|  |  |  |  |
| --- | --- | --- | --- |
| Họ và tên | Phạm Quốc Tiến | Nguyễn Đức Trung | Bùi Thái Toàn |
| MSSV | 22521472 | 22521564 | 22521485 |
| Họ và tên | Trịnh Thành Trung | Trần Quốc Trinh | Nhật Tin |
| MSSV | 22521570 | 22521542 | 22521479 |
| Họ và tên | Nguyễn Hữu Bảo Trọng |  |  |
| MSSV | 22521545 |  |  |

LAB 4: RTOS trên STM43F429

# CHUẨN BỊ

|  |  |
| --- | --- |
| Phần cứng  * KIT STM32F4 Discovery for STM32F429 MCU.  Phần mềm  * STM32CubeIDE: sử dụng để lập trình, build, nạp và debug code.  Kiến thức  * Biết cách tạo project và cấu hình project sử dụng STM32CubeIDE * Có kiến thức vững về Hệ điều hành, đặc biệt là vấn đề lập lịch và đồng bộ các tiến trình | *Hình 1.1. KIT STM32F4 Discovery* |

# HƯỚNG DẪN

## Real-time Operating System (RTOS) là gì?

Để có thể làm việc với RTOS, trước tiên chúng ta cần hiểu đây là gì và sự khác biệt giữa một hệ điều hành thông thường và hệ điều hành thời gian thực là như thế nào?

Hệ điều hành thời gian thực - RTOS, là một phần mềm giúp quản lý phần cứng và các phần mềm khác bên trong một hệ thống máy tính. Chúng ta đã khá quen thuộc với một số hệ điều hành đa dụng trên thế giới như Windows, Linux, OSX hay iOS và Android. Một hệ điều hành đa dụng (general purpose OS) sẽ thường được thiết kế tập trung vào trải nghiệm của người dùng.

Nhắc lại kiến thức về hệ đa dụng, ví dụ, khi chúng ta phát triển một phần mềm cho một hệ điều hành trên điện thoại, như Android hoặc iOS chẳng hạn. Người dùng có thể sẽ muốn stream một bộ phim, chúng ta có thể chia quá trình stream bộ phim này thành 2 công việc: (1) là tải từng đoạn video về từ Internet và (2) chiếu từng đoạn video đó cho người dùng xem. 2 công việc này là một phần của chương trình lớn và có thể được hiện thực đồng thời với nhau thông qua các tiểu trình (thread). Trong trường hợp như máy tính của chúng ta chỉ có 1 core CPU, như vậy, phần mềm stream phim sẽ phải luân phiên thực hiện liên tục công việc (1) và công việc (2), do tốc độ xử lý của CPU là rất cao vì vậy sẽ đem đến cho người dùng cảm giác như họ đang vừa load và vừa xem phim cùng lúc.

Shape

Description automatically generated with medium confidence

*Hình 2.1. Hai công việc được hệ điều hành thực hiện luân phiên liên tục với nhau.*

Hình 2.1 mô tả cách hệ điều hành chuyển đổi luân phiên liên tục giữa 2 công việc, hay còn có tên gọi khác là tiến trình hay tiểu trình. Lưu ý rằng, trong quá trình này còn xuất hiện một công việc thứ 3 chính là công việc của hệ điều hành thực hiện "chuyển ngữ cảnh" (switch context). Chuyển ngữ cảnh là công việc lưu lại PCB - chứa thông tin trạng thái - của tiến trình cũ, và nạp PCB của tiến trình mới vào bộ nhớ chính cũng như các thanh ghi.

Trong quá trình thực hiện các công việc, có thể sẽ xảy ra độ trễ, một gói tin có thể mất trong quá trình tải video, một khung hình có thể bị rớt trong quá trình chiếu video để đảm bảo video có thể bắt kịp tốc độ trình chiếu. Trọng tâm của các hệ điều hành đa dụng sẽ tập trung vào trải nghiệm của người dùng, những vấn đề nhỏ nhặt trên trong quá trình thực thi tiến trình thường sẽ không gây quá nhiều sự chú ý đối với người dùng, hay nói cách khác, người dùng có thể dễ dàng chấp nhận, hoặc không quan tâm đến những lỗi nho nhỏ đó.

