**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM**

**KHOA ĐIỆN- ĐIỆN TỬ**

Logo

Description automatically generated**BỘ MÔN KỸ THUẬT MÁY TÍNH VIỄN THÔNG**

**MÔN HỌC HỆ ĐIỀU HÀNH THỜI GIAN THỰC**

ỨNG DỤNG HỆ ĐIỀU HÀNH FREERTOS TRÊN VI ĐIỀU KHIỂN ESP32

|  |  |
| --- | --- |
| SVTH: NHÓM 11m m m m | MSSV: |
| 1. PHẠM HUY HOÀNG | 20119341 |
| 2. CHÂU THÀNH ĐẠT | 20119332 |
| 3. NGUYỄN QUANG BÌNH | 20119063 |
| 4. NGUYỄN THANH DIỆN | 20119329 |
| 5. NGUYỄN HUY LONG | 20119083 |
| 6. NGUYỄN THỊ YẾN | 20119391 |

GVHD: ThS HUỲNH HOÀNG HÀ

Lớp: Chiều thứ 5 – Tiết 1 - 4

**NHẬN XÉT CỦA GIÁO VIÊN**

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

.......................................................................................................................................

*Tp. Hồ Chí Minh, ngày…. tháng….. năm 2023*

Giáo viên hướng dẫn

Huỳnh Hoàng Hà

**BẢNG PHÂN CHIA CÔNG VIỆC**

|  |  |  |
| --- | --- | --- |
| **STT** | **Tên** | **Công việc thực hiện** |
| 1 | Châu Thành Đạt | - Bài tập 2 + Bài tập 3 |
| 2 | Nguyễn Quang Bình | - Bài tập 2 + Bài tập 3 |
| 3 | Nguyễn Thanh Diện | - Bài tập 2 + Bài tập 3 |
| 4 | Phạm Huy Hoàng | - Bài tập 2 + Bài tập 3 |
| 5 | Nguyễn Huy Long | - Tổng hợp lý thuyết + Bài tập 1 |
| 6 | Nguyễn Thị Yến | - Tổng hợp lý thuyết + Bài tập 1 |

Mục lục

# CHƯƠNG 1: GIỚI THIỆU HỆ ĐIỀU HÀNH FREERTOS SỬ DỤNG CHO VI XỬ LÍ ESP32

## 1.1 Đặc điểm của hệ điều hành FreeRTOS

Một số đặc điểm quan trọng của FreeRTOS:

* Hỗ trợ nhiều nền tảng: FreeRTOS có thể chạy trên nhiều kiến trúc vi điều khiển phổ biến, bao gồm ARM, AVR, PIC, MSP430, và nhiều nền tảng khác.
* Thiết kế nhỏ gọn: FreeRTOS được thiết kế để sử dụng tài nguyên hạn chế, đặc biệt là bộ nhớ, một cách hiệu quả. Kích thước mã thực thi của FreeRTOS nhỏ và yêu cầu ít bộ nhớ RAM.
* Thời gian thực: FreeRTOS được thiết kế để hỗ trợ ứng dụng thời gian thực, nơi các nhiệm vụ cần thực hiện trong khoảng thời gian cụ thể. Nó cung cấp các cơ chế lập lịch và ưu tiên để đảm bảo các nhiệm vụ quan trọng được thực hiện đúng thời hạn.
* Hỗ trợ đa nhiệm: FreeRTOS cho phép chạy nhiều nhiệm vụ đồng thời thông qua việc sử dụng hệ thống ngắt (interrupt-driven). Các nhiệm vụ có thể được tạo, xóa, tạm dừng và tiếp tục theo yêu cầu.
* Quản lý tài nguyên: FreeRTOS cung cấp cơ chế quản lý tài nguyên để chia sẻ và bảo vệ tài nguyên giữa các nhiệm vụ. Các tài nguyên như bộ nhớ, hệ thống tập tin, thiết bị ngoại vi có thể được quản lý an toàn và hiệu quả.
* Bảo mật: FreeRTOS hỗ trợ các cơ chế bảo mật như kiểm soát truy cập, phân quyền và xử lý lỗi. Điều này giúp bảo vệ hệ thống khỏi các tấn công và lỗi không mong muốn.
* Cộng đồng và tài liệu phong phú: FreeRTOS có một cộng đồng sôi nổi và nhiều tài liệu hữu ích. Cộng đồng này cung cấp hỗ trợ, tài liệu, ví dụ và các công cụ hữu ích để phát triển ứng dụng sử dụng FreeRTOS.

