

THIẾT KẾ HỆ THỐNG NHÚNG KHÔNG DÂY

BÁO CÁO LAB02

NHÓM 5

Họ và tên	MSSV
Trần Lê Minh Đăng	21520684
Lê Hữu Đạt	21520697
Trần Văn Dương	21520763

Bài tập 1: Tạo một project từ ví dụ esp-idf/peripherals/i2c/i2c_self_test. Chỉnh sửa lại project để giao tiếp với ESP32 theo mô tả bên dưới? Giải thích cách cài đặt?

- Tạo project:

The screenshot shows the 'New Project' wizard in the ESP-IDF environment. The 'Enter Project directory' field is set to 'd:\HKII_2023_2024\CE232\ThucHanh\Lab02'. The 'Choose ESP-IDF Target' dropdown is set to 'ESP32 module'. The 'Choose serial port' dropdown is set to 'no port'. The 'OpenOCD Configuration files' field contains the path 'interface/ftdi/esp32_devkitj_v1.cfg,target/esp32.cfg'. The 'Add your ESP-IDF Component directory' field is empty. A 'Choose Template' button is visible at the bottom right.

- Trong mục i2c, chọn i2c_self_test

The screenshot shows the 'New Project' dialog in the ESP-IDF environment. The 'i2c' category is selected in the left sidebar, and the 'i2c_self_test' template is highlighted. The main area displays the 'I2C Self-Test Example' project details, including an overview, how to use the example, and hardware requirements.

New Project

Create project using template i2c_self_test

I2C Self-Test Example

(See the README.md file in the upper level 'examples' directory for more information about examples.)

Overview

This example demonstrates basic usage of I2C driver by running two tasks on I2C bus:

1. Read external I2C sensor, here we take the BH1750 ambient light sensor (GY-30 module) for an example.
2. Use one of the ESP device's I2C port (master mode) to read and write another I2C port (slave mode) in ESP device.

If you have a new I2C application to go (for example, read the temperature data from external sensor with I2C interface), try this as a basic template, then add your own code.

How to use example

Hardware Required

To run this example, you should have one ESP development board (e.g. ESP32-WROVER Kit) or ESP core board (e.g. ESP32-DevKitC). Optionally, you can also connect an external sensor. Here we choose the BH1750 just as an example. BH1750 is a digital ambient light sensor. For more