Tuy nhiên, trọng tâm của hầu hết các chương trình trên các vi điều khiển lại khác. Việc đáp ứng các hạn định thời một cách nghiêm ngặt là một vấn đề hết sức quan trọng. Hãy nghĩ đến một bộ điều khiển động cơ hay một hệ thống phanh tự động trong một chiếc xe điện. Trong trường hợp này, nếu như việc xử lý định thời bị trễ dù chỉ vài mili giây thì tính mạng con người cũng có thể gặp nguy hiểm. Do đó, RTOS chính là sự lựa chọn tốt nhất để quản lý các công việc được thực thi đồng thời với nhau và phải đảm bảo tính thời gian thực.

Trong ví dụ trên, một bộ điều khiển điện tử (electronic control unit - ECU) được gắn trên xe có thể chịu trách nhiệm việc kiểm soát và hỗ trợ phanh xe. Quan sát Hình 2.2, công việc (2) thực hiện việc giám sát các input được ghi nhận từ tài xế, từ đó giúp đạp và bật đèn hậu. Tuy nhiên, giả sử rằng ECU của chúng ta nhận được thông báo từ các cảm biến phát hiện một vụ va chạm sắp xảy ra, khi đó, công việc (1) sẽ trưng dụng CPU từ công việc (2) và điều khiển thắng xe. Tất nhiên, tất cả điều này giả định rằng đây là một chiếc xe được trang bị hỗ trợ phanh tự động.

Chart, box and whisker chart

Description automatically generated

*Hình 2.2. Hai công việc được thực thi trong RTOS*

Lưu ý rằng đây là một ví dụ hết sức đơn giản. Chúng ta cũng có thể sử dụng các ngắt phần cứng để trưng dụng chương trình đang thực thi. Dù vậy, một RTOS có thể cho phép chúng ta tạo ra các tác vụ phần mềm thay vì phụ thuộc vào các ngắn phần cứng và gán độ ưu tiến cho các tác vụ này. Ngoài ra, hầu hết các RTOS đều có thể hoạt động như một lớp trừu tượng (abstraction layer), cho phép chúng ta viết code và có thể dễ dàng triển khai trên các vi điều khiển khác.

|  |
| --- |
| Tìm hiểu và trình bày những khác biệt giữa hệ điều hành đa dụng và hệ điều hành thời gian thực? |