## 1.2 Xung nhịp hệ điều hành

Trong FreeRTOS, "tick rate" đề cập đến tần số vòng lặp cơ bản của hệ thống, được đo bằng một đơn vị thời gian được gọi là "tick". Tick rate quyết định tần suất các hoạt động quan trọng như lập lịch và cập nhật trạng thái của các nhiệm vụ trong hệ thống.

Tick rate có thể được cấu hình trong FreeRTOS theo yêu cầu cụ thể của ứng dụng. Thông thường, tick rate được thiết lập trong tệp cấu hình của FreeRTOS (FreeRTOSConfig.h) thông qua macro configTICK\_RATE\_HZ.

Ví dụ:



Chu kì tích rate T = 1/1000 = 1ms

Điều này có nghĩa là hệ thống sẽ cập nhật trạng thái và lập lịch các nhiệm vụ mỗi millisecond. Giá trị cụ thể của tick rate phụ thuộc vào yêu cầu của ứng dụng và cấu hình phần cứng.

## 1.3 Cấu hình chế độ độc quyền và không độc quyền

Để cấu hình độc quyền trong thư viện “FreeRTOSConfig.h”, ta cấu hình đoạn mã #define configUSE\_PREEMPTION như sau:

‘0’- Độc quyền: là độc chiếm CPU cho đến khi thực hiện xong hoặc rơi vào trạng thái blocked

‘1’- Không độc quyền: task đang chạy có thể bị chiếm quyền CPU nếu có task khác có độ ưu tiên cao hơn xuất hiện





## 1.4 Các chiến lược điều phối

### 1.4.1 Điều phối bằng chiến lược round robin(độ ưu tiên bằng nhau)

Chiến lược round-robin là một trong những chiến lược lập lịch cơ bản trong hệ điều hành. Chiến lược này sử dụng cơ chế chia sẻ thời gian để lập lịch thực hiện các tác vụ (processes hoặc threads). Các tác vụ sẽ được thực hiện theo một thứ tự chu kỳ, trong đó mỗi tác vụ được thực hiện trong một khoảng thời gian nhất định (quantum) trước khi chuyển sang tác vụ tiếp theo.

Cụ thể, trong chiến lược round-robin, hệ thống sẽ tạo một danh sách các tác vụ cần thực hiện và lập lịch chúng theo một thứ tự nhất định. Khi một tác vụ được kích hoạt, nó sẽ được thực hiện trong một khoảng thời gian ngắn (quantum), sau đó bị bắt đầu chuyển sang tác vụ tiếp theo trong danh sách. Quá trình này lặp lại cho đến khi tất cả các tác vụ được thực hiện.

Ví dụ sử dụng chiến lược round robin để điều phối

#include <Arduino.h>

#include "stdio.h"

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_log.h"

#include "driver/gpio.h"

#include "FreeRTOSConfig.h"

void task1(void \*pv);

void task2(void \*pv);

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

  xTaskCreatePinnedToCore(task1, "Task 1", 1000, NULL, 1, NULL, 0);

  xTaskCreatePinnedToCore(task2, "Task 2", 1000, NULL, 1, NULL, 0);

}

void loop() {

  // put your main code here, to run repeatedly:

}

void task1(void \*pv) {

  while(1) {

    Serial.println("Task1 is Running");

  }

}

void task2(void \*pv) {

  while(1) {

    Serial.println("Task2 is Running");

  }

}

Kết quả:

Table

Description automatically generated

Trong trường hợp này, khi tác vụ Task 1 được thực hiện, nó sẽ in ra chuỗi "Task1 is Running", và sau đó chuyển sang tác vụ Task 2. Tương tự, khi tác vụ Task 2 được thực hiện, nó sẽ in ra chuỗi "Task2 is Running" và sau đó chuyển lại tác vụ Task 1. Quá trình này tiếp tục lặp lại với mỗi tác vụ được thực hiện trong khoảng thời gian quantum.