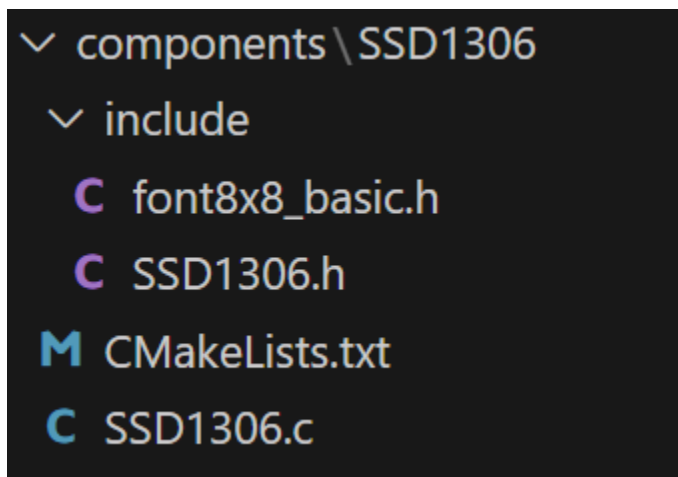
- Thay đổi giá trị của `I2C_EXAMPLE_MASTER_SCL_IO = 22` và `I2C_EXAMPLE_MASTER_SDA_IO = 21` là các chân GPIO tương ứng trên ESP32 theo sơ đồ chân GPIO của ESP32-WROOM32 và thay đổi đối số truyền vào hàm `I2C_MASTER_NUM I2C_NUMBER()` thành 0

```

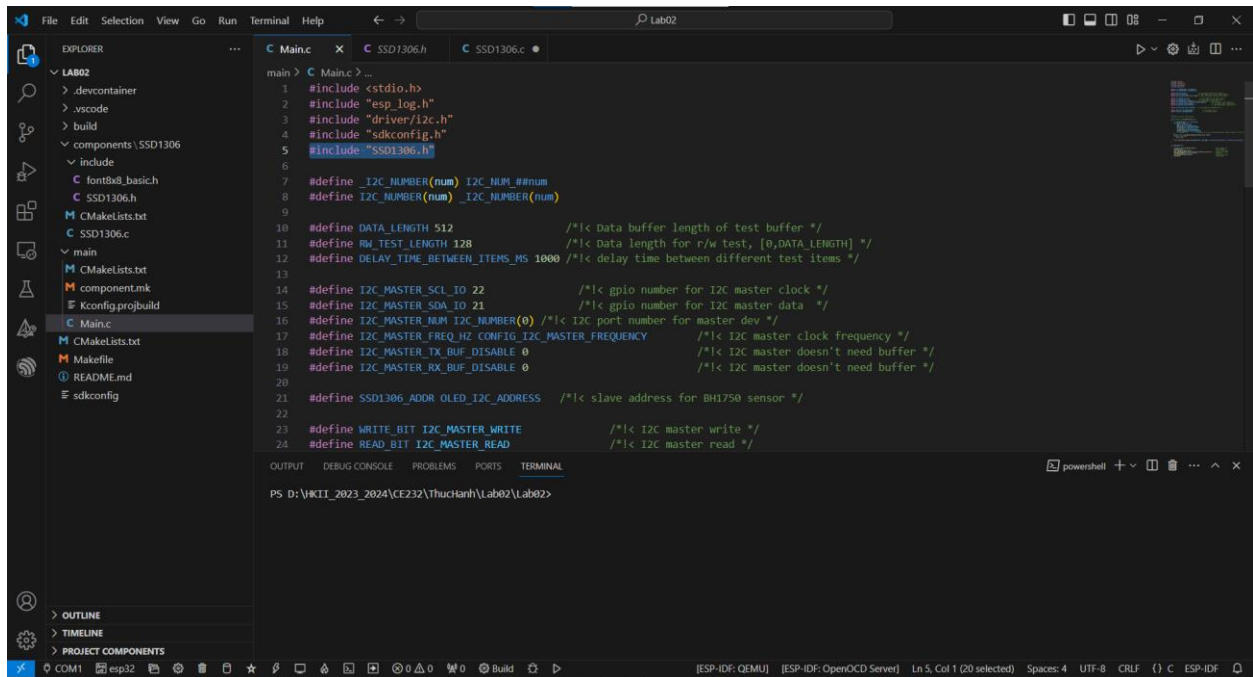
1 #include <stdio.h>
2 #include "esp_log.h"
3 #include "driver/i2c.h"
4 #include "sdkconfig.h"
5 #include "SSD1306.h"
6
7 #define I2C_NUMBER(num) I2C_NUM_#num
8 #define I2C_NUMBER(num) _I2C_NUMBER(num)
9
10 #define DATA_LENGTH 512 /*!< Data buffer length of test buffer */
11 #define RW_TEST_LENGTH 128 /*!< Data length for r/w test, [0,DATA_LENGTH] */
12 #define DELAY_TIME_BETWEEN_ITEMS_MS 1000 /*!< delay time between different test items */
13
14 #define I2C_MASTER_SCL_IO 22 /*!< gpio number for I2C master clock */
15 #define I2C_MASTER_SDA_IO 21 /*!< gpio number for I2C master data */
16 #define I2C_MASTER_NUM I2C_NUMBER(0) /*!< I2C port number for master dev */
17 #define I2C_MASTER_FREQ_HZ CONFIG_I2C_MASTER_FREQUENCY /*!< I2C master clock frequency */
18 #define I2C_MASTER_TX_BUF_DISABLE 0 /*!< I2C master doesn't need buffer */
19 #define I2C_MASTER_RX_BUF_DISABLE 0 /*!< I2C master doesn't need buffer */
20
21 #define SSD1306_ADDR_OLED_I2C_ADDRESS /*!< slave address for BH1750 sensor */
22
23 #define WRITE_BIT I2C_MASTER_WRITE /*!< I2C master write */
24 #define READ_BIT I2C_MASTER_READ /*!< I2C master read */

```

- Tạo component SSD1306 rồi thêm các file `font8x8_basic.h`, `SSD1306.h` và `SSD1306.c` vào.

