chương trình đọc một chuỗi dữ liệu từ cổng VISA Serial.

+ Chuyển đổi dữ liệu: Dữ liệu nhận được có thể là một số nguyên hoặc số thực.

+ So sánh với ngưỡng (0): Một khối so sánh (Greater Than 0) kiểm tra xem giá trị đọc được có lớn hơn 0 hay không.

+ Điều khiển Đèn Boolean:

Nếu giá trị > 0 → Boolean bật sáng (1 - ON).

Nếu giá trị ≤ 0 → Boolean tắt (0 - OFF).

❖ **Hiển thị Tín hiệu Tương Tự**

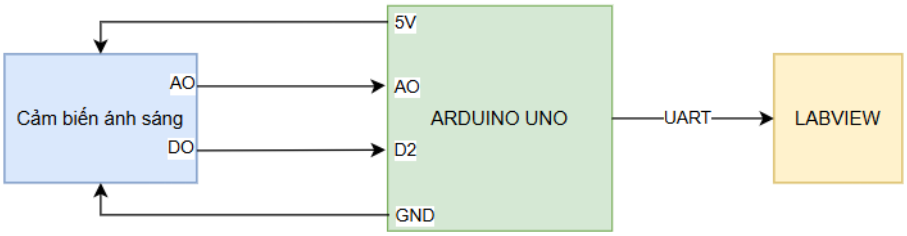
Chức năng:

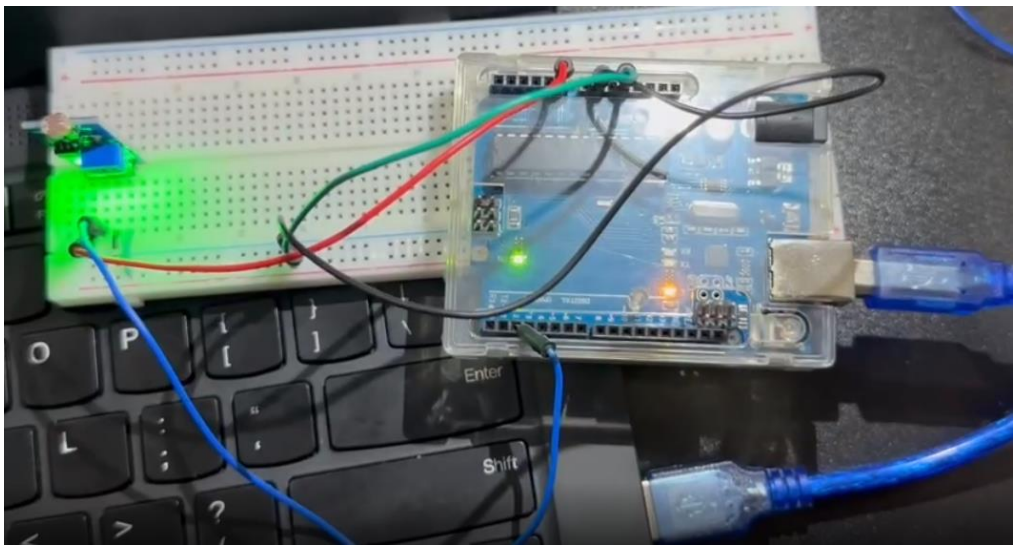
- Waveform Chart hiển thị dữ liệu một cách liên tục theo thời gian, giúp quan sát sự thay đổi của tín hiệu.
- Tín hiệu này thường là tín hiệu tương tự (Analog Signal).

Hoạt động của mạch:

- Nhận dữ liệu từ Serial: Chương trình đọc dữ liệu số từ thiết bị gửi về qua cổng VISA Serial.
- Chuyển đổi dữ liệu: Dữ liệu được chuyển đổi từ chuỗi sang dạng số.
- Cập nhật vào Waveform Chart: Mỗi lần vòng lặp chạy, dữ liệu mới sẽ được đưa vào biểu đồ, giúp theo dõi sự thay đổi của tín hiệu theo thời gian thực.

➢ **Sơ đồ hệ thống:**





Video: DAQ

4.Kiểm tra, đánh giá hiệu quả hệ thống và hiệu chỉnh

1. Đánh giá hệ thống

a. Độ chính xác và nhiễu tín hiệu

- Hệ thống sử dụng ADC trên vi điều khiển để thu thập dữ liệu từ cảm biến ánh sáng. Một số yếu tố có thể ảnh hưởng đến độ chính xác của hệ thống:

+ Độ phân giải ADC: ADC của Arduino có độ phân giải 10-bit, dẫn đến một mức điện áp tối thiểu có thể đo được khoảng 4.88mV (với nguồn 5V).

+ Nhiễu điện từ: Nhiễu từ nguồn cấp điện, môi trường xung quanh hoặc từ chính Arduino có thể ảnh hưởng đến tín hiệu ADC.

+ Lỗi lấy mẫu: Nếu tốc độ lấy mẫu không phù hợp, có thể gây ra méo tín hiệu hoặc mất dữ liệu.

b. Ổn định tín hiệu

- Việc sử dụng bộ lọc thông thấp (Low-pass filter) giúp làm mượt dữ liệu đo được, giảm nhiễu và biến động đột ngột.

- Hệ số lọc (ALPHA) quyết định mức độ phản ứng của hệ thống với thay đổi nhanh của tín hiệu.

- Cần cân nhắc giữa độ mượt của tín hiệu và độ trễ trong phản hồi.

c. Hiệu suất truyền dữ liệu

- Giao tiếp UART có thể bị nghẽn nếu tốc độ truyền quá cao hoặc dữ liệu không được xử lý kịp thời ở phía máy tính.

- Cấu trúc dữ liệu truyền cần đảm bảo dễ phân tích và không gây lỗi khi xử lý trên LabVIEW.

2. Hiệu chỉnh hệ thống

a. Hiệu chỉnh phần cứng

- Ổn định nguồn cấp: Dùng tụ lọc để giảm nhiễu từ nguồn cung cấp.

- Bổ sung mạch lọc RC: Lọc tín hiệu trước khi vào ADC để giảm nhiễu cao tần.

- Chống nhiễu điện từ: Hạn chế dây nối dài không cần thiết và giữ khoảng cách với các thiết bị phát nhiễu mạnh.

b. Hiệu chỉnh phần mềm

- Điều chỉnh hệ số lọc ALPHA: Tùy theo ứng dụng, ALPHA có thể được thay đổi để cân bằng giữa phản ứng nhanh và ổn định tín hiệu.

- Tối ưu thời gian lấy mẫu: Điều chỉnh thời gian giữa các lần đọc ADC để tránh trùng nhiễu từ nguồn tín hiệu.

