ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC BÁCH KHOA
========== o ==========
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

Embedded System Lab

Lab 2:

ESP32 GPIO and FreeRTOS task

Giảng viên hướng dẫn: Huỳnh Phúc Nghị
Sinh viên thực hiện: Nguyễn Kim Ngọc Vy – 2015112
Nguyễn Phúc Tiến – 2014725
Nguyễn Văn Thịnh – 2014603
Nguyễn Phúc Đăng – 2012968

Thành phố Hồ Chí Minh, tháng 9 năm 2023

Bảng phân công công việc:

MSSV	Tên	Đánh giá
2015112	Nguyễn Kim Ngọc Vy	100%
2014725	Nguyễn Phúc Tiến	100%
2013401	Nguyễn Văn Thịnh	100%
2012968	Nguyễn Phúc Đăng –	100%

* Note: tất cả thành viên đều làm hết exercise và commit lại trên Github: https://github.com/vynguyenkn0812/HCMUT_EmbeddedSystemsLAB

Mục lục

1 Github

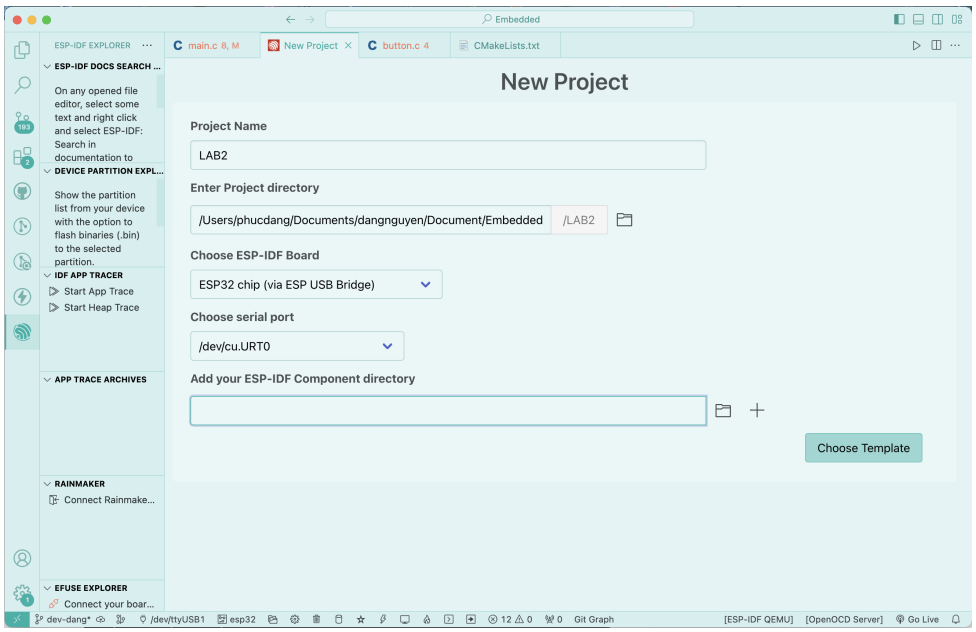
Link: https://github.com/vynguyenkn0812/HCMUT_EmbeddedSystemsLAB

2 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.

