**Phần 1: Lý thuyết**

**1 . Liệt kê bảng vectơ ngắt trên Arduino Uno với các mức ưu tiên tương ứng?**

∙  **INT0**: Chân ngắt ngoài (external interrupt) 0 - Thường sử dụng cho các tín hiệu từ các cảm biến hoặc nút nhấn.

∙  **INT1**: Chân ngắt ngoài 1 - Tương tự như INT0, nhưng sử dụng chân khác.

∙  **TIMER2\_COMPA**: Ngắt từ Timer 2, so sánh kênh A - Thường được sử dụng cho các ứng dụng thời gian như đo thời gian hoặc PWM.

∙  **TIMER2\_COMPB**: Ngắt từ Timer 2, so sánh kênh B - Tương tự như TIMER2\_COMPA nhưng cho kênh B của Timer 2.

∙  **TIMER1\_CAPT**: Ngắt từ Timer 1, sự kiện capture - Được sử dụng cho các ứng dụng liên quan đến đo lường thời gian.

∙  **TIMER1\_COMPA**: Ngắt từ Timer 1, so sánh kênh A - Tương tự như TIMER2\_COMPA nhưng sử dụng Timer 1.

∙  **TIMER1\_COMPB**: Ngắt từ Timer 1, so sánh kênh B - Tương tự như TIMER1\_COMPA nhưng cho kênh B của Timer 1.

∙  **TIMER0\_COMPA**: Ngắt từ Timer 0, so sánh kênh A - Được sử dụng cho các ứng dụng thời gian như đo thời gian hoặc PWM.

∙  **TIMER0\_OVF**: Ngắt từ Timer 0, tràn (overflow) - Khi giá trị của Timer 0 vượt qua giới hạn.

∙  **SPI\_STC**: Ngắt từ SPI (Serial Peripheral Interface) - Khi truyền nhận dữ liệu qua giao tiếp SPI.

∙  **USART\_RX**: Ngắt từ USART (Universal Synchronous Asynchronous Receiver Transmitter), chế độ nhận - Được sử dụng khi dữ liệu được nhận từ cổng nối tiếp.

∙  **USART\_UDRE**: Ngắt từ USART, trống dữ liệu để gửi - Khi dữ liệu có sẵn để gửi qua cổng nối tiếp.

∙  **USART\_TX**: Ngắt từ USART, chế độ gửi - Được sử dụng khi dữ liệu được gửi qua cổng nối tiếp.

∙  **ADC**: Ngắt từ ADC (Analog to Digital Converter) - Khi chuyển đổi analog sang digital hoàn thành.

∙  **EE\_RDY**: Ngắt từ EEPROM (Electrically Erasable Programmable Read-Only Memory) - Khi thao tác đọc/ghi vào EEPROM đã hoàn thành.

∙  **ANALOG\_COMP**: Ngắt từ so sánh analog - Khi sự kiện so sánh analog xảy ra.

**2. Mô tả 5 loại kích hoạt ngắt của Arduino và hàm khởi tạo  
ngắt trên Arduino**

**Ngắt ngoại vi (External Interrupts):** Được kích hoạt bởi sự thay đổi trạng thái của chân ngoại vi, chẳng hạn như một tín hiệu điện từ một cảm biến hoặc một nút nhấn. Arduino Uno có hai ngắt ngoại vi, thường được ký hiệu là INT0 và INT1.

**Ngắt thời gian (Timer Interrupts):** Được kích hoạt bởi các sự kiện thời gian từ các bộ đếm thời gian trong vi điều khiển. Các bộ đếm thời gian này có thể được cấu hình để tạo ra ngắt sau một khoảng thời gian nhất định hoặc sau mỗi chu kỳ PWM. Arduino Uno có ba bộ đếm thời gian, Timer 0, Timer 1 và Timer 2.

**Ngắt truyền thông (Serial Interrupts):** Được kích hoạt khi có dữ liệu mới được nhận hoặc sẵn sàng để gửi qua các giao thức truyền thông như UART (USART trên Arduino). Điều này thường được sử dụng trong việc truyền nhận dữ liệu từ và đến máy tính hoặc các thiết bị khác.

**Ngắt ADC (Analog-to-Digital Converter Interrupts**): Được kích hoạt khi quá trình chuyển đổi tín hiệu analog thành digital hoàn tất. Điều này thường được sử dụng khi cần đọc các giá trị analog từ các cảm biến hoặc thiết bị khác.

**Ngắt EEPROM (Electrically Erasable Programmable Read-Only Memory Interrupts)**: Được kích hoạt khi các hoạt động đọc hoặc ghi vào EEPROM hoàn tất. EEPROM là một loại bộ nhớ không xóa được, thường được sử dụng để lưu trữ dữ liệu không thay đổi trong thiết bị.