- Hiệu chỉnh hệ số chuyển đổi ADC: Đo điện áp thực tế tại cảm biến và so sánh với giá trị đo được để xác định sai số và hiệu chỉnh bằng phần mềm.

c. Kiểm tra và đánh giá hiệu suất

- So sánh với thiết bị đo chuẩn: Dùng đồng hồ vạn năng hoặc oscilloscope để kiểm tra tín hiệu thực tế.

- Thử nghiệm với điều kiện thực tế: Chạy hệ thống trong môi trường hoạt động thực tế để kiểm tra độ ổn định.

- Hiệu chỉnh lại nếu cần: Dựa vào kết quả đo, điều chỉnh các tham số như ALPHA, tốc độ lấy mẫu và cấu trúc truyền dữ liệu.

Bài tập 2: Thiết lập truyền thông nối tiếp theo chuẩn RS-232 giữa 2 thiết bị.

I. Yêu cầu của bài toán

1. Mục tiêu của bài toán

Mục tiêu của bài toán này là xây dựng hệ thống giao tiếp giữa hai vi điều khiển thông qua chuẩn RS232, trong đó một vi điều khiển sẽ đảm nhận vai trò Master (gửi tín hiệu) và vi điều khiển còn lại sẽ là Slave (nhận tín hiệu và thực thi hành động). Cụ thể, bài toán yêu cầu Vi điều khiển Master gửi tín hiệu điều khiển qua cổng serial để bật/tắt đèn LED13 trên Vi điều khiển Slave.

2. Các yêu cầu cụ thể

- Giao tiếp RS232: Hai vi điều khiển giao tiếp với nhau qua chuẩn RS232 sử dụng dây TX (Transmit) và RX (Receive) để truyền và nhận dữ liệu.
- Điều khiển LED13: Khi Vi điều khiển Master gửi tín hiệu (ví dụ: ký tự '1' để bật và '0' để tắt), Vi điều khiển Slave sẽ nhận tín hiệu và điều khiển LED13 trên chân 13 của mình.
- Đảm bảo tốc độ truyền ổn định: Cả hai vi điều khiển cần phải thiết lập tốc độ truyền (baud rate) giống nhau để đảm bảo tín hiệu được truyền chính xác.
- Dễ dàng mở rộng: Hệ thống phải có khả năng mở rộng, có thể điều khiển nhiều thiết bị từ một vi điều khiển Master.

3. Phạm vi và giới hạn của hệ thống

- Hệ thống chỉ sử dụng hai dây TX và RX để kết nối giữa hai vi điều khiển, không sử dụng thêm dây khác như GND.
- Sử dụng chuẩn RS232 nên cần chú ý đến mức điện áp của tín hiệu truyền và nhận.
- Hệ thống có thể thực hiện giao tiếp đơn giản, không cần sử dụng các tính năng phức tạp của chuẩn RS232 như kiểm tra lỗi (parity), dừng bit, v.v.

II. Phần thực thi hệ thống

1. Thiết kế phần cứng

Trong phần này, chúng ta sẽ thiết kế mạch phần cứng để kết nối hai vi điều khiển thông qua giao tiếp RS232.

- Kết nối giữa hai vi điều khiển:

- TX của Vi điều khiển Master sẽ được kết nối với RX của Vi điều khiển Slave.

- TX của Vi điều khiển Slave sẽ được kết nối với RX của Vi điều khiển Master nếu cần thiết để có thể phản hồi thông tin.

- Chân GND của cả hai vi điều khiển cần được nối với nhau để tạo thành mạch điện hoàn chỉnh.

2. Cấu hình phần mềm

Phần mềm cho Vi điều khiển Master

Vi điều khiển Master sẽ gửi tín hiệu điều khiển qua cổng serial để bật/tắt LED trên Vi điều khiển Slave.

```
void setup() {
  // Cấu hình UART với baudrate 9600
  UBRR0H = 0;
  UBRR0L = 103; // Giá trị cho baudrate 9600 với 16 MHz
  UCSR0B = (1 << TXEN0); // Bật UART TX
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); // 8-bit data

  // Cấu hình nút nhấn làm INPUT
  DDRD &= ~(1 << PD2); // Chân 2 làm INPUT
  PORTD |= (1 << PD2); // Kích hoạt điện trở pull-up
}

void uart_transmit(char data) {
  while (!(UCSR0A & (1 << UDRE0))); // Chờ khi buffer trống
  UDR0 = data;
}

void loop() {
  if (!(PIND & (1 << PD2))) { // Nếu nút nhấn được nhấn
    uart_transmit('1'); // Gửi ký tự '1'
  } else {
    uart_transmit('0'); // Gửi ký tự '0'
  }
  _delay_ms(500); // Chống dội
}
```

Phần mềm cho Vi điều khiển Slave

Vi điều khiển Slave sẽ nhận tín hiệu từ Vi điều khiển Master và điều khiển LED13 trên chân 13.

nhan.ino

```
#define LED_PIN 13

void setup() {
  // Cấu hình chân LED làm OUTPUT
  DDRB |= (1 << 5); // Chân 13 nằm ở bit 5 của thanh ghi DDRB

  // Khởi tạo UART với baudrate 9600
  UBRR0H = 0;
  UBRR0L = 103; // Giá trị cho baudrate 9600 với 16 MHz
  UCSR0B = (1 << RXEN0) | (1 << TXEN0); // Bật UART TX và RX
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); // 8-bit data
}

char uart_read() {
  while (!(UCSR0A & (1 << RXC0))); // Chờ dữ liệu
  return UDR0;
}

void loop() {
  char receivedValue = uart_read();
  if (receivedValue == '1') {
    PORTB |= (1 << 5); // Bật LED
  } else {
    PORTB &= ~(1 << 5); // Tắt LED
  }
  _delay_ms(1000); // Chờ 1 giây
}
```

3. Kiểm tra, đánh giá hiệu quả hệ thống và hiệu chỉnh

3.1. Kiểm tra hệ thống

- Kiểm tra phần cứng:
  - Xác nhận các kết nối giữa các vi điều khiển, đặc biệt là dây TX và RX, đảm bảo không bị ngắn mạch hoặc đứt gãy.
  - Kiểm tra các mức điện áp trên các tín hiệu TX, RX, và GND (nếu cần).
  - Đảm bảo rằng nếu sử dụng mạch chuyển đổi (chẳng hạn như MAX232), các tín hiệu được chuyển đổi đúng giữa chuẩn RS232 và TTL.
- Kiểm tra phần mềm:
  - Đảm bảo mã nguồn cho cả hai vi điều khiển được tải đúng vào bộ nhớ

vi điều khiển.

- Kiểm tra giao tiếp serial giữa các vi điều khiển bằng cách sử dụng công cụ như Serial Monitsor để xem các tín hiệu có được gửi và nhận chính xác hay không.