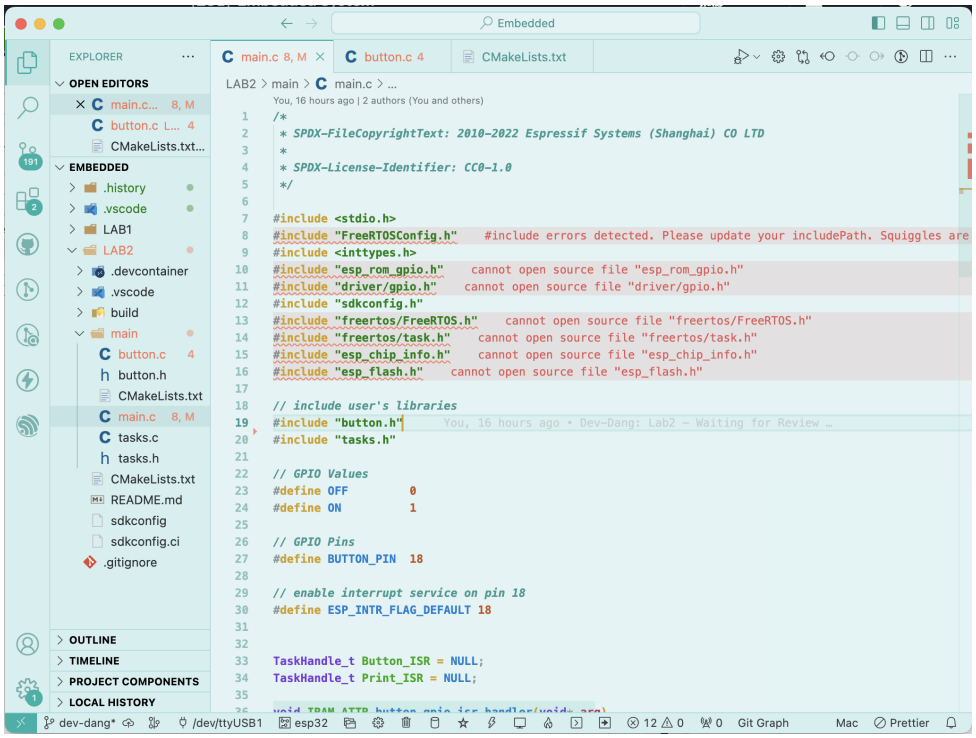
3 Các bước thực hiện

- Đầu tiên tạo một project mới bằng cách vào vscode nhập tổ hợp "Ctrol+Shift+p" (hình mẫu **3.1**).



Hình **3.1**: Create project

- Sau khi đã chọn xong, project đã được tạo như hình **5.4**.



Hình **3.2**: Project template

- Kế tiếp ta cần xác định GPIO Pin cần để gán input:

```
// GPIO Pins
#define BUTTON_PIN 18
```

- Trong hàm `app_main()`, cần khởi tạo GPIO Pin cho chân 18, điều chỉnh hướng của chân 18 (input hay output) và 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 như trong hình **??** (nhóm sử dụng phát hiện cạnh xuống 1->0):

```
// Initialize GPIO pins =======
esp_rom_gpio_pad_select_gpio(BUTTON_PIN);
// ============================

// Set GPIO directions ========
gpio_set_direction(BUTTON_PIN, GPIO_MODE_INPUT);
// ============================

//  Enable interrupt on falling (1->0) edge for button pin ===========
gpio_set_intr_type(BUTTON_PIN, GPIO_INTR_NEGEDGE);
// ==================================================================
```

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

```
gpio_install_isr_service(ESP_INTR_FLAG_DEFAULT);
```

- Thêm hàm để handle interrupt cho nút nhấn chân 18 như sau (cụ thể khi có interrupt thì hàm được add vào sẽ được gọi):

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

- Tạo 2 task như đã đề cập ở phần 1 và start Scheduler:

```
// Create 2 tasks
xTaskCreate(
              vPrintStudentID,      // function
              "print Student ID",   // name to easily debug
              2048,                 // stack allocate
              NULL,                 // none parameter
              1,                    // equal priority
              &Print_ISR            // none taskHandle
          );

xTaskCreate(
              vPrintEsp32,          // function
              "print ESP32",        // name to easily debug
              2048,                 // stack allocate
              NULL,                 // none parameter
              1,                    // equal priority
              &Button_ISR           // none taskHandle
          );

// Start Scheduler
vTaskStartScheduler(void);
```

