

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



HỆ THỐNG NHÚNG (CO3053)

Lab 1: Introduction to ESP32 and ESP-IDF
Lab 2: ESP32 GPIO and FreeRTOS task

Giảng viên hướng dẫn: Vũ Trọng Thiên
Nhóm sinh viên thực hiện - Nhóm 33: Phạm Duy Quang - 2011899
Dương Đức Nghĩa - 2011671

Thành phố Hồ Chí Minh, Tháng 10/2023



Mục lục

1	Github	2
2	Lab 1	2
2.1	Mục tiêu	2
2.2	Các bước thực hiện	2
2.2.1	Cài đặt eps-idf	2
2.2.2	Sao chép hello world project từ file example	2
2.2.3	Build project	2
2.2.4	Kiểm tra port	3
2.2.5	Nạp code	3
2.2.6	Hiện kết quả	4
3	Lab 2	5
3.1	Ý tưởng	5
3.2	Hiện thực	5
3.3	Kết quả	8
3.4	Trả lời câu hỏi	8

1. Github

Link Github: <https://github.com/ULTIMATE-Mystery/Embedded-System-HCMUT-Semester-231>.

2. Lab 1

2.1 Mục tiêu

Biết cách cài đặt Espressif IoT Development Framework, chạy chương trình hello world đầu tiên

2.2 Các bước thực hiện

2.2.1 Cài đặt eps-idf

Tải tại link <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/windows-set-up.html>

2.2.2 Sao chép hello world project từ file example

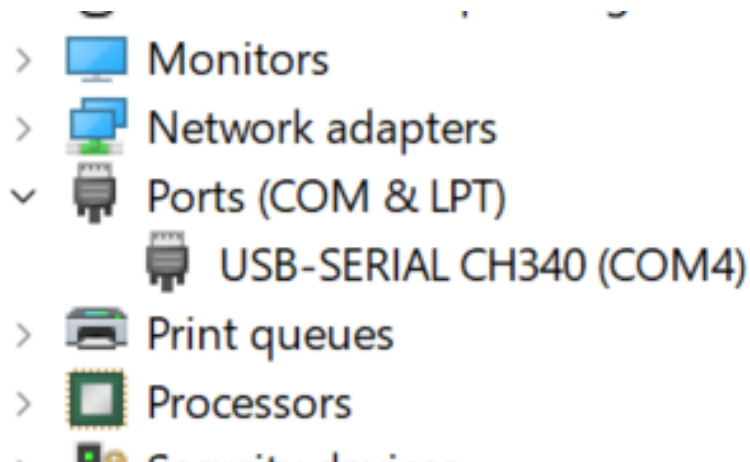
```
1 cd ~/esp
2 cp -r $IDF_PATH/examples/get-started/hello_world .
```

2.2.3 Build project

Sử dụng câu lệnh: idf.py build

```
C:\Espressif\frameworks\esp-idf-v5.1.1>E:
E:\>cd "Hoc tap"
E:\Hoc tap>cd "Lap trinh"
E:\Hoc tap\Lap trinh>cd esp32
E:\Hoc tap\Lap trinh\esp32>cd hello_world
E:\Hoc tap\Lap trinh\esp32\hello_world>idf.py build
Executing action: all (aliases: build)
Running ninja in directory 'E:\Hoc tap\Lap trinh\esp32\hello_world\build'
Executing "ninja all"...
[1/4] cmd.exe /C "cd /D "E:\Hoc tap\Lap trinh\esp32\hello_world\build\hello_world" & "C:\Espressif\frameworks\esp-idf-v5.1.1\components\esptool\python\esptool.py" -p (PORT) -b 460800 --before default_reset --after hard_reset --chip esp32 write_flash --flash_mode dio --flash_size 2MB --flash_freq 40m 0x1000 build\bootloader\bootloader.bin 0x8000 build\partition_table\partition-table.bin 0x10000 build\hello_world.bin
[1/1] cmd.exe /C "cd /D "E:\Hoc tap\Lap trinh\esp32\hello_world\build\bootloader\bootloader" & "C:\Espressif\frameworks\esp-idf-v5.1.1\components\esptool\python\esptool.py" -p (PORT) -b 460800 --before default_reset --after hard_reset --chip esp32 write_flash --flash_mode dio --flash_size 2MB --flash_freq 40m 0x1000 build\bootloader\bootloader.bin 0x8000 build\partition_table\partition-table.bin 0x10000 build\hello_world.bin
Project build complete. To flash, run this command:
C:\Espressif\python_env\idf5.1_py3.11_env\Scripts\python.exe C:\Espressif\frameworks\esp-idf-v5.1.1\components\esptool\python\esptool.py -p (PORT) -b 460800 --before default_reset --after hard_reset --chip esp32 write_flash --flash_mode dio --flash_size 2MB --flash_freq 40m 0x1000 build\bootloader\bootloader.bin 0x8000 build\partition_table\partition-table.bin 0x10000 build\hello_world.bin
or run 'idf.py -p (PORT) flash'
E:\Hoc tap\Lap trinh\esp32\hello_world>
```