- Kiểm tra tín hiệu điều khiển bằng cách gửi các ký tự '1' và '0' từ Vi điều khiển Master và quan sát xem LED13 trên Vi điều khiển Slave có bật và tắt đúng như mong muốn hay không.

- Kiểm tra tốc độ truyền:

- Đảm bảo rằng tốc độ baud rate (ví dụ: 9600 bps) là chính xác và đồng bộ giữa hai vi điều khiển.

- Nếu có sự cố trong việc truyền tín hiệu hoặc nhận dữ liệu không chính xác, hãy thử điều chỉnh tốc độ baud rate và các tham số serial khác.

3.2. Đánh giá hiệu quả hệ thống

Hiệu quả của hệ thống có thể được đánh giá qua các yếu tố sau:

- Độ ổn định của tín hiệu:

- Đảm bảo rằng tín hiệu được truyền qua dây RX và TX ổn định, không có sự mất mát dữ liệu trong suốt quá trình truyền nhận.

- Nếu tín hiệu bị nhiễu hoặc không ổn định, hãy kiểm tra độ dài của dây nối và môi trường xung quanh để hạn chế sự can thiệp của tín hiệu.

- Tốc độ và độ chính xác của việc điều khiển LED:

- Đảm bảo rằng việc gửi tín hiệu điều khiển từ Vi điều khiển Master đến Vi điều khiển Slave là chính xác và kịp thời. Đặc biệt, LED13 phải bật/tắt ngay lập tức sau khi nhận được tín hiệu.

3.3. Hiệu chỉnh và tối ưu hóa

Nếu trong quá trình kiểm tra và đánh giá, có các vấn đề như tín hiệu bị nhiễu, mất dữ liệu, hay không đồng bộ, có thể thực hiện một số hiệu chỉnh sau:

- Điều chỉnh các tham số Serial:

- Đảm bảo rằng cả Vi điều khiển Master và Slave sử dụng cùng tốc độ baud rate, độ dài bit dữ liệu, số bit dừng và bit parity.

- Có thể thử các baud rate khác nhau như 115200 hoặc 57600 để tìm ra tốc độ tối ưu cho hệ thống.
- Cải thiện tín hiệu:
- Sử dụng các dây nối chất lượng cao và giảm thiểu chiều dài của dây nối giữa hai vi điều khiển để giảm sự suy giảm tín hiệu.
- Kiểm tra các lỗi phần cứng:
- Đảm bảo rằng các vi điều khiển đều hoạt động đúng và không bị lỗi phần cứng (chẳng hạn như chân RX hoặc TX bị hỏng).
- Nếu cần, thay đổi các vi điều khiển khác nhau để xác định vấn đề.
- Kiểm tra nguồn điện:
- Đảm bảo rằng nguồn điện cho các vi điều khiển ổn định, vì nếu điện áp thay đổi hoặc không ổn định có thể ảnh hưởng đến việc truyền tín hiệu và hoạt động của hệ thống.

4.Video thực nghiệm

Link drive: 🎥

https://drive.google.com/file/d/18B9414GxG8xlNcaHyV30AeOsqjHWjtq8/view?usp=sharing

Bài tập 3: Thiết lập truyền thông giữa 2 vi điều khiển theo chuẩn I2C.

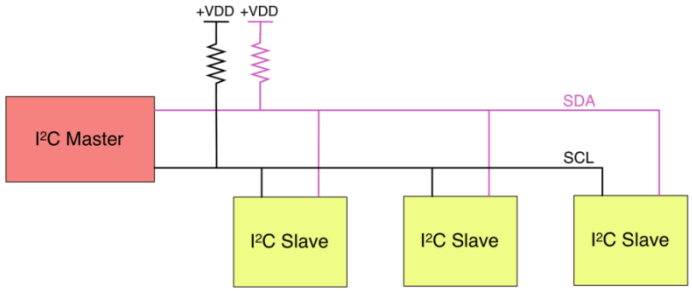
I. Yêu cầu bài toán

Thiết lập giao tiếp giữa hai vi điều khiển theo chuẩn I2C - một chuẩn giao tiếp nối tiếp 8-bit đơn cực, bán song công, hỗ trợ nhiều thiết bị phụ thuộc (multi-slave).

1. Phân tích bài toán

a. Nguyên tắc hoạt động của mạch

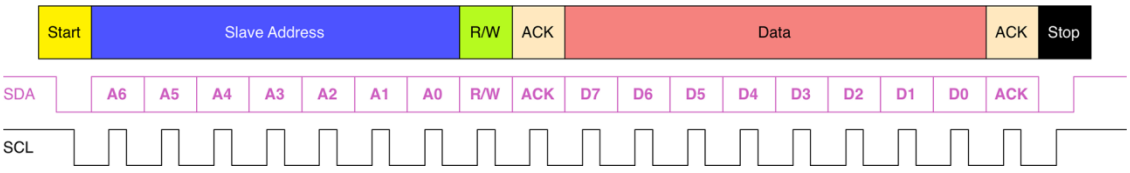
Mạch thực hiện giao tiếp giữa hai vi điều khiển sử dụng giao thức I2C.



Bus I2C gồm hai đường open – drain hai chiều:

- **SDA (Serial Data Line):** Dây truyền dữ liệu nối tiếp giữa master và slave.
- **SCL (Serial Clock Line):** Dây xung nhịp do master tạo ra để đồng bộ dữ liệu.
- Hai dây này được kéo lên bởi điện trở pull up, thường là 4,7kΩ.

Nguyên tắc hoạt động của mạch sử dụng giao thức I2C:



1. **Truyền dữ liệu**:
    - Master khởi tạo tín hiệu **START**, báo hiệu bắt đầu truyền dữ liệu.
    - Master gửi địa chỉ của thiết bị slave cần giao tiếp kèm theo bit **R/W** (Read/Write).
    - Slave có địa chỉ tương ứng phản hồi bằng tín hiệu **ACK (Acknowledge)**.

- Dữ liệu được truyền theo từng byte (8 bit), theo quy tắc **MSB first** (bit có trọng số cao trước).

2. **Nhận dữ liệu**:
   - Nếu là chế độ đọc, slave sẽ gửi dữ liệu cho master khi nhận tín hiệu **ACK**.
   - Nếu là chế độ ghi, master gửi dữ liệu cho slave, mỗi byte gửi đi đều phải được xác nhận bằng **ACK**.
   - Quá trình có thể tiếp tục với nhiều byte dữ liệu hoặc kết thúc bằng tín hiệu **STOP**.
3. **Tín hiệu điều khiển**:
   - **START Condition**: Khi SDA chuyển từ **HIGH → LOW** trong khi SCL đang ở mức **HIGH**.
   - **STOP Condition**: Khi SDA chuyển từ **LOW → HIGH** trong khi SCL đang ở mức **HIGH**.
   - **ACK/NACK**: Sau mỗi byte dữ liệu, bên nhận sẽ kéo SDA xuống mức **LOW** để xác nhận (ACK) hoặc giữ SDA mức **HIGH** nếu không chấp nhận dữ liệu (NACK).

b. Phân tích đối tượng đo và điều khiển

Master

- Đóng vai trò điều khiển chính trong hệ thống.
- Gửi dữ liệu đến Slave thông qua giao thức I2C.