**1.4.2 Điều phối bằng độ ưu tiên**

**\* Không độc quyền:**

Ví dụ với đoạn code sau:

#include <Arduino.h>

#include "stdio.h"

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_log.h"

#include "driver/gpio.h"

#include "FreeRTOSConfig.h"

void task1(void \*pvParameters);

void task2(void \*pvParameters);

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

  xTaskCreatePinnedToCore(task1, "Task 1", 1000, NULL, 1, NULL, 0);

  xTaskCreatePinnedToCore(task2, "Task 2", 1000, NULL, 2, NULL, 0);

}

void loop() {

  // put your main code here, to run repeatedly:

}

void task1(void \*pvParameters) {

  while (1) {

    Serial.println("Task 1 is running");

    vTaskDelay(pdMS\_TO\_TICKS(1000)); // Delay 1 second

  }

}

void task2(void \*pvParameters) {

  while (1) {

    Serial.println("Task 2 is running");

    vTaskDelay(pdMS\_TO\_TICKS(500)); // Delay 0.5 second

  }

}

**Kết quả:**

Text

Description automatically generated with medium confidence

Trong ví dụ này, chúng ta tạo hai tác vụ task1 và task2. task1 có độ ưu tiên là 1 và task2 có độ ưu tiên là 2. Điều này đảm bảo rằng task2 có độ ưu tiên cao hơn và sẽ được ưu tiên thực hiện trước task1.

Hàm vTaskDelay() được sử dụng để tạo ra một khoảng thời gian trễ giữa các lần chạy của các tác vụ. Trong ví dụ này, task1 được chạy mỗi 1 giây và task2 được chạy mỗi 0.5 giây.

**\* Độc quyền:**

Ví dụ với code sau:

#include <Arduino.h>

#include "stdio.h"

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_log.h"

#include "driver/gpio.h"

#include "FreeRTOSConfig.h"

void task1(void \*pvParameters);

void task2(void \*pvParameters);

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

  xTaskCreatePinnedToCore(task1, "Task 1", 1000, NULL, 1, NULL, 0);

  xTaskCreatePinnedToCore(task2, "Task 2", 1000, NULL, 2, NULL, 0);

}

void loop() {

  // put your main code here, to run repeatedly:

}

void task1(void \*pvParameters) {

  while (1) {

    Serial.println("Task 1 is running");

  }

}

void task2(void \*pvParameters) {

 vTaskDelay(1000);

  while (1) {

    Serial.println("Task 2 is running");

  }

}

Kết quả:

Graphical user interface

Description automatically generated with medium confidence

Từ kết quả cho thấy task 2 có độ ưu tiên cao hơn task, khi task 1 đang chạy nhưng đến thời gian vào của task 2, thì ngay lập tức trả CPU cho task 2 hoạt động, Task 2 không có được đưa vào trạng thái sleep nên sẽ chạy cho đến hết chương trình.

## 1.5 Suspend và Resume FreeRTOS

Trong FreeRTOS, hàm vTaskSuspend() và vTaskResume() được sử dụng để tạm dừng và tiếp tục thực thi của một task. Dưới đây là một số thông tin chi tiết về cách chúng hoạt động trong FreeRTOS:

**Hàm vTaskSuspend():**

* Cú pháp: vTaskSuspend(TaskHandle\_t xTaskToSuspend);
* Mô tả: Hàm này dùng để tạm dừng thực thi của một task. Task được tạm dừng sẽ chuyển sang trạng thái tạm dừng và không tiếp tục thực thi cho đến khi được phục hồi.
* Tham số xTaskToSuspend: Handle của task muốn tạm dừng. Nếu truyền giá trị NULL, task hiện tại sẽ bị tạm dừng.

Một task không thể tự tạm dừng chính nó. Nếu muốn tạm dừng task hiện tại, hãy sử dụng vTaskSuspend(NULL).

**Hàm vTaskResume():**

* Cú pháp: void vTaskResume(TaskHandle\_t xTaskToResume);
* Mô tả: Hàm này dùng để tiếp tục thực thi của một task đã bị tạm dừng trước đó.
* Tham số xTaskToResume: Handle của task muốn tiếp tục thực thi.

Nếu task đang ở trạng thái tạm dừng, khi được phục hồi, task sẽ tiếp tục thực thi từ điểm tạm dừng. Task sẽ không thực thi lại từ đầu.

## 1.6 Semaphore FreeRTOS

Trong FreeRTOS, Semaphore là một loại cấu trúc dữ liệu được sử dụng để đồng bộ hóa truy cập vào tài nguyên chia sẻ giữa các task (nhiệm vụ) trong hệ thống nhúng.