2.2.4 Kiểm tra port



Sử dụng COM4 để nạp code

2.2.5 Nạp code

Sử dụng câu lệnh `idf.py -p COM4 flash`

```
_freq 40M --flash_size 2MB 0x1000 bootloader/bootloader.bin 0x10000
le.bin
esptool.py v4.7.dev1
Serial port COM4
Connecting....
Chip is ESP32-D0WD-V3 (revision v3.0)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, C
Crystal is 40MHz
MAC: e0:5a:1b:a0:b9:04
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00001000 to 0x00007fff...
Flash will be erased from 0x00010000 to 0x0003afff...
Flash will be erased from 0x00008000 to 0x00008fff...
Compressed 26640 bytes to 16691...
Writing at 0x00001000... (50 %)
Writing at 0x0000768d... (100 %)
Wrote 26640 bytes (16691 compressed) at 0x00001000 in 0.8 seconds (
Hash of data verified.
Compressed 175440 bytes to 97631...
Writing at 0x00010000... (16 %)
Writing at 0x0001c0b9... (33 %)
Writing at 0x00021aaa... (50 %)
Writing at 0x0002735f... (66 %)
Writing at 0x0002d629... (83 %)
Writing at 0x00035218... (100 %)
```

2.2.6 Hiện kết quả

Sử dụng câu lệnh `idf.py -p COM4 monitor`

```
W (303) spi_flash: Detected size(4096k) larger than the size in the binary image header(2048k). Using
nary image header.
I (317) app_start: Starting scheduler on CPU0
I (322) app_start: Starting scheduler on CPU1
I (322) main_task: Started on CPU0
I (332) main_task: Calling app_main()
Hello world!
This is esp32 chip with 2 CPU core(s), WiFi/BT/BLE, silicon revision v3.0, 2MB external flash
Minimum free heap size: 301252 bytes
Restarting in 10 seconds...
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
```

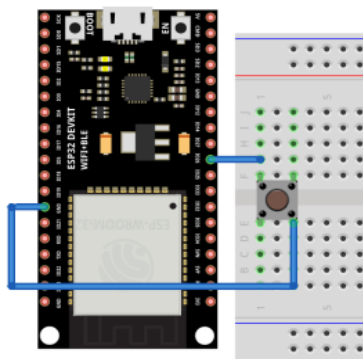
3. Lab 2

3.1 Ý tưởng

- Sẽ có 2 task: in mã số sinh viên và task xử lý tác vụ khi nhấn nút.
- Task in mã số sinh viên: sẽ chạy mỗi 1 giây
- In ra **ESP32** khi nhấn nút:
 - Đầu tiên sẽ có 1 task để in ra "ESP32" mà không có delay.
 - Task này sẽ suspend cho đến khi nhấn nút và cho phép resume.
 - Khi này để thực hiện được, ta cần sử dụng interrupt và debounce button để phát hiện xem button đã nhấn hay chưa.