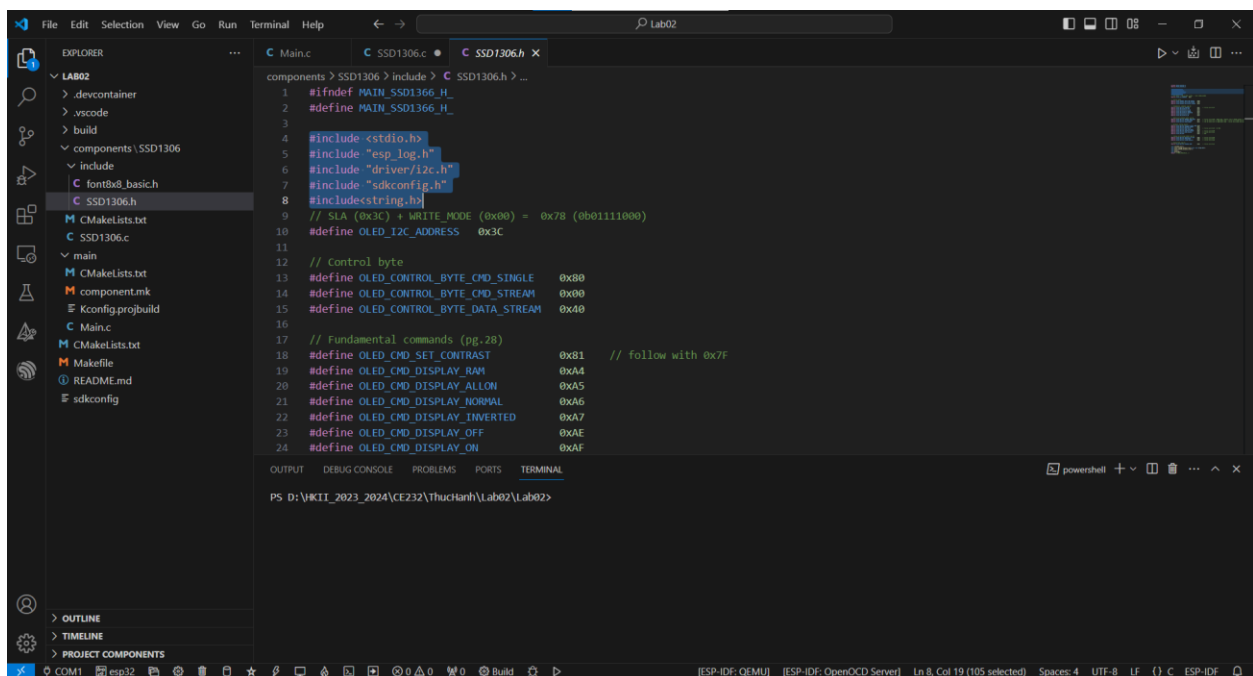


- Trong file Main.c include các thư viện cần thiết như sau:

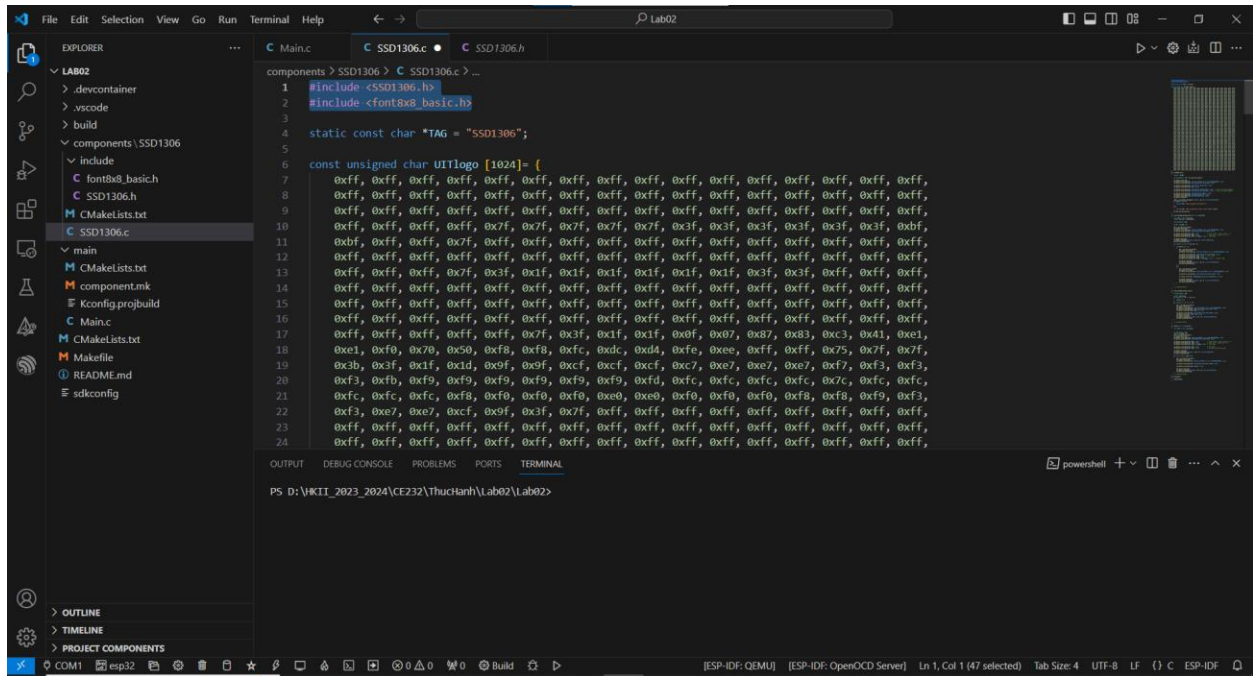


```
1 #include <stdio.h>
2 #include "esp_log.h"
3 #include "driver/i2c.h"
4 #include "sdkconfig.h"
5 #include "SSD1306.h"
6
7 #define I2C_NUMBER(num) I2C_NUM_#num
8 #define I2C_NUMBER(num) _I2C_NUMBER(num)
9
10 #define DATA_LENGTH 512 /*!< Data buffer length of test buffer */
11 #define RW_TEST_LENGTH 128 /*!< Data length for r/w test, [0,DATA_LENGTH] */
12 #define DELAY_TIME_BETWEEN_ITEMS_MS 1000 /*!< delay time between different test items */
13
14 #define I2C_MASTER_SCL_IO 22 /*!< gpio number for I2C master clock */
15 #define I2C_MASTER_SDA_IO 21 /*!< gpio number for I2C master data */
16 #define I2C_MASTER_NUM I2C_NUMBER(0) /*!< I2C port number for master dev */
17 #define I2C_MASTER_FREQ_HZ CONFIG_I2C_MASTER_FREQUENCY /*!< I2C master clock frequency */
18 #define I2C_MASTER_TX_BUF_DISABLE 0 /*!< I2C master doesn't need buffer */
19 #define I2C_MASTER_RX_BUF_DISABLE 0 /*!< I2C master doesn't need buffer */
20
21 #define SSD1306_ADDR_OLED_I2C_ADDRESS /*!< slave address for BH1750 sensor */
22
23 #define WRITE_BIT I2C_MASTER_WRITE /*!< I2C master write */
24 #define READ_BIT I2C_MASTER_READ /*!< I2C master read */
25
26 PS D:\HKIT_2023_2024\CE232\Thuchanh\Lab02\Lab02>
```

- Tương tự trong file SSD1306.h và SSD1306.c:



```
1 #ifndef MAIN_SSD1306_H_
2 #define MAIN_SSD1306_H_
3
4 #include <stdio.h>
5 #include "esp_log.h"
6 #include "driver/i2c.h"
7 #include "sdkconfig.h"
8 #include <string.h>
9
10 // SLA (0x3C) + WRITE_MODE (0x00) = 0x78 (0b01110000)
11 #define OLED_I2C_ADDRESS 0x3C
12
13 // Control byte
14 #define OLED_CONTROL_BYTE_CMD_SINGLE 0x80
15 #define OLED_CONTROL_BYTE_CMD_STREAM 0x00
16 #define OLED_CONTROL_BYTE_DATA_STREAM 0x40
17
18 // Fundamental commands (pg.28)
19 #define OLED_CMD_SET_CONTRAST 0x81 // follow with 0x7F
20 #define OLED_CMD_DISPLAY_RAM 0xA4
21 #define OLED_CMD_DISPLAY_ALLON 0xA5
22 #define OLED_CMD_DISPLAY_NORMAL 0xA6
23 #define OLED_CMD_DISPLAY_INVERTED 0xA7
24 #define OLED_CMD_DISPLAY_OFF 0xAE
25 #define OLED_CMD_DISPLAY_ON 0xAF
26
27 PS D:\HKIT_2023_2024\CE232\Thuchanh\Lab02\Lab02>
```