FreeRTOS cung cấp hai loại Semaphore là **Binary Semaphore** (Semaphore nhị phân) và **Counting Semaphore** (Semaphore đếm).

* **Binary Semaphore:** Binary Semaphore trong FreeRTOS chỉ có hai giá trị là 0 và 1. Nó thường được sử dụng để đồng bộ hóa truy cập vào tài nguyên duy nhất giữa hai task, ví dụ như truy cập vào một bộ nhớ đệm hoặc một thiết bị I/O. Một Binary Semaphore được khởi tạo với giá trị ban đầu là 0 và có thể được chuyển đổi giữa trạng thái đang sử dụng (1) và trạng thái chờ (0) bằng các hàm take và give.
* **Counting Semaphore:** Counting Semaphore trong FreeRTOS có thể có giá trị từ 0 trở lên. Nó được sử dụng để quản lý truy cập vào tài nguyên chia sẻ có số lượng lớn hơn hai. Ví dụ, nếu một tài nguyên chia sẻ có thể được truy cập bởi nhiều task cùng một lúc nhưng chỉ cho phép một số lượng giới hạn các task truy cập đồng thời, Counting Semaphore sẽ được sử dụng. Counting Semaphore có thể được khởi tạo với giá trị ban đầu tùy ý và có thể tăng hoặc giảm giá trị của nó bằng các hàm take và give.

A picture containing screenshot, diagram, text, colorfulness

Description automatically generated

# CHƯƠNG 2: ỨNG DỤNG HỆ ĐIỀU HÀNH FREERTOS TRÊN VI ĐIỀU KHIỂN ESP32

Trong phần này, nhóm sử dụng chiến lược điều phối theo độ ưu tiên không độc quyền để điều phối các task, cách cài đặt và điều phối đã nói rõ ở chương 1.

## 2.1 3 task điều khiển led

**\* Sơ đồ nối chân:**

A close-up of a circuit board

Description automatically generated with low confidence

Bảng 1: Sơ đồ nối chân 3 task điều khiển led

|  |  |
| --- | --- |
| Chân trên ESP32 | Đầu ra/Đầu vào |
| 4 | Anode led 1 |
| 5 | Anode led 2 |
| 16 | button 1 |
| 17 | button 2 |
| GND | Chân K của led1 và led 2, chân đầu còn lại của 2 button |

\* Code chương trình:

#include <stdio.h>

#include "sdkconfig.h"

#include "freertos/FreeRTOS.h"

#include "freertos/FreeRTOSConfig.h"

#include "freertos/task.h"

#include "esp\_system.h"

#include "esp\_spi\_flash.h"

#include "driver/gpio.h"

#define led\_blink 4

#define led 5

#define btn1 16

#define btn2 17

TaskHandle\_t xt1;

void configGPIO(){

    //input

    gpio\_config\_t GPIO\_config = {};

    GPIO\_config.pin\_bit\_mask = ((1ULL << led\_blink)|(1ULL <<led) );          /\*!< GPIO pin: set with bit mask, each bit maps to a GPIO \*/

    GPIO\_config.mode = GPIO\_MODE\_OUTPUT;               /\*!< GPIO mode: set input/output mode                     \*/

    GPIO\_config.pull\_up\_en = GPIO\_PULLUP\_DISABLE;       /\*!< GPIO pull-up                                         \*/

    GPIO\_config.pull\_down\_en = GPIO\_PULLDOWN\_DISABLE;   /\*!< GPIO pull-down                                       \*/

    GPIO\_config.intr\_type = GPIO\_INTR\_DISABLE;

    gpio\_config(&GPIO\_config);

    //output

    GPIO\_config.pin\_bit\_mask =((1ULL<< btn1)|(1ULL<<btn2));

    GPIO\_config.mode = GPIO\_MODE\_INPUT;

    GPIO\_config.pull\_up\_en = GPIO\_PULLUP\_ENABLE;

    gpio\_config(&GPIO\_config);

}

void Task1(void \*pvParameters)

{

    while(1)

    {

        gpio\_set\_level(led\_blink, 0);

        vTaskDelay(1000 / portTICK\_PERIOD\_MS);

        gpio\_set\_level(led\_blink, 1);

        vTaskDelay(1000 / portTICK\_PERIOD\_MS);