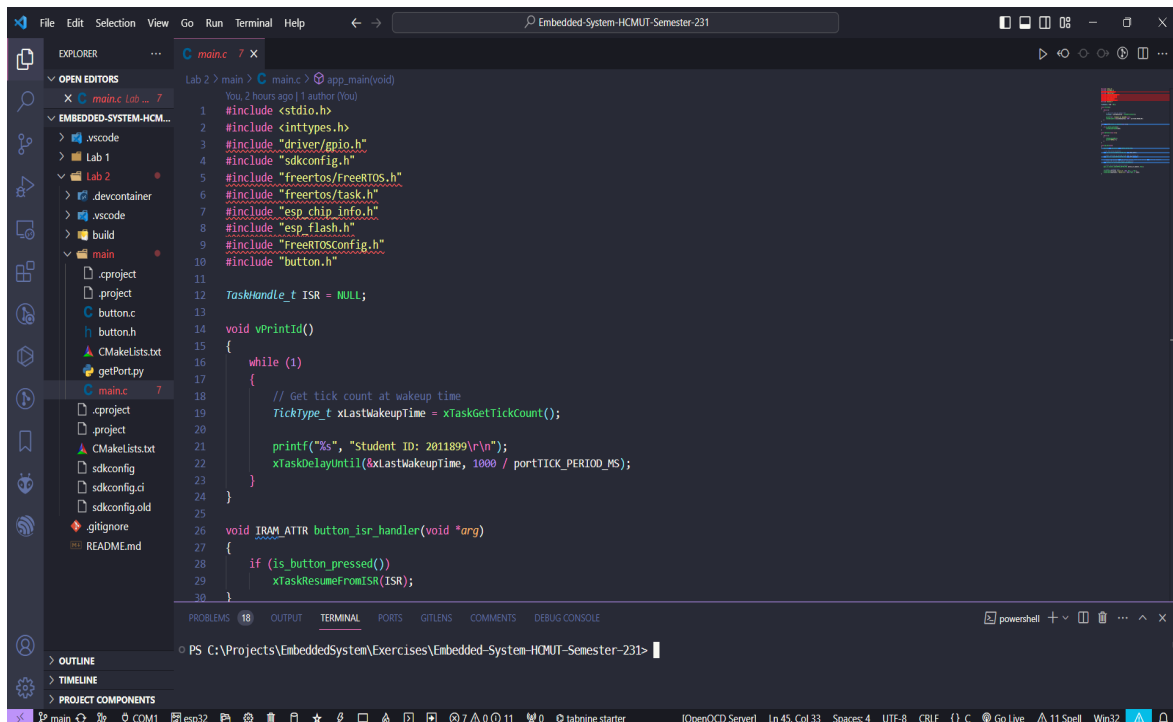
3.2 Hiện thực

- Sơ đồ thiết kế:



Hình 1: Sơ đồ thiết kế

- o Đầu tiên tạo một project mới bằng cách vào VS Code nhập tổ hợp "Ctrl + Shift + P". Sau khi đã thiết lập xong, project được tạo như hình bên dưới:



Hình 2: Tạo project

- o Kế tiếp ta cần xác định chân GPIO để gán input cho nút nhấn (button):

```
1 // GPIO Pins
2 #define BUTTON_PIN 25
```

- o Trong hàm app_main(), cần khởi tạo GPIO Pin cho chân 25. Vì là input dạng nút nhấn và nhóm sử dụng interrupt nên cần đặt cho ESP32 một interrupt khi phát hiện những sự kiện được liệt kê (nhóm sử dụng phát hiện cạnh xuống 1 → 0):

```
1 // Initialize GPIO pins =====
2 esp_rom_gpio_pad_select_gpio(BUTTON_PIN);
3 // =====
4 // Set GPIO directions =====
5 gpio_set_direction(BUTTON_PIN, GPIO_MODE_INPUT);
6 // =====
7 // Enable interrupt on falling (1->0) edge for button pin =====
8 gpio_set_intr_type(BUTTON_PIN, GPIO_INTR_NEGEDGE);
9 // =====
```

- o Sau đó ta cần cài đặt interrupt handle bằng lệnh sau với biến là ESP_INTR_FLAG_DEFAULT = 0, có nghĩa là cài đặt mặc định:

```
1 gpio_install_isr_service(ESP_INTR_FLAG_DEFAULT);
```

- o Thêm hàm để xử lý interrupt cho nút nhấn chân 25 như sau (cụ thể khi có interrupt thì hàm được thêm vào sẽ được gọi):

```
1 gpio_isr_handler_add(BUTTON_PIN, button_gpio_isr_handler, NULL);
```