Slave

- Nhận dữ liệu từ Master thông qua giao tiếp I2C.
- Điều khiển LED dựa trên giá trị nhận được từ Master.
- LED được bật/tắt theo từng bit của dữ liệu nhận được (bit 0 → LED 1, bit 1 → LED 2,...).

Đối tượng điều khiển (LED)

- Các LED được nối với Slave trên các chân digital.
- Slave điều khiển bật/tắt LED dựa trên từng bit của dữ liệu nhận từ Master.

c. Yêu cầu phần cứng

1. STM32 F401RE làm Master
2. Arduino Uno làm Slave
3. Điện trở kéo lên 4,7 – 10kΩ cho hai đường SDA và SCL
4. Mạch đèn Led

d. Yêu cầu hệ thống

1. Phần cứng: cần

- **Vi điều khiển hỗ trợ giao tiếp I²C:**
- **Đường SDA và SCL:**
  - Hai dây tín hiệu phải được kéo lên mức cao bằng điện trở **pull-up**.
  - Các chân GPIO tham gia giao tiếp I²C phải hỗ trợ chế độ **open-drain** hoặc **open-collector**.
- **Điện áp nguồn:**
  - Các thiết bị trên bus I²C phải hoạt động ở cùng mức điện áp (VDĐ), là **3.3V**.
- **Tốc độ giao tiếp:**
  - Hệ thống phải hỗ trợ một trong các tốc độ chuẩn của I²C:
    - **100 kHz** (Standard Mode)
    - **400 kHz** (Fast Mode)
    - **1 MHz** (Fast Mode Plus)
    - **3.4 MHz** (High Speed Mode)

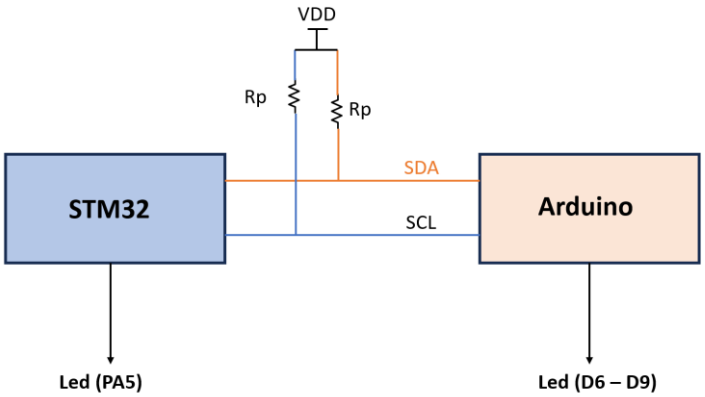
2. Phần mềm

- **Cấu hình vi điều khiển:**
  - Cấu hình chế độ **Master/Slave** phù hợp cho từng thiết bị.
  - Thiết lập địa chỉ **7-bit** cho thiết bị **slave**.
- **Quy trình truyền dữ liệu:**
  - Thiết lập đúng trình tự START, địa chỉ, dữ liệu và STOP để tránh xung đột trên bus.

- Kiểm tra phản hồi ACK/NACK sau mỗi byte dữ liệu truyền/nhận.
- Xử lý lỗi nếu thiết bị không phản hồi hoặc mất kết nối.

II. Thực thi hệ thống

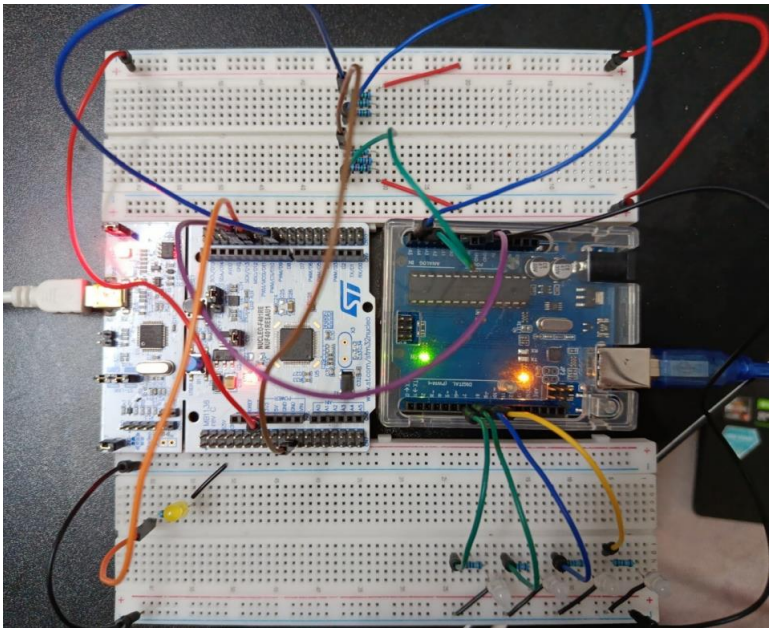
1. Sơ đồ khối hệ thống



2. Kết nối các module phần cứng

STM32 (Master)	Kết nối	Arduino (Slave)
PB6 (I2C1_SCL)	Rp = 5kΩ, VDD = 3,3V	A5 (SCL)
PB7 (I2C1_SDA)	Rp = 5kΩ, VDD = 3,3V	A4 (SDA)
GND	➔	GND

LED	Kết nối
LED 1	D6 → Điện trở 10Ω
LED 2	D7 → Điện trở 10Ω
LED 3	D8 → Điện trở 10Ω
LED 4	D9 → Điện trở 10Ω
LED 5	PA5 → Điện trở 10Ω



3. **Viết chương trình cho vi điều khiển**

Master (STM32F401RE)

- Gửi dữ liệu (0-15) đến Arduino qua I2C.
- Khi gửi giá trị 15, Master sẽ yêu cầu phản hồi từ Arduino.
- Nếu nhận được phản hồi là 1, nó sẽ bật LED trên PA5, nếu không sẽ tắt.

Slave (Arduino)

- Nhận dữ liệu từ Master.
- Điều khiển LED từ D6 đến D9 theo từng bit của dữ liệu nhận được.
- Nếu dữ liệu nhận được là 15, nó sẽ gửi phản hồi 1 khi Master yêu cầu.
- Nếu dữ liệu khác 15, nó sẽ gửi phản hồi 0.