    }

}

void Task2(void \*pvParameters)

{

    int state\_led = 0;

    int btn1\_state = 0;

    while(1)

    {

        btn1\_state = gpio\_get\_level(btn1);

        while(gpio\_get\_level(btn1)==0){}

        if(btn1\_state == 0)

            {

                printf("task2 is running... \n");

                state\_led = !state\_led;

            }

        gpio\_set\_level(led,state\_led);

        vTaskDelay(10/portTICK\_PERIOD\_MS);

    }

}

void Task3(void \*pvParameters)

{

    int state\_task1 = 0;

    int bnt2\_state = 1;

    while (1)

    {

        bnt2\_state = gpio\_get\_level(btn2);

        while(gpio\_get\_level(btn2)==0){}

        if (bnt2\_state ==0)

        {

            state\_task1 = !state\_task1;

        }

        if(state\_task1 == 0)

        {

            vTaskSuspend(xt1);

            printf("task1 is suspending... \n");

        }

        else if (state\_task1 == 1)

        {

            vTaskResume(xt1);

            printf("task1 is running... \n");

        }

        vTaskDelay(100/portTICK\_PERIOD\_MS);

    }

}

void app\_main(void)

{

    configGPIO();

    xTaskCreate(Task1,"Task1",1024,NULL,3,&xt1);

    xTaskCreate(Task2,"Task2",1024,NULL,2,NULL);

    xTaskCreate(Task3,"Task3",1024,NULL,1,NULL);

}

Đầu tiên, trong hàm configGPIO(), chúng ta thiết lập chế độ và cấu hình cho các chân GPIO. Cụ thể, chân GPIO được gán cho LED sẽ được đặt trong chế độ đầu ra (output), còn chân GPIO được gán cho các nút nhấn sẽ được đặt trong chế độ đầu vào (input) với chế độ kéo lên (pull-up enable).

Tiếp theo, chúng ta khai báo và định nghĩa ba nhiệm vụ (tasks) chính: Task1, Task2, và Task3.

Task1 đơn giản là một vòng lặp vô hạn trong đó LED sẽ nhấp nháy liên tục sau mỗi khoảng thời gian 1 giây. Điều này được thực hiện bằng cách đặt mức logic của chân GPIO điều khiển LED thành 0, chờ 1 giây bằng vTaskDelay(), sau đó đặt mức logic của chân GPIO thành 1 và chờ thêm 1 giây.

Task2 là một nhiệm vụ để kiểm tra trạng thái của nút nhấn btn1 và thay đổi trạng thái của LED dựa trên trạng thái của nút nhấn. Nếu nút nhấn btn1 được nhấn (trạng thái 0), LED sẽ chuyển sang trạng thái ngược lại. Nếu nút nhấn không được nhấn, nhiệm vụ này chỉ đọc trạng thái của nút nhấn và tiếp tục chờ.

Task3 là nhiệm vụ để kiểm tra trạng thái của nút nhấn btn2 và tạm dừng hoặc tiếp tục thực hiện Task1 dựa trên trạng thái của nút nhấn. Nếu nút nhấn btn2 được nhấn (trạng thái 0), nhiệm vụ này sẽ thay đổi trạng thái của Task1. Nếu trạng thái là 0, nhiệm vụ Task1 sẽ bị tạm dừng bằng cách sử dụng vTaskSuspend(). Nếu trạng thái là 1, nhiệm vụ Task1 sẽ được tiếp tục chạy bằng cách sử dụng vTaskResume(). Nếu trạng thái là 1, nhiệm vụ , nhiệm vụ Task1 sẽ tiếp tục bằng cách sử dụng vTaskResume().

Ở hàm app\_main(), chúng ta gọi hàm configGPIO() để cấu hình các chân GPIO như đã mô tả ở trên. Sau đó, chúng ta tạo các nhiệm vụ (Task1, Task2, Task3) bằng cách sử dụng hàm xTaskCreate(). Các nhiệm vụ này được gán cho các tasks có ưu tiên (priority) tương ứng và kích thước ngăn xếp (stack size) là 1024 bytes.

Tiếp theo, chúng ta có một vòng lặp vô hạn trong hàm app\_main(). Trong vòng lặp này, chúng ta sử dụng vTaskDelay() để tạm dừng hàm app\_main() trong 1 giây trước khi tiếp tục thực thi các nhiệm vụ. Điều này đảm bảo rằng hệ thống FreeRTOS có thể lên lịch và thực thi các nhiệm vụ khác một cách chính xác.