Bài tập 2: Sử dụng font, thư viện và các hàm cho sẵn theo tài nguyên đính kèm, viết chương trình hiển thị MSSV của các thành viên trong nhóm lên OLED SSD1306. Lưu ý, cần giải thích rõ và cách hoạt động của các hàm hiển thị, ý nghĩa của các command trên SSD1306?

```
void app_main(void)
{
    ESP_ERROR_CHECK(i2c_master_init());           //Innit master i2c
    ssd1306_init();                               //Init ssd1306
    task_ssd1306_display_clear();                 //clear screen oled
    task_ssd1306_display_text("21520684\n21520697\n21520763\n"); //display id student
}
```

- ESP_ERROR_CHECK() là một macro hoặc hàm dùng để kiểm tra lỗi. Nó thường được sử dụng để kiểm tra các hàm trả về có trạng thái lỗi hay không và xử lý lỗi nếu có.
- Hàm i2c_master_init() được gọi để khởi tạo giao diện I2C dưới dạng master (chế độ điều khiển) trên thiết bị.
- Hàm ssd1306_init() được gọi để khởi tạo màn hình OLED loại SSD1306. Hàm này chuẩn bị màn hình để sử dụng.

```

void ssd1306_init()
{
    esp_err_t espRc;

    i2c_cmd_handle_t cmd = i2c_cmd_link_create();

    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS << 1) | I2C_MASTER_WRITE, true);
    i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_CMD_STREAM, true);

    i2c_master_write_byte(cmd, OLED_CMD_SET_CHARGE_PUMP, true);
    i2c_master_write_byte(cmd, 0x14, true);

    i2c_master_write_byte(cmd, OLED_CMD_SET_SEGMENT_REMAP, true); // reverse left-right mapping
    i2c_master_write_byte(cmd, OLED_CMD_SET_COM_SCAN_MODE, true); // reverse up-bottom mapping

    i2c_master_write_byte(cmd, OLED_CMD_DISPLAY_NORMAL, true);
    i2c_master_write_byte(cmd, OLED_CMD_DISPLAY_OFF, true);
    i2c_master_write_byte(cmd, OLED_CMD_DISPLAY_ON, true);
    i2c_master_stop(cmd);

    espRc = i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
    if (espRc == ESP_OK)
    {
        ESP_LOGI(TAG, "OLED configured successfully");
    }
    else
    {
        ESP_LOGE(TAG, "OLED configuration failed. code: 0x%.2X", espRc);
    }
    i2c_cmd_link_delete(cmd);
}

```

- “esp_err_t espRc”: Đây là khai báo của một biến để lưu trữ mã lỗi của ESP-IDF (ESP32 IoT Development Framework). Biến này sẽ được sử dụng để kiểm tra kết quả của các hoạt động I2C sau này.
- “i2c_cmd_handle_t cmd = i2c_cmd_link_create()”: Tạo một cấu trúc lệnh I2C mới để lưu trữ các lệnh I2C sắp được thực hiện.
- “i2c_master_start(cmd)”: Gửi tín hiệu khởi đầu trên bus I2C.
- “i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS << 1) | I2C_MASTER_WRITE, true)”: Ghi byte địa chỉ thiết bị OLED vào bus I2C để bắt đầu giao tiếp với thiết bị. Địa chỉ được dịch trái một bit và thêm bit ghi (bit thấp nhất là 0) cho phép ghi vào thiết bị.
- “i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_CMD_STREAM, true)”: Gửi một byte điều khiển cho OLED để xác định dữ liệu sắp được gửi là dữ liệu lệnh (command).