Code cho Stm32:

```c
#include "stm32f4xx.h"

void I2C1_Init(void);
void I2C1_Write(uint8_t slaveAddr, uint8_t data);
uint8_t I2C1_Read(uint8_t slaveAddr);

int main(void) {
    I2C1_Init();

    // Bật clock GPIOA để dùng LED PA5
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
    GPIOA->MODER |= (1 << (5 * 2)); // PA5 Output mode

    while (1) {
        for (uint8_t i = 0; i < 16; i++) {
            I2C1_Write(0x08, i); // Gửi dữ liệu i đến Arduino

            // Nếu gửi 15, kiểm tra phản hồi từ Arduino
            if (i == 15) {
                uint8_t received = I2C1_Read(0x08);
                if (received == 1) {
                    GPIOA->BSRR = (1 << 5); // Bật LED PA5
                }
            } else {
                GPIOA->BSRR = (1 << (5 + 16)); // Tắt LED PA5 ở chu kỳ tiếp theo
            }

            for (volatile int j = 0; j < 500000; j++); // Delay
        }
    }
}

void I2C1_Init(void) {
    RCC->APB1ENR |= RCC_APB1ENR_I2C1EN;  // Bật clock I2C1
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN; // Bật clock GPIOB

    // Cấu hình PB6 (SCL) và PB7 (SDA) ở chế độ AF4 (I2C)
    GPIOB->MODER |= (2 << (6 * 2)) | (2 << (7 * 2));
    GPIOB->OTYPER |= (1 << 6) | (1 << 7);
    GPIOB->OSPEEDR |= (3 << (6 * 2)) | (3 << (7 * 2));
    GPIOB->AFR[0] |= (4 << (6 * 4)) | (4 << (7 * 4));

    I2C1->CR1 |= I2C_CR1_SWRST;   // Reset I2C
    I2C1->CR1 &= ~I2C_CR1_SWRST;  // Thoát reset

    I2C1->CR2 = 16;              // Tần số APB1 = 16MHz
    I2C1->CCR = 80;              // Chu kỳ clock I2C (100kHz)
    I2C1->TRISE = 17;            // Thiết lập thời gian
    I2C1->CR1 |= I2C_CR1_PE;     // Bật I2C1
}
```

```
52  void I2C1_Write(uint8_t slaveAddr, uint8_t data) {
53      while (I2C1->SR2 & I2C_SR2_BUSY); // Chờ nếu bus đang bận
54
55      I2C1->CR1 |= I2C_CR1_START;  // Tạo tín hiệu START
56      while (!(I2C1->SR1 & I2C_SR1_SB));
57
58      I2C1->DR = (slaveAddr << 1);  // Gửi địa chỉ Slave + Write
59      while (!(I2C1->SR1 & I2C_SR1_ADDR));
60      (void)I2C1->SR2;
61
62      I2C1->DR = data;  // Gửi dữ liệu
63      while (!(I2C1->SR1 & I2C_SR1_BTF));
64
65      I2C1->CR1 |= I2C_CR1_STOP;  // STOP
66  }
68  uint8_t I2C1_Read(uint8_t slaveAddr) {
69      uint8_t receivedData = 0;
70
71      while (I2C1->SR2 & I2C_SR2_BUSY);
72
73      I2C1->CR1 |= I2C_CR1_START;
74      while (!(I2C1->SR1 & I2C_SR1_SB));
75
76      I2C1->DR = (slaveAddr << 1) | 1;  // Đọc từ Slave
77      while (!(I2C1->SR1 & I2C_SR1_ADDR));
78      (void)I2C1->SR2;
79
80      while (!(I2C1->SR1 & I2C_SR1_RXNE));
81      receivedData = I2C1->DR;
82
83      I2C1->CR1 |= I2C_CR1_STOP;  // STOP
84
85      return receivedData;
86  }
```

Code cho arduino:

```ino
#include <Wire.h>

#define NUM_LEDS 4
const int ledPins[NUM_LEDS] = {6, 7, 8, 9}; // LED từ D2 đến D9
volatile uint8_t lastData = 0; // Biến lưu giá trị nhận được

void setup() {
    Wire.begin(0x08);  // Địa chỉ I2C của Slave (0x08)
    Wire.onReceive(receiveEvent);  // Gọi khi nhận dữ liệu từ Master
    Wire.onRequest(requestEvent);  // Gọi khi Master yêu cầu dữ liệu

    // Cấu hình chân LED làm OUTPUT
    for (int i = 0; i < NUM_LEDS; i++) {
        pinMode(ledPins[i], OUTPUT);
        digitalWrite(ledPins[i], LOW); // Ban đầu tắt hết LED
    }

    Serial.begin(115200);
}

void loop() {
    // Cập nhật LED theo dữ liệu nhận được
    updateLEDs(lastData);
}

// Khi nhận dữ liệu từ Master
void receiveEvent(int numBytes) {
    if (Wire.available()) {
        lastData = Wire.read();  // Lưu dữ liệu nhận được

        Serial.print("Received: ");
        Serial.println(lastData);
    }
}

// Khi Master yêu cầu dữ liệu
void requestEvent() {
    if (lastData == 15) {
        Wire.write(1);  // Trả về 1 nếu nhận được 15
    } else {
        Wire.write(0);  // Trả về 0 nếu không phải 15
    }
}

// Điều khiển LED dựa trên dữ liệu nhận được
void updateLEDs(uint8_t data) {
    for (int i = 0; i < NUM_LEDS; i++) {
        digitalWrite(ledPins[i], (data >> i) & 0x01); // Bật/tắt LED theo từng bit
    }
}
```

4. Kiểm tra, đánh giá hiệu quả hệ thống và hiệu chỉnh

4.1. Tiêu chí đánh giá

- **I2C hoạt động ổn định:** STM32 gửi đúng dữ liệu, Arduino nhận và phản hồi chính xác.
- **LED điều khiển đúng:** Arduino bật/tắt LED theo dữ liệu nhận từ STM32, STM32 bật LED PA5 khi nhận phản hồi từ Arduino.
- **Tín hiệu SDA, SCL chuẩn I2C:** Kiểm tra bằng oscilloscope hoặc logic analyzer.
- **Hệ thống không bị treo, thời gian đáp ứng nhanh.**

4.2. Kiểm tra thực tế

1. **Gửi dữ liệu từ STM32 đến Arduino**
    - STM32 gửi giá trị 15, Arduino nhận và bật LED D6.
    - Kiểm tra Serial Monitor trên Arduino hiển thị Received: 15.
2. **Arduino phản hồi dữ liệu về STM32**
    - Khi nhận 15, Arduino gửi 1.
    - STM32 nhận phản hồi và bật LED PA5.
3. **Kiểm tra SDA, SCL bằng oscilloscope**
    - Nếu không có tín hiệu, kiểm tra dây kết nối, điện trở pull-up 5kΩ, tốc độ I2C.
4. **Kiểm tra lỗi NACK (No Acknowledge)**
    - Nếu STM32 không nhận phản hồi, kiểm tra lại Wire.onRequest() trên Arduino.