**Kết quả thực hiện:** VideoDemo > 3taskled.mp4

**2.2 Hiển thị oled**

**\* Sơ đồ nối chân**

A picture containing text, electronics, screenshot

Description automatically generated

|  |  |
| --- | --- |
| **Chân trên ESP32** | **Chân trên oled** |
| 21 | SDA |
| 22 | SDL |
| GND | GND |
| 3V3 | VCC |

**\* Code thực hiện:**

#include <Arduino.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SH110X.h>

const unsigned char GAU[] PROGMEM = {

  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, 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, 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, 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, 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, 0xe3, 0xff, 0xff, 0x7f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xdd, 0xff, 0xfc, 0x1f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x9e, 0xc0, 0xfd, 0xef, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xbe, 0x00, 0x0d, 0xf7, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xbb, 0xff, 0xf1, 0x37, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xdf, 0xff, 0xff, 0xf7, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xcf, 0xff, 0xff, 0xef, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x9f, 0xff, 0xff, 0xcf, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x1f, 0xff, 0xbf, 0xef, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xfc, 0x6f, 0xfe, 0x0f, 0xef, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xf9, 0xb7, 0xfd, 0xf7, 0xf7, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xfa, 0xd3, 0xf9, 0x43, 0xf7, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xf7, 0xe3, 0xfa, 0xeb, 0xf7, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xf7, 0xfa, 0x33, 0xfb, 0xf3, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xef, 0xf2, 0x37, 0xfb, 0xfb, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xef, 0xf7, 0x77, 0xfb, 0xfb, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xef, 0xf7, 0x77, 0xfb, 0xfb, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xcf, 0xf4, 0x07, 0xfb, 0xfb, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xcf, 0xe4, 0x0f, 0xf9, 0xfb, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xdf, 0xe7, 0xef, 0xfd, 0xf1, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xdf, 0xef, 0xef, 0xfd, 0xed, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xcf, 0xef, 0xe7, 0xfd, 0xbd, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xcf, 0xef, 0xf7, 0xfc, 0xfd, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xef, 0xef, 0xf7, 0xff, 0xfd, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xef, 0xef, 0xf7, 0xff, 0xfd, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xe7, 0xef, 0xf3, 0xff, 0xfd, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xf7, 0xff, 0xfb, 0xff, 0xfd, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xf3, 0xbf, 0xfc, 0xf3, 0xfc, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xfc, 0x3f, 0xfe, 0x07, 0xfc, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x7f, 0xff, 0xff, 0xfe, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x7f, 0xff, 0xff, 0xfe, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x7f, 0xff, 0xff, 0xfe, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,

  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x7f, 0xff, 0xff, 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, 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, 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, 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, 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, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff

};

// Khởi tạo đối tượng màn hình OLED

Adafruit\_SH1106G display(128, 64, &Wire);

void oled\_task(void \*pvParameter){

  // Khởi tạo màn hình OLED

  display.begin(0x3c,true );

  // Đặt màu cho chữ

  display.setTextColor(SH110X\_WHITE);

  // Đặt font\_size nhỏ nhất

  display.setTextSize(1);

  while(true)

  {

    // Tắt điểm ảnh

    display.clearDisplay();

    display.setCursor(0, 0);

    display.println("1. Nguyen Thanh Dien");

    display.setCursor(0, 10);

    display.println("2. Nguyen Quang Binh");

    display.setCursor(0, 20);

    display.println("3. Chau Thanh Dat");

    display.setCursor(0, 30);

    display.println("4. Nguyen Thi Yen");

    display.setCursor(0, 40);

    display.println("5. Nguyen Huy Long");

    display.setCursor(0, 50);

    display.println("6. Pham Huy Hoang");

    display.display();

    delay(3000);

    display.clearDisplay(); // Xóa toàn bộ màn hình trước khi vẽ

    // Vẽ Con Gấu

    display.drawBitmap(0, 0, GAU, 128, 64, SH110X\_WHITE);

    // Hiển thị lên màn hình

    display.display();

    delay(3000);

    vTaskDelay(500);

  }

}

void setup() {

  Wire.begin();

  xTaskCreate(oled\_task, "OLED\_task", 4096, NULL, 1, NULL);

}

void loop() {

  // không cần làm gì trong hàm loop()

}

Với bài tập này chúng tôi sử dụng thư viện Adafruit\_GFX.h và Adafruit\_SH110X.h để hiển thị oled đang sử dụng là SH1106.