- Các dòng tiếp theo gửi các lệnh cụ thể đến OLED thông qua bus I2C, như “OLED_CMD_SET_CHARGE_PUMP”, “OLED_CMD_SET_SEGMENT_REMAP”, “OLED_CMD_SET_COM_SCAN_MODE”, “OLED_CMD_DISPLAY_NORMAL”, “OLED_CMD_DISPLAY_OFF”, “OLED_CMD_DISPLAY_ON”.
- “i2c_master_stop(cmd)”: Gửi tín hiệu kết thúc trên bus I2C.
- “espRc = i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS)”: Gọi hàm để thực hiện các lệnh đã được lưu trữ trong cấu trúc lệnh I2C. Kết quả của hoạt động này sẽ được gán cho espRc.
- “if (espRc == ESP_OK) {...} else {...}”: Kiểm tra xem việc gửi các lệnh I2C có thành công hay không. Nếu thành công (ESP_OK), một thông báo thông tin (ESP_LOGI) sẽ được ghi vào log, ngược lại một thông báo lỗi (ESP_LOGE) sẽ được ghi với mã lỗi cụ thể.
- “i2c_cmd_link_delete(cmd)”: Xóa cấu trúc lệnh I2C sau khi đã sử dụng xong để giải phóng bộ nhớ.
- Hàm task_ssd1306_display_clear() được gọi để xóa màn hình OLED. Hàm này có thể sẽ xóa toàn bộ nội dung trên màn hình, chuẩn bị cho việc hiển thị nội dung mới.

```

void task_ssd1306_display_clear()
{
    i2c_cmd_handle_t cmd;

    uint8_t clear[128];
    for (uint8_t i = 0; i < 128; i++)
    {
        clear[i] = 0;
    }
    for (uint8_t i = 0; i < 8; i++)
    {
        cmd = i2c_cmd_link_create();
        i2c_master_start(cmd);
        i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS << 1) | I2C_MASTER_WRITE, true);
        i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_CMD_SINGLE, true);
        i2c_master_write_byte(cmd, 0xB0 | i, true);

        i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_DATA_STREAM, true);
        i2c_master_write(cmd, clear, 128, true);
        i2c_master_stop(cmd);
        i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
        i2c_cmd_link_delete(cmd);
    }

    // vTaskDelete(NULL);
}

```

- Khởi tạo một mảng “clear” gồm 128 phần tử, với mỗi phần tử được đặt giá trị là 0. Điều này tương ứng với việc xóa một dòng trên màn hình OLED, với mỗi dòng gồm 128 pixel.
- Dùng vòng lặp để duyệt qua mỗi dòng trên màn hình OLED (có tổng cộng 8 dòng).
- Trong mỗi lần lặp, tạo một lệnh I2C mới để gửi các tín hiệu điều khiển và dữ liệu đến màn hình OLED.
- Gửi các tín hiệu điều khiển đến màn hình OLED để thiết lập vị trí cần xóa. Trong trường hợp này, lệnh “i2c_master_write_byte()” được sử dụng để gửi byte điều khiển đến OLED.
 “OLED_CONTROL_BYTE_CMD_SINGLE” là một byte đặc biệt để chỉ định rằng byte tiếp theo sẽ là một lệnh. “0xB0 | i” được gửi để chọn dòng cần xóa, trong đó “i” là biến đếm của vòng lặp.
- Gửi tín hiệu dữ liệu đến màn hình OLED để ghi dữ liệu xóa. Tương tự như bước trước, “i2c_master_write_byte()” được sử dụng để gửi byte điều khiển, và “OLED_CONTROL_BYTE_DATA_STREAM” được

sử dụng để chỉ định byte tiếp theo sẽ là dữ liệu. Sau đó, lệnh “i2c_master_write()” được sử dụng để gửi dữ liệu từ mảng “clear” đến màn hình OLED.

- Kết thúc lệnh I2C và gửi nó đi thông qua “i2c_master_cmd_begin()”.
- Xóa lệnh I2C để giải phóng bộ nhớ được cấp phát cho lệnh.
- Quay lại bước 3 để xóa các dòng còn lại trên màn hình OLED.

- Hàm `task_ssd1306_display_text("")` được gọi để hiển thị văn bản trên màn hình OLED

```
void task_ssd1306_display_text(const void *arg_text)
{
    char *text = (char *)arg_text;
    uint8_t text_len = strlen(text);

    i2c_cmd_handle_t cmd;

    uint8_t cur_page = 0;

    cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS << 1) | I2C_MASTER_WRITE, true);
    i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_CMD_STREAM, true);

    i2c_master_write_byte(cmd, 0x00, true); // reset column - choose column --> 0
    i2c_master_write_byte(cmd, 0x10, true); // reset line - choose line --> 0
    i2c_master_write_byte(cmd, 0xB0 | cur_page, true); // reset page

    i2c_master_stop(cmd);
    i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
    i2c_cmd_link_delete(cmd);

    for (uint8_t i = 0; i < text_len; i++)
    {
        if (text[i] == '\n')
        {
            cmd = i2c_cmd_link_create();
            i2c_master_start(cmd);
            i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS << 1) | I2C_MASTER_WRITE, true);

            i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_CMD_STREAM, true);
            i2c_master_write_byte(cmd, 0x00, true); // reset column
            i2c_master_write_byte(cmd, 0x10, true);
            i2c_master_write_byte(cmd, 0xB0 | ++cur_page, true); // increment page
        }
    }
}
```

```

        i2c_master_stop(cmd);
        i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
        i2c_cmd_link_delete(cmd);
    }
    else
    {
        cmd = i2c_cmd_link_create();
        i2c_master_start(cmd);
        i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS << 1) | I2C_MASTER_WRITE, true);

        i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_DATA_STREAM, true);

        i2c_master_write(cmd, font8x8_basic_tr[(uint8_t)text[i]], 8, true);

        i2c_master_stop(cmd);
        i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
        i2c_cmd_link_delete(cmd);
    }
}

// vTaskDelete(NULL);
}