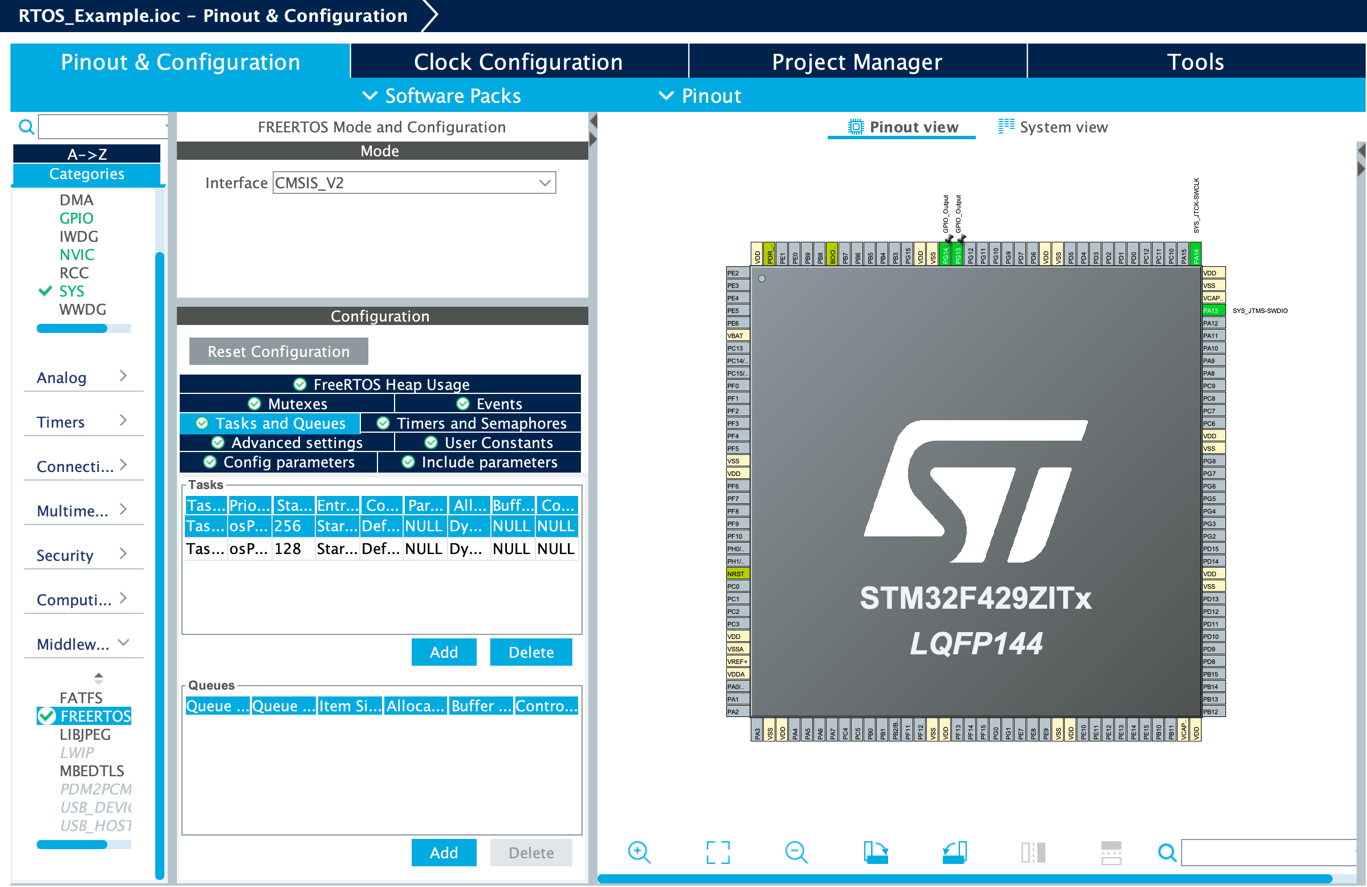
|  |  |  |
| --- | --- | --- |
| **Đặc điểm** | **GPOS** | **RTOS** |
| **Độ chính xác thời gian** | **Không yêu cầu** | **Rất quan trọng** |
| **Lập lịch** | **Công bằng hoặc hiệu suất** | **Ưu tiên nghiêm ngặt** |
| **Thời gian đáp ứng** | **Không đảm bảo** | **Đảm bảo đúng thời gian** |
| **Ngành ứng dụng** | **Máy tính, smartphone** | **Hệ thống nhúng, IoT, công nghiệp** |

## Lập trình với RTOS trên STM32F429

Trong phần này, chúng ta sẽ thực hiện lại việc lập trình cho LED 3 và LED 4 cùng nhấp nháy với chu kỳ khác nhau dựa trên việc tạo ra tác vụ thực thi đồng thời trong RTOS trên STM32F429.

***Cấu hình Project:***

Trước tiên, chúng ta cần tạo một project trên STM32CubeIDE và thực hiện cấu hình như sau:



*Hình 2.3. Kích hoạt RTOS trên KIT*

* Trong mục Middlewares, chọn FREERTOS, trong mục Interface chọn CMSIS\_V2
* Trong phần Configurations cho FREERTOS, chọn tab Tasks and Queues:
  + Double-click vào task mặc định và thay đổi thông tin cho Task như bên dưới. Trong này, ta đặc biệt chú ý đến trường Priority sẽ cài đặt độ ưu tiên cho task, trường Stack Size cài đặt kích thước vùng Stack (dùng để chứa các biến cục bộ), trường Entry Function xác định tên hàm chứa code cụ thể cho task.