4.3. Hiệu chỉnh hệ thống

Sau khi kiểm tra hoạt động của hệ thống, nếu xảy ra lỗi hoặc hoạt động không như mong muốn, cần thực hiện hiệu chỉnh dựa trên nguyên nhân cụ thể.

4.3.1. Hiệu chỉnh phần cứng

1. **Kiểm tra kết nối dây giữa STM32 và Arduino**
    - Đảm bảo chân **SDA (PB7)** và **SCL (PB6)** của STM32 kết nối đúng với **SDA (A4)** và **SCL (A5)** của Arduino.
    - Kiểm tra nguồn cấp **3.3V/5V** cho từng module.

2. **Kiểm tra điện trở pull-up**
    - Nếu tín hiệu yếu hoặc nhiễu, thay đổi điện trở pull-up.
3. **Sử dụng oscilloscope hoặc logic analyzer**
    - Kiểm tra SDA, SCL có tín hiệu đúng không.
    - Nếu tín hiệu méo hoặc mất xung clock, kiểm tra tốc độ I2C.

4.3.2. Hiệu chỉnh phần mềm trên STM32

1. **STM32 không gửi được dữ liệu I2C**
    - Kiểm tra thanh ghi **SR2_BUSY** có bị kẹt ở trạng thái bận không.
    - Nếu có lỗi NACK (No Acknowledge), kiểm tra địa chỉ I2C **(0x08 << 1)**.
2. **STM32 không nhận phản hồi từ Arduino**
    - Kiểm tra hàm I2C1_Read(), thử đọc dữ liệu từ Arduino bằng debug.
    - Nếu luôn nhận giá trị 0, kiểm tra Wire.onRequest() trên Arduino.
3. **LED PA5 không bật khi nhận phản hồi từ Arduino**
    - Debug bằng printf() để xem STM32 có nhận 1 từ Arduino không.
    - Nếu nhận sai, kiểm tra logic xử lý dữ liệu trong I2C1_Read().

4.3.3. Hiệu chỉnh phần mềm trên Arduino

1. **Arduino không nhận dữ liệu từ STM32**
    - Kiểm tra Wire.onReceive(receiveEvent); có được gọi không.
    - Thêm debug Serial.print(lastData); để kiểm tra giá trị nhận được.
2. **Arduino không phản hồi đúng khi STM32 đọc dữ liệu**
    - Kiểm tra Wire.onRequest(requestEvent); có được gọi không.
    - Nếu không gửi đúng 1 khi nhận 15, kiểm tra lại if (lastData == 15).
3. **LED không sáng theo dữ liệu nhận được**

- Debug bằng Serial.print(lastData, BIN); để kiểm tra dữ liệu dạng nhị phân.
- Kiểm tra updateLEDs(lastData); có được gọi không.

4.3.4. Thử nghiệm lại sau khi hiệu chỉnh

Sau khi khắc phục các lỗi trên, cần kiểm tra lại hệ thống theo các bước sau:

1. Gửi dữ liệu từ STM32 đến Arduino, kiểm tra LED Arduino có sáng đúng không.
2. Yêu cầu dữ liệu từ Arduino về STM32, kiểm tra LED PA5 có bật không.
3. Dùng oscilloscope kiểm tra lại tín hiệu I2C.

Video thực nghiệm

 I2C.mp4

Bài tập 4: Thiết lập mạng truyền thông sử dụng chuẩn RS485 (không bắt buộc)

I. Yêu cầu của bài toán

1. Mục tiêu của bài toán

Mục tiêu của bài toán này là xây dựng một hệ thống giao tiếp giữa hai vi điều khiển Arduino sử dụng chuẩn RS485, thay vì chuẩn UART thông thường. Hệ thống sử dụng hai module RS485 to TTL để kết nối hai vi điều khi khiển Arduino, và qua đó, điều khiển đèn LED trên vi điều khiển Slave từ vi điều khiển Master. Việc sử dụng RS485 giúp đảm bảo tính ổn định và khả năng truyền tín hiệu ở khoảng cách xa hơn, đặc biệt trong môi trường có nhiều nhiễu.

Khi vi điều khiển Master gửi tín hiệu '1', đèn LED trên vi điều khiển Slave sẽ bật lên. Ngược lại, khi Master gửi tín hiệu '0', đèn LED trên Slave sẽ tắt. Giao tiếp giữa các vi điều khiển này sử dụng chuẩn RS485 với hai module RS485 to TTL.

2. Các yêu cầu cụ thể

- Giao tiếp RS485: Hệ thống sử dụng chuẩn giao tiếp RS485 giữa hai vi điều khiển. Các tín hiệu truyền giữa các vi điều khiển được chuyển qua hai dây A và B của chuẩn RS485, thay vì sử dụng các chân TX và RX của giao tiếp UART thông thường.
- Điều khiển LED: Vi điều khiển Master sẽ gửi tín hiệu điều khiển qua RS485 đến vi điều khiển Slave. Vi điều khiển Slave sẽ nhận tín hiệu và điều khiển bật hoặc tắt đèn LED kết nối với chân 13 của nó.
- Cấu hình phần mềm: Vi điều khiển Master và Slave được lập trình để giao tiếp qua RS485, thay vì UART. Vi điều khiển Master sẽ gửi các tín hiệu '1' và '0' qua RS485, còn Slave sẽ xử lý các tín hiệu này và bật/tắt đèn LED tương ứng.

II. Phần thực thi hệ thống

1. Thiết kế phần cứng

1.1. Kết nối giữa các vi điều khiển

Kết nối giữa các vi điều khiển Arduino Master và Slave thông qua hai module RS485 to TTL. Dưới đây là sơ đồ kết nối chi tiết:

- Vi điều khiển Master:
  - Chân TX của vi điều khiển Master nối với chân RX của module RS485 to TTL thứ nhất (module RS485 Master).
  - Chân RX của vi điều khiển Master nối với chân TX của module RS485 to TTL thứ nhất.
  - Chân GND của vi điều khiển Master nối với chân GND của module RS485 Master.
- Module RS485 Master (Module RS485 to TTL thứ 1):
  - Chân TX và RX của module RS485 Master được nối đến chân TX và RX của vi điều khiển Master.
  - Dây A và B trên module RS485 Master được nối đến dây A và B của module RS485 Slave.
- Module RS485 Slave (Module RS485 to TTL thứ 2):
  - Chân A và B của module RS485 Slave được nối với dây A và B của module RS485 Master.
  - Chân TX và RX của module RS485 Slave được nối đến chân TX và RX của vi điều khiển Slave.
  - Chân GND của module RS485 Slave được nối với chân GND của vi điều khiển Slave.
- Vi điều khiển Slave:
  - Chân TX của vi điều khiển Slave nối với chân RX của module RS485 Slave.
  - Chân RX của vi điều khiển Slave nối với chân TX của module RS485 Slave.