```

- “const void *arg_text”: Hàm này nhận vào một con trỏ không thay đổi trỏ đến văn bản cần hiển thị trên màn hình. Kiểu const void * cho phép truyền vào bất kỳ dữ liệu nào, nhưng ở đây nó được ép kiểu thành char * vì chúng ta biết rằng nó là một chuỗi ký tự.
- “char *text = (char *)arg_text”: Đoạn mã này ép kiểu con trỏ không thay đổi arg_text thành một con trỏ tới một mảng ký tự (chuỗi) để xử lý văn bản.
- “uint8_t text_len = strlen(text)”: Đoạn mã này tính độ dài của chuỗi ký tự text.
- “for (uint8_t i = 0; i < text_len; i++) { ... }”: Vòng lặp này duyệt qua từng ký tự trong chuỗi text.
- Trong vòng lặp, nếu ký tự là \n (dấu xuống dòng), nó sẽ tăng trang lên và điều chỉnh cột và dòng của màn hình. Nếu không, nó sẽ hiển thị ký tự đó trên màn hình.
- Đoạn mã điều khiển I2C để gửi các byte dữ liệu đến màn hình OLED. Điều này bao gồm lựa chọn cột, dòng và trang, sau đó gửi dữ liệu ký tự hoặc lệnh điều khiển đến màn hình.

Bài tập 3: Vẽ logo UIT lên OLED SSD1306. Giải thích cách làm và kết quả?



- Tạo logo UIT với định dạng bitmap:

```
const unsigned char UITlogo [1024]= {
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0x7f, 0x7f, 0x7f, 0x7f, 0x7f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0xbf,
    0xbf, 0xff, 0xff, 0x7f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0x7f, 0x3f, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f, 0x3f, 0x3f, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0x7f, 0x3f, 0x1f, 0x0f, 0x07, 0x87, 0x83, 0xc3, 0x41, 0xe1,
    0xe1, 0xf0, 0x70, 0x50, 0xf8, 0xf8, 0xfc, 0xdc, 0xd4, 0xfe, 0xee, 0xff, 0xff, 0x75, 0x7f, 0x7f,
    0x3b, 0x3f, 0x1f, 0x1d, 0x9f, 0x9f, 0xcf, 0xcf, 0xcf, 0xc7, 0xe7, 0xe7, 0xe7, 0xf7, 0xf3, 0xf3,
    0xf3, 0xfb, 0xf9, 0xf9, 0xf9, 0xf9, 0xf9, 0xf9, 0xfd, 0xfc, 0xfc, 0xfc, 0xf8, 0xf8, 0xfc, 0xfc,
    0xfc, 0xfc, 0xfc, 0xf8, 0xf0, 0xf0, 0xf0, 0xe0, 0xe0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf9, 0xf9, 0xf3,
    0xf3, 0xe7, 0xe7, 0xcf, 0x9f, 0x3f, 0x7f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0x7f, 0xf7, 0xf3, 0x51, 0x78, 0xbc, 0xf4, 0xd6, 0xfe, 0xef, 0xf5, 0xf7, 0x7f, 0x7f, 0x3d, 0x1f,
    0x1f, 0x1f, 0x8f, 0xcf, 0xc7, 0xc7, 0xe3, 0xf3, 0xf1, 0xf1, 0xf9, 0xf8, 0xfc, 0xfc, 0xfe, 0xfe,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xf7, 0xef, 0xbf, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf8, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0x7d, 0x3d, 0x1e, 0x1f, 0x0f, 0x07, 0x07, 0x83, 0xc1, 0xe1, 0xf0, 0xf0, 0xf8, 0xfc, 0xfc, 0xfe,
    0xff, 0xff, 0xff, 0x3f, 0x1f, 0x07, 0x83, 0xc1, 0xe0, 0x70, 0x70, 0x38, 0x1c, 0x0d, 0x0f, 0x07,
    0x07, 0x07, 0x07, 0x0d, 0x1d, 0x18, 0x38, 0x70, 0xe0, 0xe1, 0xc3, 0x87, 0x0f, 0x1f, 0x7f,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xfc, 0x9f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xc0, 0xe0, 0xf0, 0xf8, 0xfe, 0xff, 0xff, 0xff, 0xe7, 0xc3, 0x83, 0x83, 0x83, 0x83, 0xff,
    0xff, 0xff, 0xfb, 0xff, 0x9f, 0x8f, 0x87, 0x87, 0x86, 0xc6, 0xe6, 0xee, 0xfc, 0xfc, 0xfc,
```