- o Tạo 2 task như đã đề cập (ở phần 1: "FreeRTOS task" của đề bài) và bắt đầu Scheduler:

```

1 // Create 2 tasks
2 xTaskCreate(vPrintStudentID,      // function
3             "print_Student_ID",  // name to easily debug
4             2048,                  // stack allocate
5             NULL,                  // none parameter
6             1,                     // equal priority
7             &Print_ISR            // none taskHandle
8 );
9 xTaskCreate(vPrintEsp32,          // function
10            "print_ESP32",         // name to easily debug
11            2048,                  // stack allocate
12            NULL,                  // none parameter
13            1,                     // equal priority
14            &Button_ISR           // none taskHandle
15 );
16 // Start Scheduler
17 vTaskStartScheduler(void);

```

Sau đây là phần hiện thực 2 task *vPrintStudentID* và *vPrintEsp32*:

- o *vPrintStudentID*: Khi task này được gọi, đầu tiên nó sẽ tính tick của nó hiện tại và in ra giá trị MSSV và đợi cho đến khi đủ 1000 ms thì mới hoàn thành task và được scheduler gọi lại.

```

1 void vPrintStudentID(void* pvParameters) {
2     while (1) {
3         TickType_t xWakeUpTime = xTaskGetTickCount();
4         printf("Student_ID: 2011899\r\n");
5         xTaskDelayUntil(&xWakeUpTime, 1000 / portTICK_PERIOD_MS);
6     }
7 }

```

- o *vPrintEsp32*: Như đã đề cập ở trên, task này sẽ hoãn cho đến khi nó được tiếp tục bởi interrupt.

```

1 void vPrintEsp32(void* pvParameters) {
2     while (1) {
3         vTaskSuspend(NULL);
4         printf("ESP32\n");
5     }
6 }

```

Sau đây là hàm `button_gpio_isr_handler()`, dùng để tiếp tục task *vPrintEsp32* sau khi đã được xử lý debounce button:

```

1 void IRAM_ATTR button_gpio_isr_handler(void* arg) {
2     if (is_button_pressed()) {
3         xTaskResumeFromISR(vPrintEsp32);
4     }
5 }

```

- o Khi nút nhấn được nhấn: `current_interrupt_time` sẽ được ghi lại và lấy trừ cho interrupt time trước đó, nếu lớn hơn khoảng `tick_difference = (DEBOUNCE_TIME * configTICK_RATE_HZ)/1000` thì trả về 1 còn không thì nó bị bounce và trả về 0.

```

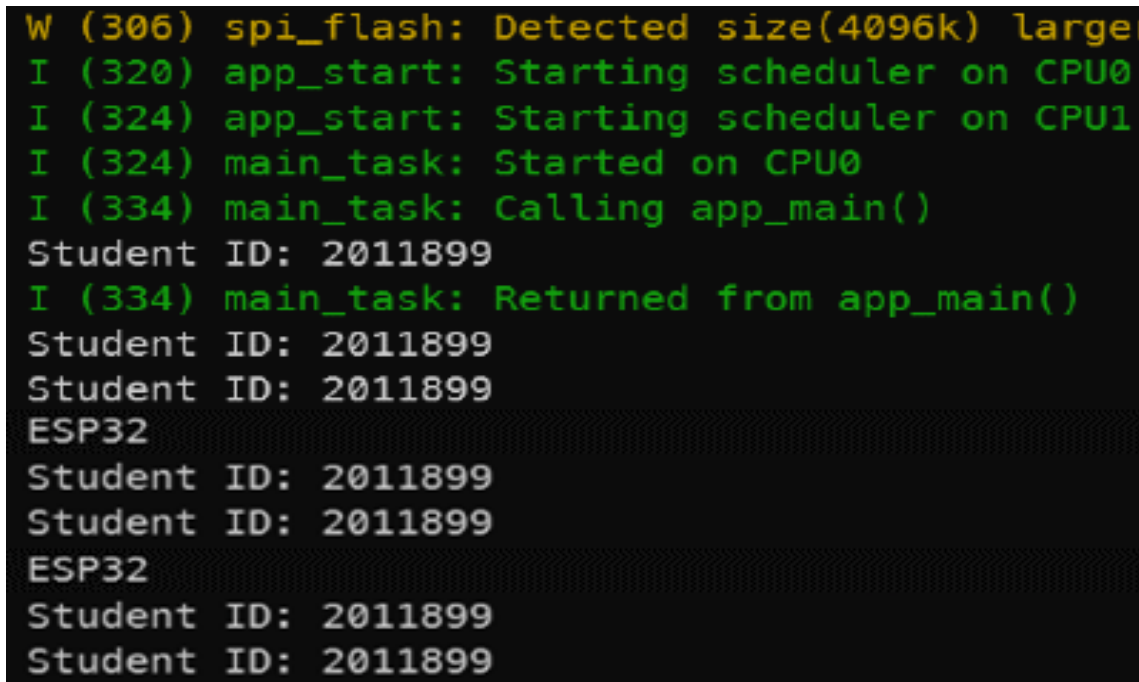
1 int is_button_pressed() {
2     static TickType_t last_interrupt_time = 0;
3     TickType_t current_interrupt_time = xTaskGetTickCount();
4     // If interrupts come faster than the tick_difference, assume it is a bounce and ignore
5     if (current_interrupt_time - last_interrupt_time <
6         (DEBOUNCE_TIME * configTICK_RATE_HZ) / 1000) {
7         return 0;
8     }
9     last_interrupt_time = current_interrupt_time;