Để khởi tạo một ngắt trên Arduino, bạn có thể sử dụng các hàm thư viện có sẵn như attachInterrupt() và detachInterrupt(). Cú pháp chung của các hàm này như sau:

* **attachInterrupt()**: Được sử dụng để kích hoạt một ngắt.

attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);

* Trong đó:
* pin: Chân kích hoạt ngắt.
* ISR: Hàm được gọi khi ngắt xảy ra (Interrupt Service Routine).
* mode: Chế độ kích hoạt ngắt, có thể là LOW, CHANGE, RISING, hoặc FALLING.
* **detachInterrupt()**: Được sử dụng để ngừng kích hoạt một ngắt.

detachInterrupt(digitalPinToInterrupt(pin));

**3. Nêu các vấn đề xảy ra khi các ISR sử dụng các hàm API thông thường của FreeRTOS mà không sử dụng các API an toàn ngắt (Interrupt-Safe API)**

Khi sử dụng hàm API thông thường của FreeRTOS trong các ISR mà không sử dụng các API an toàn ngắt (Interrupt-Safe API), có một số vấn đề tiềm ẩn có thể xảy ra:

1. **Độ tin cậy của hệ thống giảm:** Các hàm API thông thường của FreeRTOS không được thiết kế để hoạt động trong môi trường ngắt, vì vậy sử dụng chúng trong ISR có thể gây ra hành vi không mong muốn hoặc dẫn đến sự cố hệ thống.
2. **Nguy cơ deadlock hoặc livelock:** Sử dụng các hàm API không an toàn trong ISR có thể gây ra deadlock hoặc livelock trong hệ thống. Điều này có thể xảy ra khi ISR cố gắng thực hiện các hoạt động có thể gây chặn (block) trong khi hệ thống đang ở trong một trạng thái tương tự.
3. **Mất dữ liệu hoặc xung đột dữ liệu:** Khi các hàm API không an toàn được sử dụng trong ISR, có nguy cơ mất dữ liệu hoặc xung đột dữ liệu giữa ISR và các tác vụ khác trong hệ thống.
4. **Hiệu suất kém:** Sử dụng các hàm API không an toàn trong ISR có thể làm giảm hiệu suất của hệ thống, vì các hoạt động trong ISR có thể làm tăng thời gian xử lý ngắt và ảnh hưởng đến các tác vụ khác trong hệ thống.
5. **Khả năng mở rộng và bảo trì kém:** Việc sử dụng các hàm API không an toàn trong ISR có thể làm giảm khả năng mở rộng và bảo trì của mã nguồn, vì các lỗi hoặc hành vi không mong muốn có thể khó phát hiện và sửa chữa.

Vì vậy, để đảm bảo tính ổn định và tin cậy của hệ thống, nên luôn sử dụng các API an toàn ngắt (Interrupt-Safe API) của FreeRTOS trong các ISR, và tránh sử dụng các hàm API thông thường.

**4. Khái niệm ngắt lồng (Nesting Interrupt) và giải thích 2 hằng số liên quan đến độ ưu tiên của ngắt khi thiết lập ngắt lồng**

Ngắt lồng (Nesting Interrupt) là một khái niệm trong hệ thống nhúng, xảy ra khi một ngắt mới xảy ra trong khi một ngắt khác đang được xử lý. Điều này có nghĩa là, trong một hệ thống với nhiều ngắt có ưu tiên khác nhau, một ngắt mới có thể xảy ra trong khi một ngắt khác đang được xử lý.

Trong hệ thống có khả năng ngắt lồng, có hai hằng số liên quan đến độ ưu tiên của ngắt khi thiết lập ngắt lồng:

1. **Priority Level (Mức Ưu Tiên)**: Đây là mức ưu tiên của mỗi ngắt, thường được xác định bằng một số nguyên. Các ngắt có mức ưu tiên cao hơn được ưu tiên hơn và có thể "lồng" vào các ngắt có mức ưu tiên thấp hơn. Trong hầu hết các hệ thống, mức ưu tiên càng thấp, ưu tiên càng cao. Ví dụ, ngắt với mức ưu tiên 0 thường cao hơn ngắt với mức ưu tiên 1.
2. **Interrupt Priority Level (Mức Ưu Tiên của Ngắt)**: Đây là mức ưu tiên được đặt cho ngắt trong hệ thống. Trong một số kiến trúc, mức ưu tiên của ngắt có thể được cấu hình để ứng với một mức ưu tiên cụ thể. Khi một ngắt xảy ra, hệ thống sẽ xác định xem ngắt mới có được xử lý ngay lập tức hay không dựa trên mức ưu tiên của nó so với mức ưu tiên của ngắt hiện tại đang được xử lý.