Graphical user interface, application

Description automatically generated

*Hình 2.4. Cài đặt Task01*

* + Tiếp theo, cũng trong tab Tasks and Queues, bấm nút Add để tạo thêm một task mới có cấu hình như bên dưới.

Graphical user interface, application

Description automatically generated

*Hình 2.5. Cài đặt Task02*

Tiếp theo, chúng ta cần phải quan tâm đến SysTick. SysTick là một timer đặc biệt trong hầu hết các các bộ xử lý ARM và thường được dùng cho các mục đích của hệ điều hành. Mặc định, SysTick trong một chip STM32 sẽ kích hoạt một ngắt mỗi 1ms. Nếu chúng ta đang sử dụng framework STM32 HAL, mặc định, SysTick sẽ được dùng cho các hàm như HAL\_Delay() (đã đề cập ở LAB 01) và HAL\_GetTick(). Do đó, STM32 HAL framework cài đặt độ ưu tiên rất cao cho SysTick. Tuy nhiên, FreeRTOS cần dùng SysTick cho bộ định thời của nó, và nó yêu cầu SysTick ở một mức độ ưu tiên thấp hơn nhiều. Chúng ta có vài cách để xử lý vấn đề xung đột này, nhưng dễ nhất chính là gán một timer khác, chưa được sử dụng để làm timebase source cho HAL. Timer 6 và 7 trong hầu hết các vi điều khiển STM32 thường rất là cơ bản, do đó nó có thể được dễ dàng sử dụng để làm timebase source cho HAL. Để thực hiện thao tác này, trong mục System Core 🡪 SYS, trong mục Timebase Source chọn TIM6.

Graphical user interface, text, application

Description automatically generated

*Hình 2.6. Thay đổi Timebase Source cho HAL*

Sau cùng của bước cấu hình project, chọn File 🡪 Save để lưu và sinh code.

***Viết các task/tiểu trình thực thi đồng thời:***

Sau khi code được tự động sinh ra, quan sát file main.c, ta thấy trong phần Includes, thư viện "cmsis\_os.h" được thêm vào để làm một lớp trừu tượng giúp chúng ta giao tiếp và làm việc với nhân RTOS được dễ dàng.

|  |
| --- |
| /\* Includes ------------------------------------------------------------------\*/  **#include** "main.h"  **#include** "cmsis\_os.h" |

Tiếp tục quan sát, trong phần định nghĩa hai task, ta có:

|  |
| --- |
| /\* Definitions for Task01 \*/  osThreadId\_t Task01Handle;  **const** osThreadAttr\_t Task01\_attributes = {  .name = "Task01",  .stack\_size = 256 \* 4,  .priority = (osPriority\_t) osPriorityNormal,  };  /\* Definitions for Task02 \*/  osThreadId\_t Task02Handle;  **const** osThreadAttr\_t Task02\_attributes = {  .name = "Task02",  .stack\_size = 128 \* 4,  .priority = (osPriority\_t) osPriorityNormal,  }; |

Dễ thấy, các thông số mà ta tạo ra cho tác vụ Task01 và Task02 trong giao diện cấu hình trước đó được định nghĩa tại đoạn code này. Tiếp tục, ta sẽ thấy các hàm sau lần lượt được gọi:

* **osKernelInitialize**(): Khởi tạo nhân OS
* Task01Handle = **osThreadNew**(StartTask01, NULL, &Task01\_attributes): Xác định công việc cho Task01
* Task02Handle = **osThreadNew**(StartTask02, NULL, &Task02\_attributes): Xác định công việc cho Task02
* **osKernelStart**(): Khởi chạy nhân OS. Lưu ý, khi hàm này được gọi, ngay lập tức, Task01 và Task02 ở trên sẽ được chạy. Công việc trong Task01 và Task02 thường là vòng lặp vô tận và được thực thi đồng thời với nhau. Do đó, ta sẽ không để bất kỳ đoạn code nào khác sau osKernelStart() trong hàm main(), bởi một cách lý tưởng, chương trình sẽ không còn return về main() sau khi osKernelStart() chạy mà chỉ thực thi Task01 và Task02 theo đúng thiết kế của chúng ta.

Lúc này, chương trình của chúng ta sẽ có 2 tác vụ được thi đồng thời là Task01 và Task02 với các công việc được xác định trong hàm StartTask01 và StartTask02. Các công việc này là liên tục và không bao giờ dừng. Bên cạnh đó, sẽ có một tác vụ định thời được chạy nền để thực hiện công việc chuyển ngữ cảnh giữa các task trên.

Như đã nói ở trên, mục đích của chúng ta là nhấp nháy LED 3 và LED 4, chúng ta sẽ thực hiện việc này lần lượt trong hàm StartTask01 và StartTask02:

|  |
| --- |
| **void** **StartTask01**(**void** \*argument)  {  /\* USER CODE BEGIN 5 \*/  /\* Infinite loop \*/  **for**(;;)  {  **HAL\_GPIO\_TogglePin**(GPIOG, GPIO\_PIN\_13);  **osDelay**(1000);  }  /\* USER CODE END 5 \*/  } |
| **void** **StartTask02**(**void** \*argument)  {  /\* USER CODE BEGIN StartTask02 \*/  /\* Infinite loop \*/  **for**(;;)  {  **HAL\_GPIO\_TogglePin**(GPIOG, GPIO\_PIN\_14);  **osDelay**(2000);  }  /\* USER CODE END StartTask02 \*/  } |

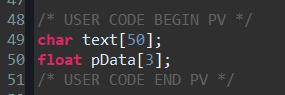
Lưu ý, ở đây thay vì sử dụng hàm HAL\_Delay() như trong LAB 01, chúng ta sẽ sử dụng hàm osDelay(). Lý do là vì hàm HAL\_Delay() có độ ưu tiên rất cao và có thể sẽ can thiệp vào tiến trình, ngăn cản việc chuyển ngữ cảnh của hệ điều hành. Đồng thời nó cũng sẽ ngăn cản bộ định thời nghỉ ngơi, làm cho hệ thống không thể tiết kiệm năng lượng. Do đó, chúng ta sử dụng hàm osDelay() để báo với bộ định thời rằng nó có thể chuyển sang Task khác trong khi chúng ta chờ đợi.

# BÀI TẬP

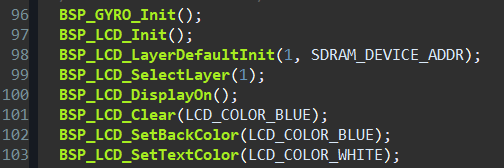
**Bài tập 1:** Không sử dụng FreeRTOS, hãy thiết kế một hệ thống đọc dữ liệu từ Gyroscope, sau đó gửi cho máy tính thông qua Virtual Com Port, đồng thời hiện hướng của Gyroscope lên màn hình LCD như mô tả trong Hình 3.1.