- Hàm vẽ logo UIT:

```
void draw(const void *arg_text)
{
    char *text = (char *)arg_text;

    i2c_cmd_handle_t cmd;
    uint8_t cur_page = 0;
    cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS << 1) | I2C_MASTER_WRITE, true);
    i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_CMD_STREAM, true);

    i2c_master_write_byte(cmd, 0x00, true); // reset column - choose column --> 0
    i2c_master_write_byte(cmd, 0x10, true); // reset line - choose line --> 0
    i2c_master_write_byte(cmd, 0xB0 | cur_page, true); // reset page

    i2c_master_write_byte(cmd, 0x20, true); // set memory
    i2c_master_write_byte(cmd, 0x00, true); // Horizontal addressing mode

    i2c_master_stop(cmd);
    i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
    i2c_cmd_link_delete(cmd);

    for (int i = 0; i < 1024; i++)
    {
        cmd = i2c_cmd_link_create();
        i2c_master_start(cmd);
        i2c_master_write_byte(cmd, (OLED_I2C_ADDRESS << 1) | I2C_MASTER_WRITE, true);

        i2c_master_write_byte(cmd, OLED_CONTROL_BYTE_DATA_STREAM, true);

        i2c_master_write_byte(cmd, text[i], true);

        i2c_master_stop(cmd);
        i2c_master_cmd_begin(I2C_NUM_0, cmd, 10 / portTICK_PERIOD_MS);
        i2c_cmd_link_delete(cmd);
    }
    // vTaskDelete(NULL);
}
```

- Hàm này nhận đầu vào là một con trỏ void tới mảng lưu trữ bitmap cần vẽ (arg_text). Đầu tiên, chuyển đổi con trỏ này thành một con trỏ char (text) để dễ dàng truy cập từng ký tự trong văn bản.
- Tiếp theo, tạo một biến cmd kiểu “i2c_cmd_handle_t” để lưu trữ các lệnh I2C.
- Sau đó, bắt đầu một chuỗi lệnh I2C bằng cách gửi một tín hiệu start và gửi địa chỉ của màn hình OLED thông qua giao tiếp I2C. Lệnh “i2c_master_write_byte” được sử dụng để gửi các byte dữ liệu theo định dạng của giao thức I2C.

- Tiếp theo, gửi một loạt các lệnh điều khiển để cấu hình màn hình OLED như đặt cột, đặt dòng, và đặt trang.
- Sau đó, gửi các lệnh để cấu hình chế độ địa chỉ của bộ nhớ trong màn hình OLED.
- Sau khi cấu hình xong, bắt đầu gửi dữ liệu của mảng vào màn hình OLED. Lặp qua từng ký tự trong mảng và gửi chúng dưới dạng byte thông qua giao tiếp I2C.
- Cuối cùng, sau khi đã gửi xong tất cả dữ liệu, hàm kết thúc chuỗi lệnh I2C bằng cách gửi một tín hiệu stop.
- Mọi lệnh I2C được gửi thông qua giao thức I2C Master bằng cách sử dụng hàm “i2c_master_cmd_begin” và sau đó giải phóng bộ nhớ cho biến cmd bằng hàm “i2c_cmd_link_delete”.

Link github:

<https://github.com/DangUIT/CE232.O21.git>

Link video kết quả bài tập 2 và bài tập 3:

<https://youtube.com/shorts/R12EX7A1IFc?feature=share>