```



```
10     return 1;  
11 }
```

3.3 Kết quả



```
W (306) spi_flash: Detected size(4096k) larger  
I (320) app_start: Starting scheduler on CPU0  
I (324) app_start: Starting scheduler on CPU1  
I (324) main_task: Started on CPU0  
I (334) main_task: Calling app_main()  
Student ID: 2011899  
I (334) main_task: Returned from app_main()  
Student ID: 2011899  
Student ID: 2011899  
ESP32  
Student ID: 2011899  
Student ID: 2011899  
ESP32  
Student ID: 2011899  
Student ID: 2011899
```

Hình 3: Kết quả

3.4 Trả lời câu hỏi

- Câu hỏi: ESP-IDF có yêu cầu hàm `vTaskStartScheduler()` để bắt đầu scheduler không?
- Trả lời:
 - **Đối với hệ thống:** Có, gọi hàm `vTaskStartScheduler()` là điều bắt buộc. Nếu hàm này không được gọi thì các task của FreeRTOS cũng sẽ không được thực thi.
 - **Đối với developers:** Không cần gọi vì hệ thống đã tự động gọi hàm `vTaskStartScheduler()` trước khi hàm `app_main()` được thực thi. Vì vậy, chúng ta không cần phải gọi thêm hàm này sau khi đã tạo task. Ta có thể truy tìm hàm `vTaskStartScheduler()` được gọi ở Github Espressif.

```
void esp_startup_start_app(void)
{
    #if CONFIG_ESP_INT_WDT...
    #elif CONFIG_ESP32_EC03_CACHE_LOCK_FIX...
    #endif...
    #if CONFIG_ESP_SYSTEM_GDBSTUB_RUNTIME && !CONFIG_IDF_TARGET_ESP32C2...
    #endif // CONFIG_ESP_SYSTEM_GDBSTUB_RUNTIME

    BaseType_t res = xTaskCreatePinnedToCore(main_task, "main", ...
    assert(res == pdTRUE);
    (void)res;

    /*
    If a particular FreeRTOS port has port/arch specific OS startup behavior, they can implement a function of type
    "void port_start_app_hook(void)" in their `port.c` files. This function will be called below, thus allowing each
    FreeRTOS port to implement port specific app startup behavior.
    */
    void __attribute__((weak)) port_start_app_hook(void);
    if (port_start_app_hook != NULL) {
        port_start_app_hook();
    }

    ESP_EARLY_LOGI(APP_START_TAG, "Starting scheduler on CPU0");
    vTaskStartScheduler();
}
```

Hình 4: *vTaskStartScheduler()* đã được gọi trong hàm *esp_startup_start_app()*