**Nhóm sẽ setup như những lab trước đã thực hiện**

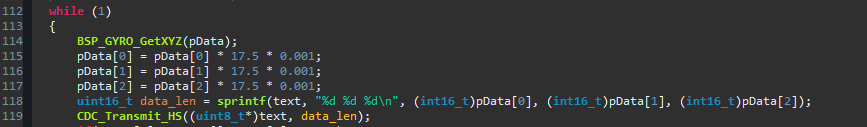
**Khai báo biến toàn cục**

****

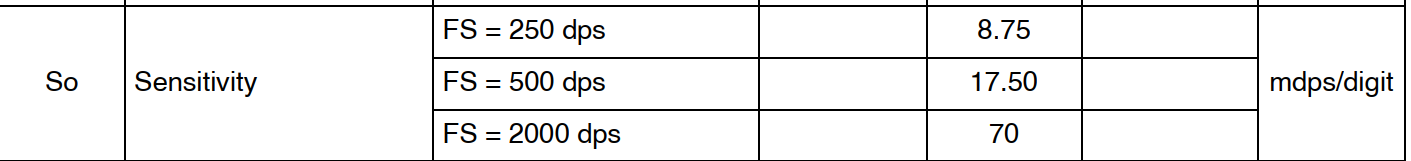
**Setup LCD, Gyroscope**

****

**Trong vòng while nhóm sẽ thực hiện đọc giá trị mà gyroscope trả về, chuyển đổi dữ liệu và sau đó sẽ gửi ba tham số XYZ đã chuyển đổi qua USB**

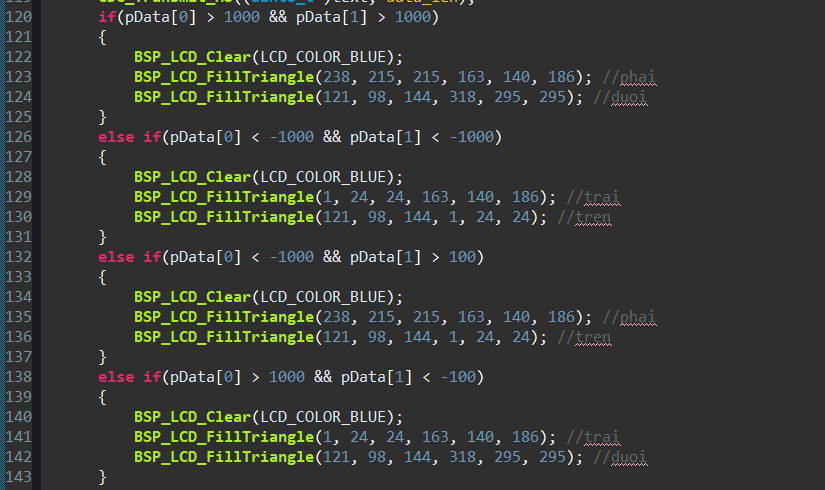
****

**Lý do mà nhóm thực hiện nhân 17.5 \* 0.0001 cho các giá trị XYZ là để đưa các giá trị về tốc độ góc. Theo datasheet thì mỗi đơn vị sẽ được quy đổi thành mdps (Millidegrees per Second) theo giá trị full scale (fs) trong sensitivity tương ứng và khi đọc trong hàm BSP\_GYRO\_Init() thì nhóm thấy thư viện sử dụng FS = 500 dps**

****

****

**Sau đó nhóm sẽ thực hiện so sánh các giá XYZ và xuất ra các trường hợp mũi tên tương ứng. Do gyroscope trên kit khá nhiễu nên giá trị trong lệnh điều kiện nhóm chỉ lấy giá trị tương đối**

****

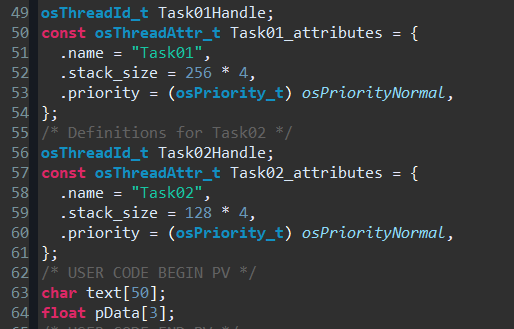
**Sau đó nhóm sẽ thực hiện delay 100 ms**

****

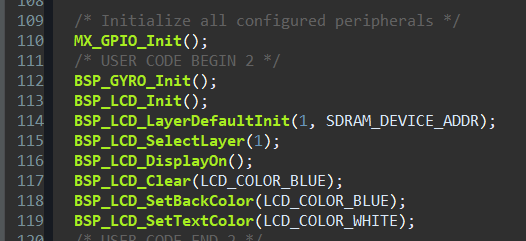
**Bài tập 2:** Thực hiện lại bài tập trên nhưng có sư dụng FreeRTOS.