Trong một hệ thống có khả năng ngắt lồng, việc quản lý độ ưu tiên của các ngắt là rất quan trọng để đảm bảo tính ổn định và đáng tin cậy của hệ thống. Điều này đặc biệt quan trọng trong các ứng dụng yêu cầu thời gian thực hoặc độ tin cậy cao.

Phần 2: Thực hành

Bài 7.1  
#include <Arduino\_FreeRTOS.h> /

// Định nghĩa số chân cho LED và chân ngắt

const byte ledPin = 13; // Số chân cho LED (LED tích hợp trên nhiều board Arduino)

const byte interruptPin = 2; // Số chân cho ngắt (kết nối với nút nhấn)

// Định nghĩa biến volatile để giữ trạng thái của LED

volatile byte state = LOW; // Trạng thái ban đầu của LED là LOW (tắt)

void setup() {

   // Thiết lập chế độ chân cho chân LED là OUTPUT

   pinMode(ledPin, OUTPUT);

   // Thiết lập chế độ chân cho chân ngắt là INPUT\_PULLUP

   pinMode(interruptPin, INPUT\_PULLUP);

   // Gắn hàm xử lý ngắt (glow) cho sự thay đổi trạng thái của chân ngắt

   attachInterrupt(digitalPinToInterrupt(interruptPin), glow, CHANGE);

}

void loop() {

   // Ghi giá trị của biến state vào chân LED

   digitalWrite(ledPin, state);

}

// Hàm xử lý ngắt, thay đổi trạng thái của LED

void glow() {

   state = !state; // Đảo ngược trạng thái của LED

}

Bài 7.2   
#include <Arduino\_FreeRTOS.h> // Bao gồm thư viện FreeRTOS cho Arduino

#include <semphr.h> // Bao gồm thư viện semaphore của FreeRTOS

SemaphoreHandle\_t interruptSemaphore; // Khai báo một semaphore để sử dụng trong ngắt

void setup()

{

  Serial.begin(9600);  pinMode(LED\_BUILTIN, OUTPUT); // Thiết lập chân LED tích hợp (LED\_BUILTIN) là OUTPUT

  // Tạo task cho việc bật LED

  xTaskCreate(TaskLedon, // Hàm thực thi task

              "Ledon", // Tên task

              128, // Kích thước stack

              NULL, // Tham số truyền vào task

              0 , // Mức ưu tiên của task

              NULL ); // Con trỏ để lưu trữ tay cầm của task

  // Tạo task cho việc tắt LED

  xTaskCreate(TaskLedoff, // Hàm thực thi task

              "Ledoff", // Tên task

              128, // Kích thước stack

              NULL, // Tham số truyền vào task

              0, // Mức ưu tiên của task

              NULL ); // Con trỏ để lưu trữ tay cầm của task

  interruptSemaphore = xSemaphoreCreateBinary(); // Tạo một semaphore kiểu binary

  if (interruptSemaphore != NULL) // Kiểm tra xem semaphore có được tạo thành công không

  {

    attachInterrupt(digitalPinToInterrupt(2), interruptHandler, HIGH); // Gắn hàm xử lý ngắt cho chân số 2

  }

}

void loop() {

 }

void interruptHandler()

{

  Serial.println("Semaphore is given");

  BaseType\_t xHigherPriorityTaskWoken = pdFALSE; // Biến để kiểm tra nếu một task có ưu tiên cao hơn cần được đánh thức

  xSemaphoreGiveFromISR(interruptSemaphore, &xHigherPriorityTaskWoken); // Đưa semaphore từ trong ngắt

}

void TaskLedon(void \*pvParameters)

{

  (void) pvParameters; // Bỏ qua tham số không sử dụng

  for (;;) // Vòng lặp vô hạn cho task

  {

    if (xSemaphoreTake(interruptSemaphore, portMAX\_DELAY) == pdPASS) // Chờ nhận semaphore với thời gian chờ vô hạn

    {

      Serial.println("TaskLedon Received Semaphore");

      digitalWrite(LED\_BUILTIN, HIGH); // Bật LED

    }

  }

}

void TaskLedoff(void \*pvParameters)

{

  (void) pvParameters; // Bỏ qua tham số không sử dụng

  for (;;) // Vòng lặp vô hạn cho task

  {

    if (xSemaphoreTake(interruptSemaphore, portMAX\_DELAY) == pdPASS) // Chờ nhận semaphore với thời gian chờ vô hạn

    {

      Serial.println("TaskLedoff Received Semaphore");

      digitalWrite(LED\_BUILTIN, LOW); // Tắt LED

    }

  }

}