Sử dụng một bit map để lưu hình ảnh, ở đây chúng tôi sử dụng bitmap để lưu hình ảnh như sau:



Các công việc thực hiện trong task oled:

* Khởi tạo màn hình OLED với địa chỉ 0x3C.
* Thiết lập màu chữ là trắng (SH110X\_WHITE).
* Đặt kích thước font chữ là nhỏ nhất (size 1).
* Trong một vòng lặp vô hạn:
* Tắt điểm ảnh của màn hình OLED.
* Đặt con trỏ văn bản ở vị trí (0, 0) và in ra dòng chữ "1. Nguyen Thanh Dien".
* Tương tự, in ra các dòng chữ tương ứng cho các tên khác nhau.
* Hiển thị nội dung lên màn hình OLED.
* Chờ 3 giây.
* Xóa toàn bộ nội dung trên màn hình OLED.
* Vẽ một hình ảnh con gấu tại vị trí (0, 0) trên màn hình.
* Hiển thị hình ảnh lên màn hình OLED.
* Chờ 3 giây.
* Đợi 500ms trước khi bắt đầu một vòng lặp mới.
* Công việc này lặp đi lặp lại vô hạn, hiển thị danh sách tên người và sau đó hiển thị hình ảnh con gấu trên màn hình OLED.

Trong hàm main, thực hiện các công việc:

* Bắt đầu giao tiếp I2C bằng cách gọi hàm Wire.begin(). Điều này cần thiết để kết nối với màn hình OLED qua giao thức I2C.
* Sử dụng hàm xTaskCreate() để tạo một task mới có tên là "OLED\_task"

**Kết quả sau khi chạy chương trình trên ESP32 để hiển thị oled như sau**

Graphical user interface

Description automatically generated

A picture containing text, monitor, computer

Description automatically generated

**2.3 Sử dụng nhiều module và đọc giá trị hiển thị lên oled**

**\* Sơ đồ nối chân**

Diagram, schematic

Description automatically generated

\* Code chương trình:

#include <Arduino.h>

#include <stdio.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_log.h"

#include "driver/gpio.h"

#include "FreeRTOSConfig.h"

#include <Adafruit\_GFX.h>

#include <Adafruit\_SH110X.h>

#include <DHT.h>

#include <Adafruit\_Sensor.h>

// Khởi tạo đối tượng màn hình OLED

Adafruit\_SH1106G display(128, 64, &Wire);

#define DHTPIN GPIO\_NUM\_14           // Chân dữ liệu của cảm biến DHT11

#define DHTTYPE DHT11       // Loại cảm biến DHT11

#define FIRESENSOR\_PIN GPIO\_NUM\_4

#define MOTIONSENSOR\_PIN GPIO\_NUM\_5

#define HUMR\_PIN GPIO\_NUM\_26

#define SOUND\_PIN  GPIO\_NUM\_27

#define MQ3\_PIN GPIO\_NUM\_25

#define JOYSTICK\_X GPIO\_NUM\_35     // Chân GPIO cho trục X của joystick

#define JOYSTICK\_Y GPIO\_NUM\_34

DHT dht(DHTPIN, DHTTYPE);

float humidity = 0;

float temperature = 0;

void dht11\_task(void \*pvParameter) {

  for (;;) {

    humidity = dht.readHumidity();

    temperature = dht.readTemperature();

    vTaskDelay(500); // Delay 2 giây

  }

}

int fireState = 0;

void firesensor\_task(void \*pvParameter){

  pinMode(FIRESENSOR\_PIN, INPUT);

  while(1)

  {

    fireState = analogRead(FIRESENSOR\_PIN);

    vTaskDelay(200);

  }

}

int soundState = 0;

String sound = "NO";

void soundsensor\_task(void \*pvParameter){

  pinMode(SOUND\_PIN,INPUT);

  while(1)

  {

    soundState = analogRead(SOUND\_PIN);

    Serial.println(soundState);

    vTaskDelay(200);

  }

}

int HumState;

int percent;

void humr\_task(void \*pvParameter){

  pinMode(HUMR\_PIN,INPUT);

  while(1)