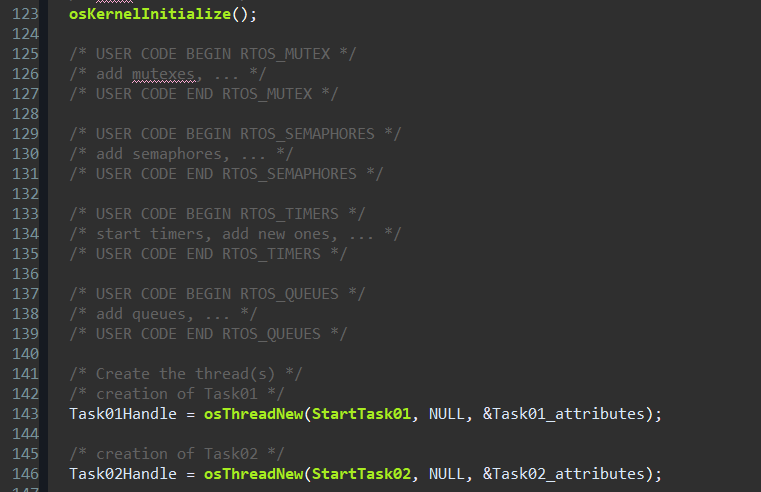
**Với ý tưởng và giải thuật như trên bài tập 1, nhóm sẽ tách quá trình xử lý trong vòng while tại hàm main ra hai task độc lập. Với Task01 sẽ xử lý việc đọc, convert dữ liệu và gửi data qua USB và Task02 sẽ là giải thuật xuất lên LCD mũi tên tương ứng. Do Task01 sẽ xử lý nhiều hơn nên nhóm sẽ cấp bộ nhớ stack nhiều hơn Task02**