- Chân GND của vi điều khiển Slave nối với chân GND của module RS485 Slave.

1.2.Cấu hình chân

- Vi điều khiển Slave sẽ sử dụng chân 13 (PB5) để điều khiển LED.
- Chân TX và RX trên cả hai module RS485 to TTL sẽ được kết nối trực tiếp với các chân TX và RX của Arduino Master và Arduino Slave.

2. Cấu hình phần mềm

2.1. Phần mềm cho Arduino Master

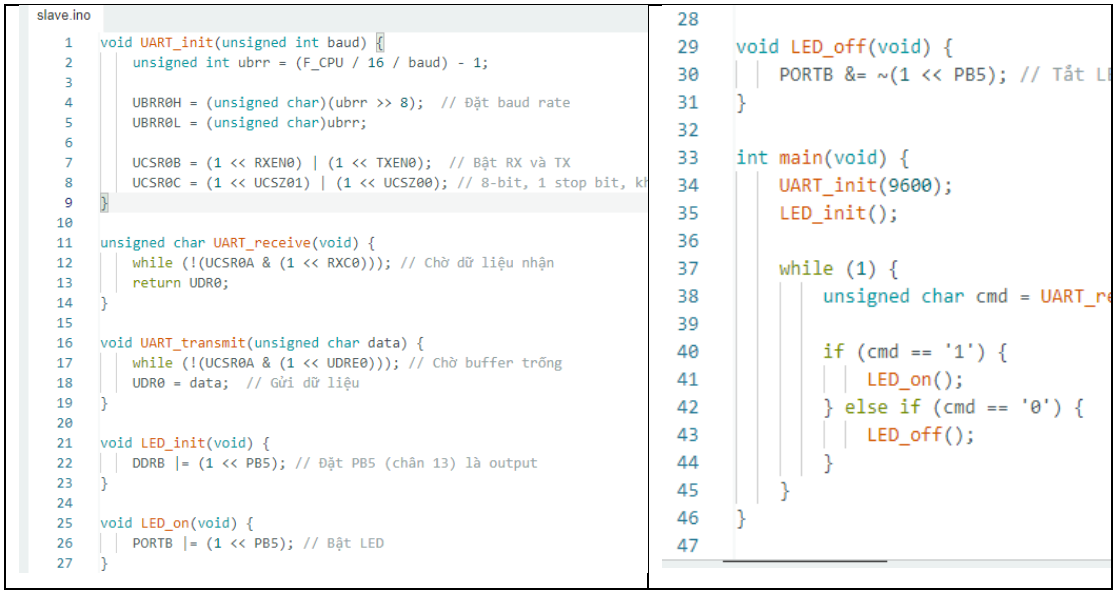
Vi điều khiển Master cần được lập trình để gửi tín hiệu qua giao tiếp RS485. Dưới đây là mã nguồn phần mềm cho vi điều khiển Master:

master.ino

```
#include <avr/io.h>
#include <util/delay.h>

void UART_init(unsigned int baud) {
    unsigned int ubrr = (F_CPU / 16 / baud) - 1;

    UBRR0H = (unsigned char)(ubrr >> 8);  // Đặt baud rate
    UBRR0L = (unsigned char)ubrr;

    UCSR0B = (1 << TXEN0);  // Bật TX
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); // 8-bit, 1 stop bit, không có parity
}

void UART_transmit(unsigned char data) {
    while (!(UCSR0A & (1 << UDRE0))); // Chờ cho đến khi buffer trống
    UDR0 = data;  // Gửi dữ liệu
}

int main(void) {
    UART_init(9600);

    while (1) {
        UART_transmit('1');  // Gửi '1'
        _delay_ms(2000);

        UART_transmit('0');  // Gửi '0'
        _delay_ms(2000);
    }
}

```

2.2. Phần mềm cho Arduino Slave

Vi điều khiển Slave sẽ nhận tín hiệu từ Master qua RS485 và điều khiển LED. Dưới đây là mã nguồn phần mềm cho vi điều khiển Slave:



```
slave.ino
1  void UART_init(unsigned int baud) {
2      unsigned int ubrr = (F_CPU / 16 / baud) - 1;
3
4      UBRR0H = (unsigned char)(ubrr >> 8);  // Đặt baud rate
5      UBRR0L = (unsigned char)ubrr;
6
7      UCSR0B = (1 << RXEN0) | (1 << TXEN0);  // Bật RX và TX
8      UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); // 8-bit, 1 stop bit, kh
9  }
10
11 unsigned char UART_receive(void) {
12     while (!(UCSR0A & (1 << RXC0))); // Chờ dữ liệu nhận
13     return UDR0;
14 }
15
16 void UART_transmit(unsigned char data) {
17     while (!(UCSR0A & (1 << UDRE0))); // Chờ buffer trống
18     UDR0 = data;  // Gửi dữ liệu
19 }
20
21 void LED_init(void) {
22     DDRB |= (1 << PB5); // Đặt PB5 (chân 13) là output
23 }
24
25 void LED_on(void) {
26     PORTB |= (1 << PB5); // Bật LED
27 }
```

```
28
29 void LED_off(void) {
30     PORTB &= ~(1 << PB5); // Tắt LE
31 }
32
33 int main(void) {
34     UART_init(9600);
35     LED_init();
36
37     while (1) {
38         unsigned char cmd = UART_re
39
40         if (cmd == '1') {
41             LED_on();
42         } else if (cmd == '0') {
43             LED_off();
44         }
45     }
46 }
47
```

3. Kiểm tra, đánh giá hiệu quả hệ thống và hiệu chỉnh

3.1. Kiểm tra kết nối phần cứng

Trước khi thực hiện kiểm tra phần mềm và đánh giá hiệu quả hệ thống, điều quan trọng là phải đảm bảo rằng tất cả các kết nối phần cứng giữa các vi điều khiển Arduino và các module RS485 to TTL đã được thực hiện chính xác. Các bước kiểm tra cơ bản bao gồm:

- Kiểm tra kết nối TX, RX: Đảm bảo rằng chân TX của vi điều khiển Master nối với chân RX của module RS485 Master và tương tự với vi điều khiển Slave. Chân RX của Master phải được nối với chân TX của module RS485, và các chân A và B trên các module RS485 phải được kết nối chính xác.