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

- `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.

```
void vPrintStudentID(void* pvParameters){
    while (1)
    {
        TickType_t xWakeUpTime = xTaskGetTickCount();
        printf("Student ID: 2012968\r\n");
        xTaskDelayUntil(&xWakeUpTime, 1000/portTICK_PERIOD_MS);
    }
}
```

- `vPrintEsp32`: Như đã đề cập ở trên, task này sẽ suspend cho đến khi nó được resume bởi interrupt.

```
void vPrintEsp32(void* pvParameters){
    while (1)
    {
        vTaskSuspend(NULL);
        printf("ESP32\n");
    }
}
```

Sau đây là hàm `button_gpio_isr_handler()`, dùng để resume task `vPrintEsp32` sau khi đã được debounce button:

```
void IRAM_ATTR button_gpio_isr_handler(void* arg)
{
    if( is_button_pressed() ){
        xTaskResumeFromISR(vPrintEsp32);
    }
}
```

Xử lý debounce button:

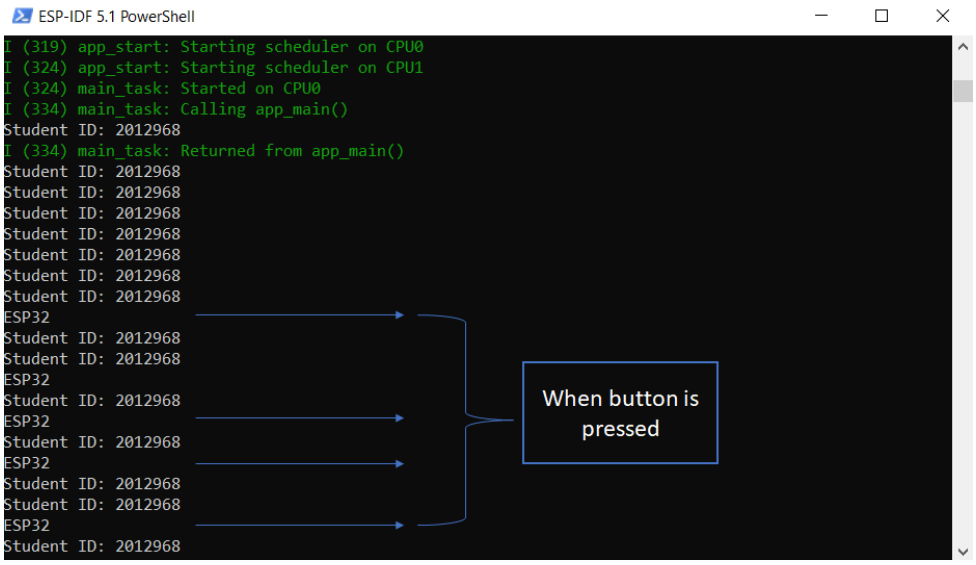
- Khi nút nhấn được nhấn: `current_interrupt_time` sẽ được ghi lại và lấy trừ cho interrupt time trước đó, nếu lớn hơn khoảng `tick_difference` = (`DEBOUNCE_TIME` * `configTICK_RATE_HZ`)/1000 thì trả về 1 còn không thì nó bị bounce và trả về 0.

```
int is_button_pressed(){
    static TickType_t last_interrupt_time = 0;
    TickType_t current_interrupt_time = xTaskGetTickCount();

    // If interrupts come faster than the tick_difference, assume it's a bounce and ignore
    if (current_interrupt_time - last_interrupt_time
    < (DEBOUNCE_TIME * configTICK_RATE_HZ) / 1000)
    {
        return 0;
    }

    last_interrupt_time = current_interrupt_time;
    return 1;
}
```

4 Kết quả



Hình **4.3**: Outcomes

5 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 espressidf.

```
void esp_startup_start_app(void)
{
#if CONFIG_ESP_INT_WDT ···
#elif CONFIG_ESP32_ECO3_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 **5.4**: vTaskStartScheduler() is already called in esp_startup_start_app()