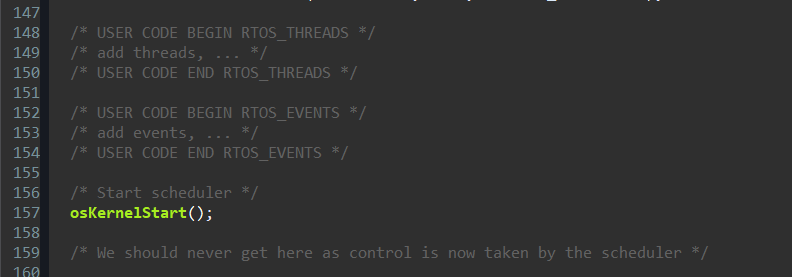
**Khai báo biến và task**

****

**Code trong hàm main**

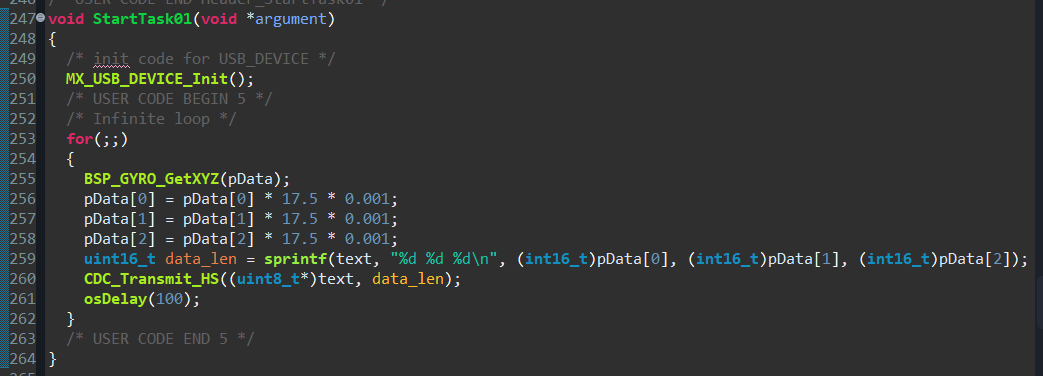
****

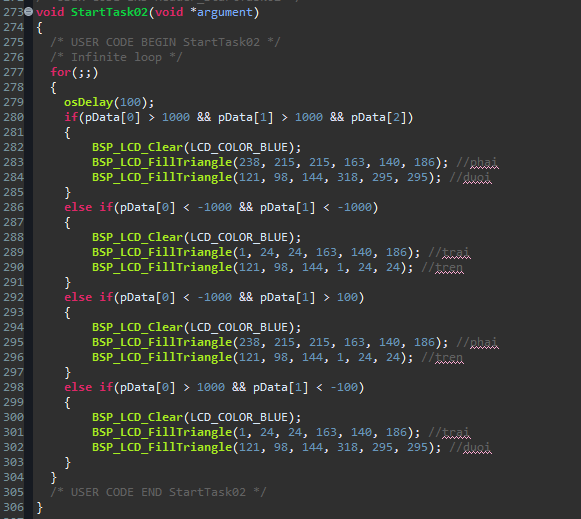
****

****

**Có thể thấy trong code main lại không gọi hàm MX\_USB\_DEVICE\_Init() do IDE đã đưa phần này xuống Task01**

**Ở hai Task nhóm sẽ thực hiện delay 100 ms ở hai chỗ khác nhau, một ở đầu vòng lặp và một là ở cuối vòng lặp với mục đích là để tránh bị tranh chấp ở array pData[]**

****

****

**So sánh bài tập 1 và bài tập 2**

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Bài tập 1** | **Bài tập 2** |
| **Độ phức tạp** | **Đơn giản, dễ hiểu hơn cho người mới** | **Phức tạp hơn, yêu cầu hiểu rõ về FreeRTOS và cách chia task** |
| **Hiệu quả sử dụng CPU** | **Không tận dụng được CPU khi có các công việc không liên quan** | **Tận dụng CPU tốt hơn nhờ osDelay() và hệ thống lập lịch của FreeRTOS** |
| **Khả năng mở rộng** | **Khó mở rộng nếu thêm nhiều tác vụ** | **Dễ mở rộng bằng cách thêm các task mới với độ ưu tiên tương ứng** |
| **Xử lý đồng thời** | **Không thực hiện đồng thời thật sự, chỉ là tuần tự trong vòng lặp** | **Thực hiện đồng thời hai task nhờ lập lịch của FreeRTOS** |
| **Độ ổn định** | **Có thể gặp tranh chấp hoặc lỗi nếu vòng lặp phức tạp hơn** | **Tránh tranh chấp nhờ FreeRTOS quản lý tài nguyên tốt hơn** |

Icon

Description automatically generated

*Hình 3.1. Hiển thị hướng của GyroScope lên LCD*

Tham khảo:

[*https://hocarm.org/rtos-co-ban-phan-1/*](https://hocarm.org/rtos-co-ban-phan-1/)

[*https://hocarm.org/rtos-co-ban-phan-2/*](https://hocarm.org/rtos-co-ban-phan-2/)

[*https://www.st.com/resource/en/datasheet/l3gd20.pdf*](https://www.st.com/resource/en/datasheet/l3gd20.pdf)

[*https://stm32f4-discovery.net/2014/08/library-28-l3gd20-3-axis-gyroscope/*](https://stm32f4-discovery.net/2014/08/library-28-l3gd20-3-axis-gyroscope/)

[*https://itecnotes.com/electrical/converting-raw-gyro-l3gd20h-values-into-angles/*](https://itecnotes.com/electrical/converting-raw-gyro-l3gd20h-values-into-angles/)