- Kiểm tra nguồn cung cấp: Kiểm tra nguồn cấp điện cho các vi điều khiển và module RS485 để đảm bảo tất cả các thiết bị nhận được đủ nguồn điện để hoạt động ổn định.

- Kiểm tra độ ổn định của tín hiệu: Nếu có thể, sử dụng một máy phân tích tín hiệu hoặc oscilloscope để kiểm tra độ ổn định của tín hiệu RS485 trong quá trình truyền nhận. Điều này giúp đảm bảo không có tín hiệu mất mát hoặc xung đột trên đường truyền.

3.2. Kiểm tra phần mềm

Sau khi phần cứng đã được kiểm tra và đảm bảo kết nối đúng, việc kiểm tra phần mềm là bước tiếp theo. Các bước cần thực hiện trong kiểm tra phần mềm:

- Kiểm tra giao tiếp giữa Master và Slave: Thực hiện việc gửi tín hiệu điều khiển từ vi điều khiển Master đến Slave. Xác nhận rằng khi Master gửi tín hiệu '1', LED trên Slave bật lên, và khi Master gửi tín hiệu '0', LED trên Slave tắt.
- Kiểm tra sự ổn định của tín hiệu truyền: Kiểm tra hệ thống trong suốt một khoảng thời gian dài để đảm bảo rằng các tín hiệu không bị gián đoạn hoặc mất mát, đặc biệt khi hệ thống chạy ở khoảng cách xa hơn.
- Kiểm tra tốc độ truyền tín hiệu: Nếu có bất kỳ sự cố nào xảy ra khi truyền dữ liệu với tốc độ baud rate cao, hãy giảm tốc độ baud rate và kiểm tra lại. Một tốc độ baud rate quá cao có thể gây ra lỗi khi truyền dữ liệu trong môi trường có nhiễu.

3.3. Đánh giá hiệu quả hệ thống

Để đánh giá hiệu quả của hệ thống, có thể tiến hành các bước dưới đây:

- Đánh giá về khoảng cách truyền tín hiệu: RS485 có thể truyền tín hiệu ổn định ở khoảng cách xa hơn so với chuẩn UART thông thường. Hãy kiểm tra khả năng truyền tín hiệu trên các khoảng cách khác nhau để xác định phạm vi hoạt động của hệ thống. Thử nghiệm với khoảng cách 10 mét, 50 mét, và 100 mét để đánh giá độ ổn định của tín hiệu.
- Đánh giá độ ổn định và độ nhiễu: Kiểm tra xem hệ thống có gặp phải vấn đề về nhiễu tín hiệu trong môi trường nhiều thiết bị điện tử hay không. RS485 được thiết kế để chống nhiễu tốt hơn UART, nhưng việc kiểm tra trong các môi trường có nhiễu điện từ mạnh vẫn rất quan trọng.
- Đánh giá khả năng tương thích: Kiểm tra hệ thống khi kết nối với nhiều vi điều khiển Arduino hoặc nhiều thiết bị cùng một lúc, để xem hệ thống có thể mở rộng và vẫn hoạt động ổn định không. Mặc dù RS485 hỗ trợ

kết nối nhiều thiết bị, nhưng cần đảm bảo rằng các module và phần mềm đã được tối ưu hóa cho việc mở rộng.

3.4. Hiệu chỉnh hệ thống

Trong quá trình kiểm tra và đánh giá, nếu phát hiện bất kỳ vấn đề nào, các bước hiệu chỉnh cần thực hiện bao gồm:

- Hiệu chỉnh phần cứng: Nếu tín hiệu RS485 bị suy giảm hoặc mất tín hiệu ở khoảng cách dài, có thể cần thêm các bộ khuếch đại tín hiệu RS485 hoặc tối ưu hóa lại cách nối dây để giảm thiểu suy hao tín hiệu. Cũng có thể cần thay đổi cách nối dây A-B để đảm bảo tín hiệu không bị đảo ngược.
- Hiệu chỉnh phần mềm: Nếu có sự cố liên quan đến độ trễ trong việc nhận dữ liệu, có thể cần phải kiểm tra lại cấu hình của phần mềm như tốc độ baud rate, độ dài của thông điệp hoặc số lượng ký tự cần truyền. Cũng có thể thử tối ưu hóa cách sử dụng bộ đệm (buffer) trong phần mềm để cải thiện hiệu suất.
- Kiểm tra lại các thành phần khác: Nếu việc truyền tín hiệu không ổn định, có thể xem xét thay đổi các module RS485 to TTL hiện tại hoặc kiểm tra lại phần cứng của vi điều khiển (đặc biệt là phần cổng UART).

3.5. Tổng kết và Đề xuất cải tiến

- Tính ổn định: Hệ thống đã chứng minh được tính ổn định khi truyền tín hiệu qua RS485, đặc biệt khi truyền ở khoảng cách xa. Việc sử dụng RS485 là một giải pháp hiệu quả để thay thế giao tiếp UART trong các ứng dụng cần truyền tín hiệu xa hoặc trong môi trường có nhiều nhiễu.
- Khả năng mở rộng: Hệ thống này có khả năng mở rộng để kết nối nhiều thiết bị Arduino khác nhau trong cùng một hệ thống sử dụng bus RS485, giúp dễ dàng xây dựng các hệ thống phức tạp hơn như hệ thống giám sát hoặc điều khiển phân tán.
- Đề xuất cải tiến: Để nâng cao tính ổn định và hiệu suất của hệ thống, có thể thêm các bộ khuếch đại tín hiệu RS485 hoặc sử dụng các module

RS485 có tính năng chống nhiễu mạnh hơn trong các môi trường có nhiều điện từ.

4. Video thực nghiệm

Link drive: 

https://drive.google.com/file/d/18B2w8gNXe9twOzELH3t8DnpLDN1ndDey/view?usp=sharing

Bảng đánh giá thành viên

Thành viên	Công việc	Đóng góp
22022118 Phạm Văn Duy	Nghiên cứu lí thuyết, mã nguồn và thực thi bài tập 1, 2, 3, 4; Làm báo cáo word bài 3: thiết lập giao tiếp I2C	**25%**
22022103 Ngô Đức Hiếu	Nghiên cứu lí thuyết, mã nguồn và thực thi bài tập 1, 2, 3, 4; Làm báo cáo word bài 4: thiết lập giao tiếp RS485	**25%**
22022175 Nguyễn Quốc Toản	Nghiên cứu lí thuyết, mã nguồn và thực thi bài tập 1, 2, 3, 4; Làm báo cáo word bài 1: thiết kế bộ DAQ đơn giản	**25%**
22022145 Tạ Đình Kiên	Nghiên cứu lí thuyết, mã nguồn và thực thi bài tập 1, 2, 3, 4; Làm báo cáo word bài 2: thiết lập giao tiếp RS232	**25%**