  {

    HumState = analogRead(HUMR\_PIN);

    percent = map(HumState, 0, 1023, 0, 100);

    vTaskDelay(200);

  }

}

int x = 0;

int y = 0;

void joystick\_task(void \*pvParameter) {

  for (;;) {

    x = analogRead(JOYSTICK\_X);

    y = analogRead(JOYSTICK\_Y);

    vTaskDelay(200); // Delay 0.2 giây

  }

}

int mq3 = 0;

const float referenceVoltage = 3.3;

void MQ3\_task(void \*pvParameter){

  while(1)

  {

    mq3 = analogRead(MQ3\_PIN);  // Đọc giá trị từ cảm biến

    // float voltage = mq3 \* referenceVoltage / 4095;  // Chuyển đổi giá trị ADC thành điện áp

    // float ppm = map(voltage, 0.0, referenceVoltage, 0.0, 10000.0);  // Ánh xạ giá trị điện áp sang PPM

    vTaskDelay(200);  // Tần số đọc cảm biến

  }

}

void oled\_task(void \*pvParameter){

  // Khởi tạo màn hình OLED

  display.begin(0x3c,true );

  // Tắt điểm ảnh

  display.clearDisplay();

  // Đặt màu cho chữ

  display.setTextColor(SH110X\_WHITE);

  while(true)

  {

    // Tắt điểm ảnh

    display.clearDisplay();

    display.setCursor(0, 0);

    display.println("MQ3: " + String(mq3));

    display.setCursor(0, 12);

    display.println("JoyX: " + String(x));

    display.setCursor(0, 24);

    display.println("JoyY: " + String(y));

    display.setCursor(0, 36);

    display.println("HumR: " + String(percent));

    display.setCursor(0, 48);

    display.println("Sound: " + String(soundState));

    display.setCursor(64, 0);

    display.println("Tem: " + String(temperature, 1) + "C");

    display.setCursor(64, 12);

    display.println("Hum: " + String(humidity, 1) + "%");

    display.setCursor(64, 24);

    display.println("Fire: " + String(fireState));

    display.display();

    vTaskDelay(500);

  }

}

void setup() {

  Serial.begin(9600);

  Wire.begin();

  dht.begin();

  pinMode(JOYSTICK\_X, INPUT);

  pinMode(JOYSTICK\_Y, INPUT);

  xTaskCreate(firesensor\_task, "firesensor\_task", 4096, NULL, 1, NULL);

  xTaskCreate(soundsensor\_task, "sound\_task", 4096, NULL, 2, NULL);

  xTaskCreate(humr\_task, "humr\_task", 4096, NULL, 3, NULL);

  xTaskCreate(dht11\_task, "dht11\_task", 4096, NULL, 4, NULL);

  xTaskCreate(joystick\_task, "joystick\_task", 4096, NULL, 5, NULL);

  xTaskCreate(MQ3\_task, "MQ3\_task", 4096, NULL, 6, NULL);

xTaskCreate(oled\_task, "OLED\_task", 4096, NULL, 7, NULL);

}

void loop() {

  // không cần làm gì trong hàm loop()

}

Tóm tắt công việc được thực hiện trong chương trình:

* Khai báo và khởi tạo các thư viện và biến cần thiết.
* Khởi tạo đối tượng màn hình OLED với kích thước 128x64 pixels sử dụng giao thức I2C.
* Định nghĩa và khởi tạo chân GPIO cho các cảm biến và joystick.
* Tạo các task để đọc dữ liệu từ cảm biến và joystick:
* dht11\_task: Đọc dữ liệu độ ẩm và nhiệt độ từ cảm biến DHT11.
* firesensor\_task: Đọc giá trị từ cảm biến cháy (fire sensor).
* soundsensor\_task: Đọc giá trị từ cảm biến âm thanh (sound sensor).
* humr\_task: Đọc giá trị từ cảm biến độ ẩm (humidity sensor).
* joystick\_task: Đọc giá trị từ joystick.
* MQ3\_task: Đọc giá trị từ cảm biến khí gas (MQ3).
* oled\_task: Hiển thị các giá trị đọc được từ các cảm biến lên màn hình OLED.
* Trong hàm setup, khởi tạo giao tiếp Serial, giao tiếp I2C, cấu hình cảm biến và joystick, và tạo các task được định nghĩa ở trên.

Kết quả: