08 Huong dan lap trinh voi hệ điều hành Free RTOS trên STM32

Kỹ thuật lập trình (Trường Đại học Công nghiệp Hà Nội)

BÀI 4.2 LẬP TRÌNH ỨNG DỤNG VỚI HỆ ĐIỀU HÀNH FREERTOS

➢ **Mục đích.**

Trang bị cho sinh viên các kiến thức về hệ điều hành trong vi điều khiển ARM STM32. Cách lập trình sử dụng hệ điiều hành FreeRTOS để thiết kế ứng dụng.

➢ **Yêu cầu.**

Sinh viên nắm được:

- Khái niệm, ứng dụng và hoạt động của hệ điều hành FreeRTOS
- Cách cài đặt hệ điều hành FreeRTOS trên STM32CubeMx.

1. Hệ điều hành thời gian thực FreeRTOS.

1.1 Hệ điều hành thời gian thực(RTOS)

Hệ điều hành (Operating System – viết tắt: OS): là một phần mềm dùng để điều hành, quản lý toàn bộ tất cả thành phần (bao gồm cả phần cứng và phần mềm) của thiết bị điện tử.

Hệ điều hành thời gian thực (Real Time Operating System – RTOS) là một hệ điều hành đa nhiệm hướng đến các ứng dụng thời gian thực, thường được nhúng trong các dòng vi điều khiển dùng để điều khiển thiết bị một cách nhanh chóng và đa tác vụ (multi tasking). Cũng có thể hiểu, RTOS là một chương trình có nhiệm vụ lập lịch thực hiện nhiều tác vụ, các tác vụ thực thi đảm bảo tính thời gian thực, quản lý tài nguyên, cung cấp một sự thiết lập phù hợp cho các mã chương trình ứng dụng.

Tính thời gian thực của một hệ thống là hệ thống đó chỉ cần hoàn thành các công việc, các tác vụ trong một khoảng thời gian cho phép (deadline)

RTOS thường được ứng dụng trong các hệ thống đòi hỏi cao đáp ứng về mặt thời gian trong một số lĩnh vực như: viễn thông(router mạng, switch mạng..), thiết bị y tế, thiết bị hàng không, robot, ô tô hay các xe tự hành, các thiết bị chăm sóc sức khỏe IoT...Lấy ví dụ với các hệ điều hành trên máy tính như Windows, Ubuntu.. có thể bị treo phải chờ khởi động lại, tuy nhiên hệ điều hành trên ô tô nếu xảy ra trường hợp bị treo sẽ dẫn đến hậu quả nghiêm trọng, do đó trên ô tô sử dụng hệ điều hành RTOS sẽ xử lý các nhiệm vụ theo thời gian thực, nếu trong quá trình hoạt động có 1 nhiệm vụ nào bị sự cố thì hệ điều hành cô lập nhiệm vụ đó để đảm bảo các nhiệm vụ khác vẫn hoạt động bình thường.

Hiện nay có rất nhiều hệ điều hành thời gian thực phổ biến có thể kể đến như: FreeRTOS, Embedded Linux, VxWorks, Win CE, Lynyos, BSD, Green Hills, QNX và DOS, uClinux..

Lợi ích khi sử dụng RTOS:

- Chia sẻ tài nguyên một cách đơn giản: cung cấp cơ chế để phân chia các yêu cầu về bộ nhớ và ngoại vi của MCU.

- Giúp dễ quản lý và phát triển chương trình do phân chia chương trình thành các tác vụ riêng biệt, các tác vụ này có thể phát triển độc lập.
- Tăng tính linh động và dễ bảo trì thông qua hàm API của RTOS

Các khái niệm cơ bản trong RTOS

Kernel – Nhân: Có nhiệm vụ quản lý và điều phối các Task. Mọi sự kiện (Even) như ngắt, Timer, data truyền tới… đều qua Kernel xử lý để quyết định xem nên làm gì tiếp theo.Thời gian xử lý của Kernel thường rất nhanh nên độ trễ rất thấp.

Task – Tác vụ : Task là một đoạn chương trình thực thi một hoặc nhiều nhiệm vụ gì đó, được Kernel quản lý. Có thể hiểu tác vụ giống như chương trình con của ngắt. Một chương trình thường sẽ có nhiều task khác nhau, Kernel sẽ quản lý việc chuyển đổi giữa các task, nó sẽ lưu lại ngữ cảnh của task sắp bị hủy và khôi phục lại ngữ cảnh của task tiếp theo.

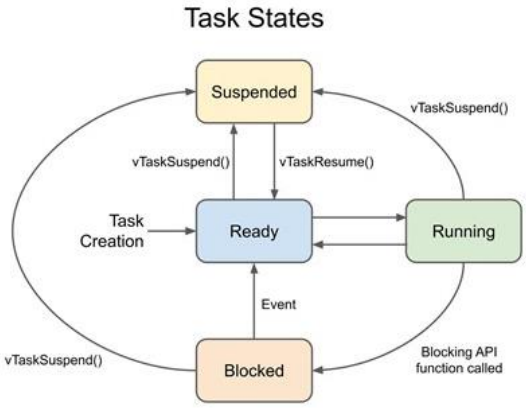
Task States – Trạng thái Task : Một task trong RTOS thường có các trạng thái như sau :

RUNNING: đang thực thi

READY: sẵn sàng để thực hiện

WAITING: chờ sự kiện

INACTIVE: không được kích hoạt



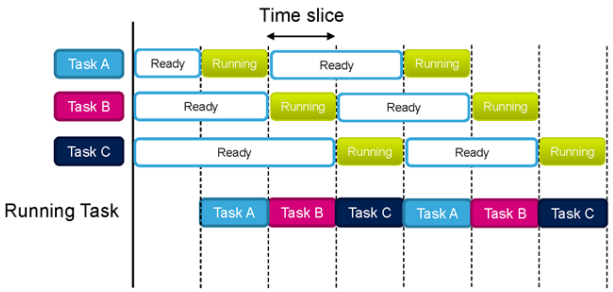
Hình 1: Trạng thái của các task[1]

Scheduler – Lập lịch : là 1 thành phần của kernel quyết định trình tự các task nào được thực thi. Có một số luật cho scheduling như:

Cooperative: giống với lập trình thông thường, mỗi task chỉ có thể thực thi khi task đang chạy dừng lại, nhược điểm của nó là task này có thể dùng hết tất cả tài nguyên của CPU

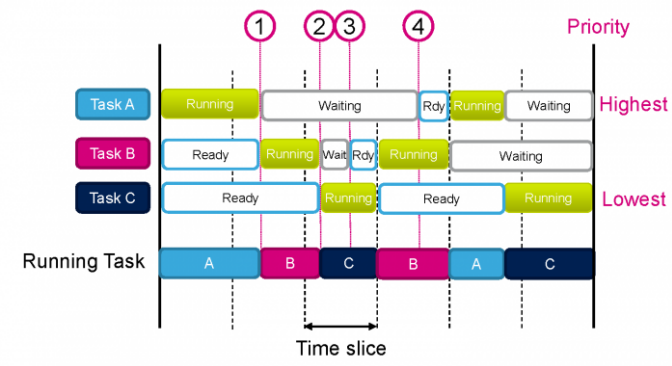
Round-robin: mỗi task được thực hiện trong thời gian định trước (time slice) và không có ưu tiên.

[1] https://www.freertos.org/RTOS-task-states.html



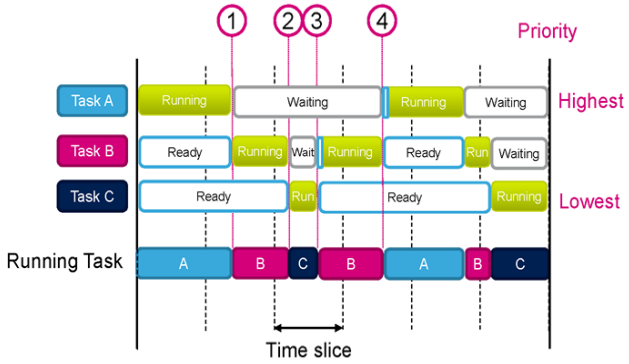
Hình 2: Hoạt động các task theo luật Round-robin

Priority base: Task được phân quyền cao nhất sẽ được thực hiện trước, nếu các task có cùng quyền như nhau thì sẽ giống với round-robin, các task có mức ưu tiên thấp hơn sẽ được thực hiện cho đến cuối time slice



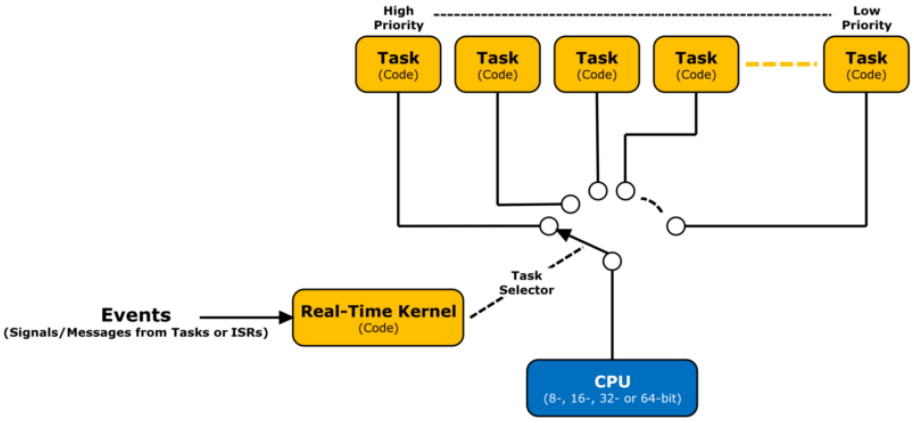
Hình 3: Hoạt động các task theo luật Priority base

Priority-based pre-emptive: Các task có mức ưu tiên cao nhất luôn nhường các task có mức ưu tiên thấp hơn thực thi trước.



Hình 4: Hoạt động các task theo luật Priority-based pre-emptive

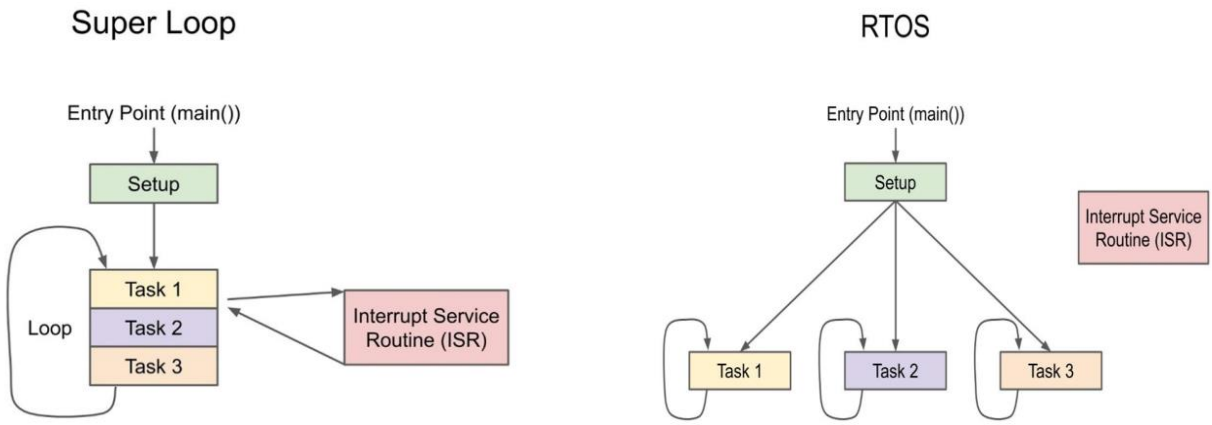
Cơ chế hoạt động của RTOS



Hình 5: Cơ chế hoạt động của RTOS

Chương trình sẽ bao gồm các tác vụ riêng biệt, nhân Kernel sẽ điều phối sự hoạt động của các tác vụ (Task), mỗi task sẽ thực thi một hoặc nhiều công việc, mỗi task sẽ có một mức ưu tiên (prioritize) và thực thi theo chu kì cố định. Nếu có sự tác động như ngắt, tín hiệu hoặc tin nhắn giữa các Task, Kernel sẽ điều phối chuyển tới Task tương ứng với Code đó. Sự chuyển dịch giữa các Task rất linh động, độ trễ thấp mang lại độ tin cậy cao cho chương trình.

So sánh chương trình dùng siêu vòng lặp với chương trình đa tác vụ:



Hình 6: So sánh chương trình dùng siêu vòng lặp với đa tác vụ

Chương trình sử dụng siêu vòng lặp ví dụ các vòng lặp while(1), loop().. các tác vụ thực hiện một cách tuần tự. Tuy nhiên khi có quá nhiều tác vụ cần phải hoàn thành trong 1 thời gian nhất định nhưng vẫn phải "xếp hàng" chờ cho nhiệm vụ trước hoàn thành xong dẫn đến các tác vụ bị trễ, không đảm bảo tính thời gian thực. Trong khi đó RTOS thay vì thực hiện các tác vụ lần lượt thì về cơ bản có thể thực hiện các tác vụ một các đồng thời bằng cách chuyển đổi qua lại giữa các tác vụ, quy định thời gian thực hiện và mức ưu tiên của các tác vụ đảm bảo tính thời gian thực.

1.2 Hệ điều hành thời gian thực FreeRTOS trên STM32

FreeRTOS là một hệ điều hành nhúng thời gian thực (RTOS) mã nguồn mở được phát triển bởi Real Time Engineers Ltd, sáng lập và sở hữu bởi Richard Barry. FreeRTOS được thiết kế phù hợp cho nhiều hệ nhúng nhỏ gọn như chạy trên các bộ vi điều khiển vì nó chỉ triển khai rất ít các chức năng như: cơ chế quản lý bộ nhớ và tác vụ cơ bản, các hàm API quan trọng cho cơ chế đồng bộ. Nó không cung cấp sẵn các giao tiếp mạng, drivers, hay hệ thống quản lý tệp (file system) như những hệ điều hành nhúng cao cấp khác. Tuy vậy, FreeRTOS có nhiều ưu điểm, hỗ trợ nhiều kiến trúc vi điều khiển khác nhau, kích thước nhỏ gọn (4.3 Kbytes sau khi biên dịch trên ARM7), được viết bằng ngôn ngữ C và có thể sử dụng, phát triển với nhiều trình biên dịch C khác nhau (GCC, OpenWatcom, Keil, IAR, Eclipse, …), cho phép không giới hạn các tác vụ chạy đồng thời, không hạn chế quyền ưu tiên thực thi, khả năng khai thác phần cứng. Ngoài ra, nó cũng cho phép triển khai các cơ chế điều độ giữa các tiến trình như: queues, counting semaphore, mutexes. Chú ý Ở trong FreeRTOS thì "task" là "thread".

FreeRTOS là RTOS phổ biến nhất trên thị trường hiện nay, cung cấp mã nguồn mở, miễn phí. Đây là hệ điều hành thời gian thực có cộng đồng phát triển rỗng rãi rất phù hợp cho nghiên cứu, học tập về các kỹ thuật, công nghệ trong viết hệ điều hành nói chung và hệ điều hành nhúng thời gian thực nói riêng, cũng như việc phát triển mở rộng tiếp các thành phần cho hệ điều hành hành (bổ sung modules, driver, thực hiện porting)

FreeRTOS có thể thực thi trên rất nhiều các dòng vi điều khiển khác nhau như STM, PIC, AVR, MSP, LPC... đặc biệt phát triển mạnh mẽ trên các chip lõi ARM. Hãng sản xuất chíp ST Microelectronics đã hỗ trợ và kết hợp FreeRTOS trên nền tảng thư viện CubeHAL cho các dòng chip STM32F1/F2/F3/F4/F7. Bằng cách xây dựng lớp thư viện CMSIS-RTOS kết hợp nền tảng thư viện CubeHAL người lập trình có thể ứng dụng FreeRTOS trên các vi điều khiển STM32 thông qua các API CMSIS-RTOS.

Các API CMSIS-RTOS điều khiển FreeRTOS

- Hàm **osThreadDef_t**

Nguyên mẫu :

osThreadDef(name, thread, priority, instances, stacksz)

Mô tả chức năng : Khai báo một tác vụ

Tham số :

- name: tên tác vụ
- thread: tên hàm thực hiện tác vụ
- priority: mức ưu tiên của tác vụ

Bảng 7.2.1: Bảng mức ưu tiên của tác vụ

Mức ưu tiên	Mô tả
osPriorityIdle	Mức ưu tiên thấp nhất
osPriorityLow	Mức ưu tiên thấp
osPriorityBelowNormal	Mức ưu tiên dưới mức bình thường
osPriorityNormal	Mức ưu tiên bình thường
osPriorityAboveNormal	Mức ưu tiên trên mức bình thường
osPriorityHigh	Mức ưu tiên cao
osPriorityRealtime	Mức ưu tiên cao nhất

- instances: mặc định là 0
- stacksz: kích thước của **tác** vụ

- Hàm **osThreadCreate**

Nguyên mẫu :

osThreadCreate (const osThreadDef_t *thread_def, void *argument)

Mô tả chức năng : Khởi tạo một tác vụ

Tham số :

- thread_def : tên tác vụ cần tạo
- argument : NULL

- Hàm **StartDefaultTask**

Nguyên mẫu :

StartDefaultTask(void const * argument)

Mô tả chức năng : Hàm thực hiện DefaultTask, các chương trinh của task sẽ được viết tại đây

Tham số :

- argument : NULL

- Hàm **StartTask02**

Nguyên mẫu :

StartTask02(void const * argument)

Mô tả chức năng : Hàm thực hiện Task02, các chương trinh của task sẽ được viết tại đây

Tham số :

- argument : NULL

- Hàm **osDelay**

Nguyên mẫu :

osStatus osDelay (uint32_t millisec)

Mô tả chức năng :

Hàm làm tác vụ ngủ trong một khoảng thời gian

Tham số :

- Millisec : thời gian ngủ của tác vụ đơn vị ms

- Hàm **osThreadSetPriority**

Nguyên mẫu :

osThreadSetPriority(osThreadId thread_id, osPriority priority);

Mô tả chức năng : Hàm thiết lập mức ưu tiên cho tác vụ

Tham số :

- thread_id : tên tác vụ
- priority: mức ưu tiên

- Hàm **osThreadGetPriority**

Nguyên mẫu :

osThreadGetPriority(osThreadId thread_id);

Mô tả chức năng : Hàm đọc ra mức ưu tiên hiện tại của tác vụ

Tham số :

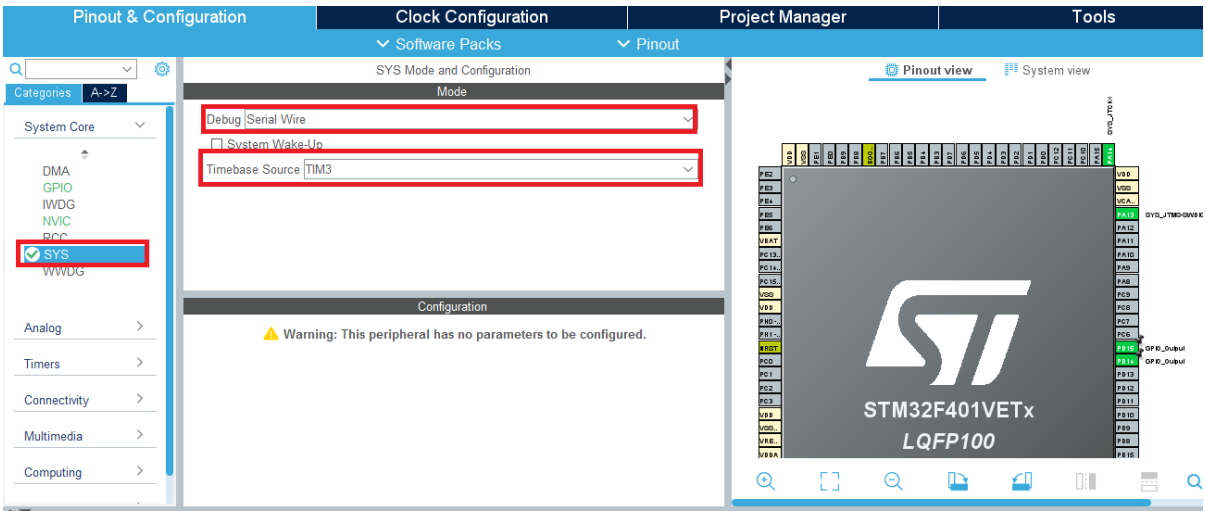
- thread_id : tên tác vụ

2. Cài đặt hệ điều hành thời gian thực FreeRTOS trên STM32CubeMx

Bước 1:

- Bật chế độ Debug Serial Wire, do sysTick được sử dụng cho FreeRTOS do vậy cần chọn nguồn xung clock cho timebase từ các bộ timer trong ví dụ này sử dụng TIM3

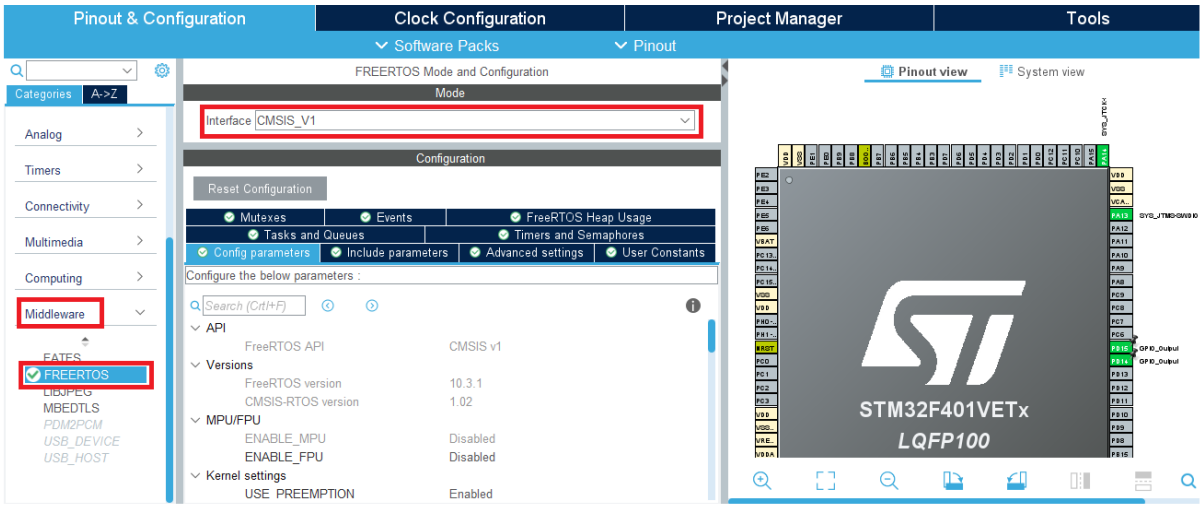
- Khởi tạo các tài nguyên cần sử dụng(GPIO, ADC, PWM, UART...) trong ví dụ này khởi tạo PD14, PA15 Output Push Pull để điều khiển LED



Hình 7: Khởi tạo các tài nguyên cần sử dụng

Bước 2: Thêm FreeRTOS vào chương trình

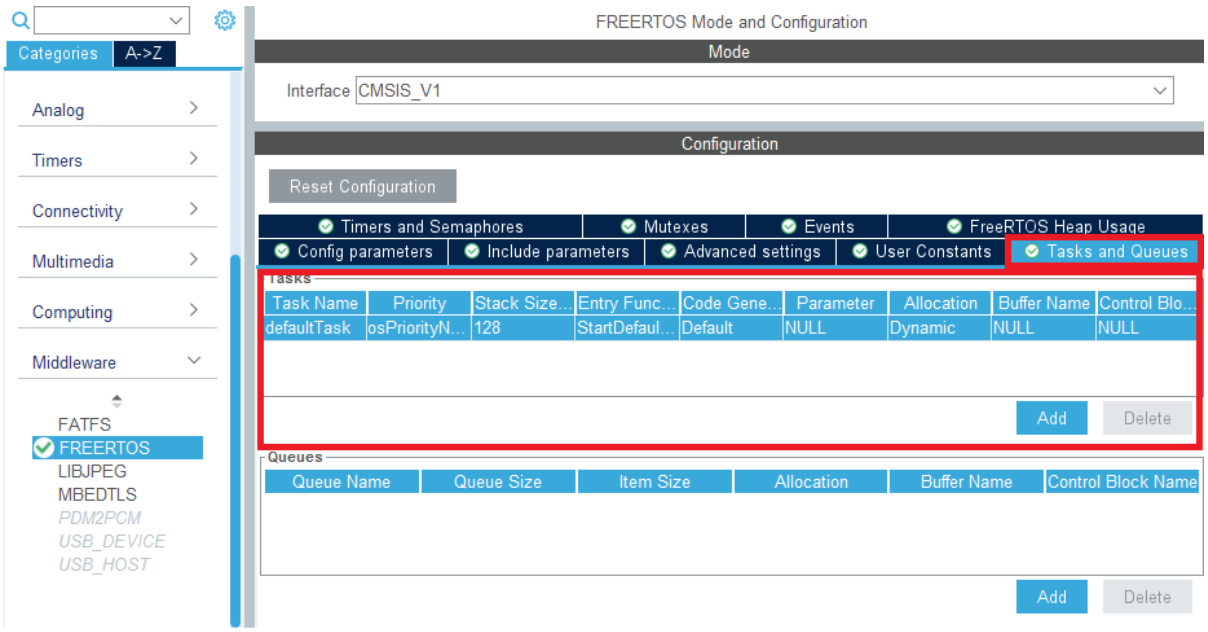
Chọn Middleware ->FreeRTOS ->CMSIS_V1



Hình 8: Add FreeRTOS vào chương trình

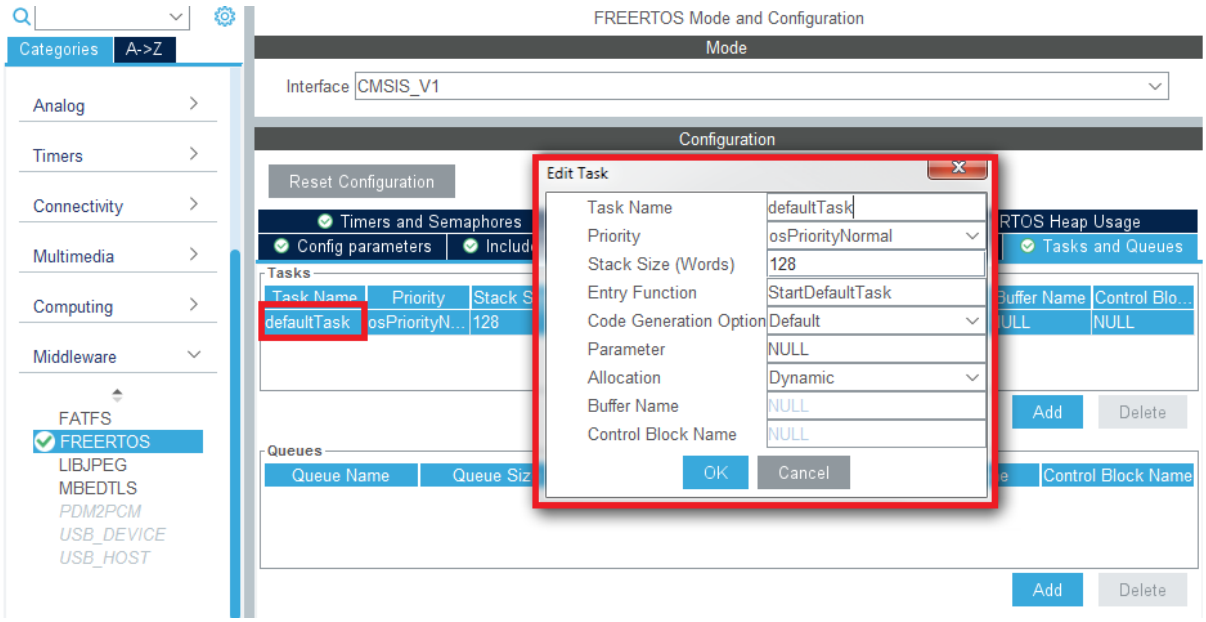
Bước 3: Thiết lập các tác vụ(task)

Chọn Tasks and Queues, trong mục Tasks chứa danh mục các task



Hình 9: Danh mục tác vụ

Theo mặc định sinh ra một task với tên là defaultTask. Kích đúp vào tên task để xem và chỉnh sửa các tham số cho task



Hình 10: Các tham số của tác vụ

Các tham số có thể chỉnh sửa gồm:

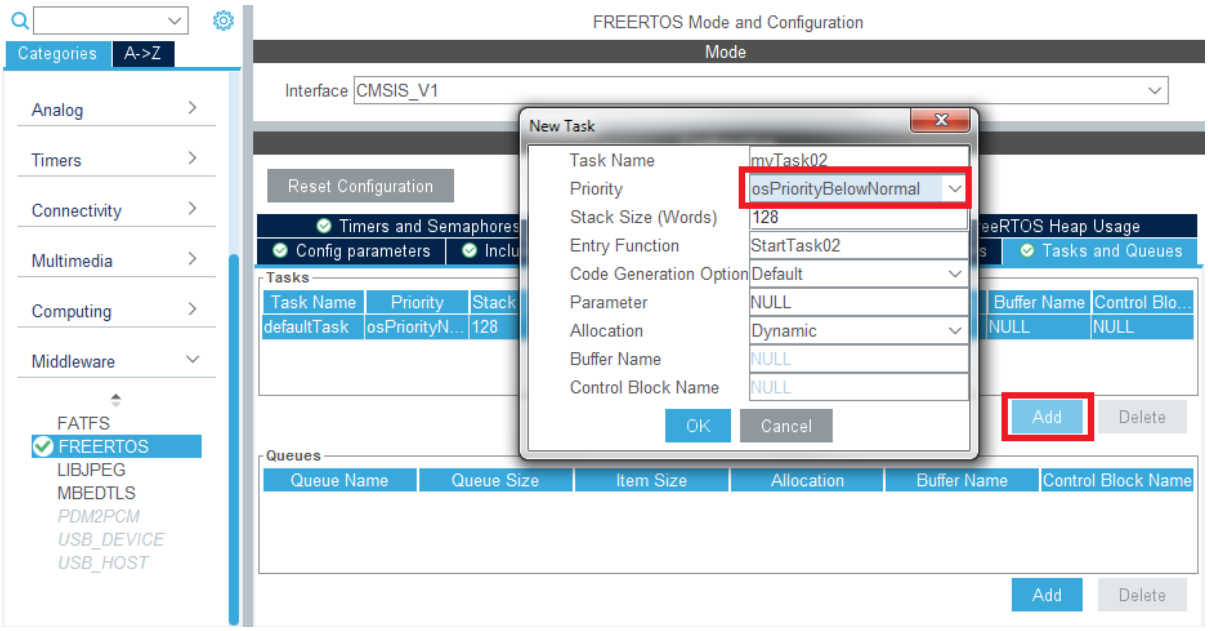
- **Task Name:** tên tác vụ, có thể tên tùy chọn đặt tên khác
- **priority:** đặt mức ưu tiên của tác vụ, có 7 mức ưu tiên(xem bảng 1), trong ví dụ này chọn mức ưu tiên osPriorityNormal

- **Stack Size:** kích thước của tác vụ, mặc định là 128
- **Entry Function:** tên hàm thực hiện task có thể tùy chọn đặt tên khác, nơi sẽ viết chương trình của task

Để thêm tác vụ chọn Add, sau đó chỉnh sửa các tham số cho tác vụ, chỉnh sửa mức ưu tiên cho task2

- **priority:** chọn mức ưu tiên osPriorityBelowNormal

Các tham số này sẽ được truyền vào hàm **osThreadDef_t** để khai báo các tham số cho task



Hình 11: Chỉnh sửa tham số tác vụ 2

Như vậy FreeRTOS sẽ quản lý 2 task gồm defaultTask và myTask02 trong đó defaultTask sẽ có mức ưu tiên cao hơn myTask02

Bước 4: Thiết lập xung clock và tạo Project tương tự như các bài học trước.

Khung chương trình lập trình FreeRTOS

Các hàm chức năng thực hiện các tác vụ được viết trong main.c

```
#include "main.h"
#include "cmsis_os.h"
osThreadId defaultTaskHandle;
osThreadId myTask02Handle;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
void StartDefaultTask(void const * argument);
void StartTask02(void const * argument);
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0,
128);
  defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);
  osThreadDef(myTask02, StartTask02, osPriorityBelowNormal, 0,
128);
  myTask02Handle = osThreadCreate(osThread(myTask02), NULL);
  osKernelStart();
  while (1)
  {  }
}
void StartDefaultTask(void const * argument)
{
  for(;;)
  {
          // chương trình viết ở đây
    osDelay(1);
  }
}
void StartTask02(void const * argument)
{
  for(;;)
  {
          // chương trình viết ở đây
    osDelay(1);
  }
}
```

Khai báo ngắt của tác vụ

Khai báo hàm thực hiện tác vụ

Khởi tạo defaultTask

Khởi tạo myTask02

Bắt đầu chạy

Hàm thực hiện các nhiệm vụ của defaultTask

Hàm thực hiện các nhiệm vụ của mytask

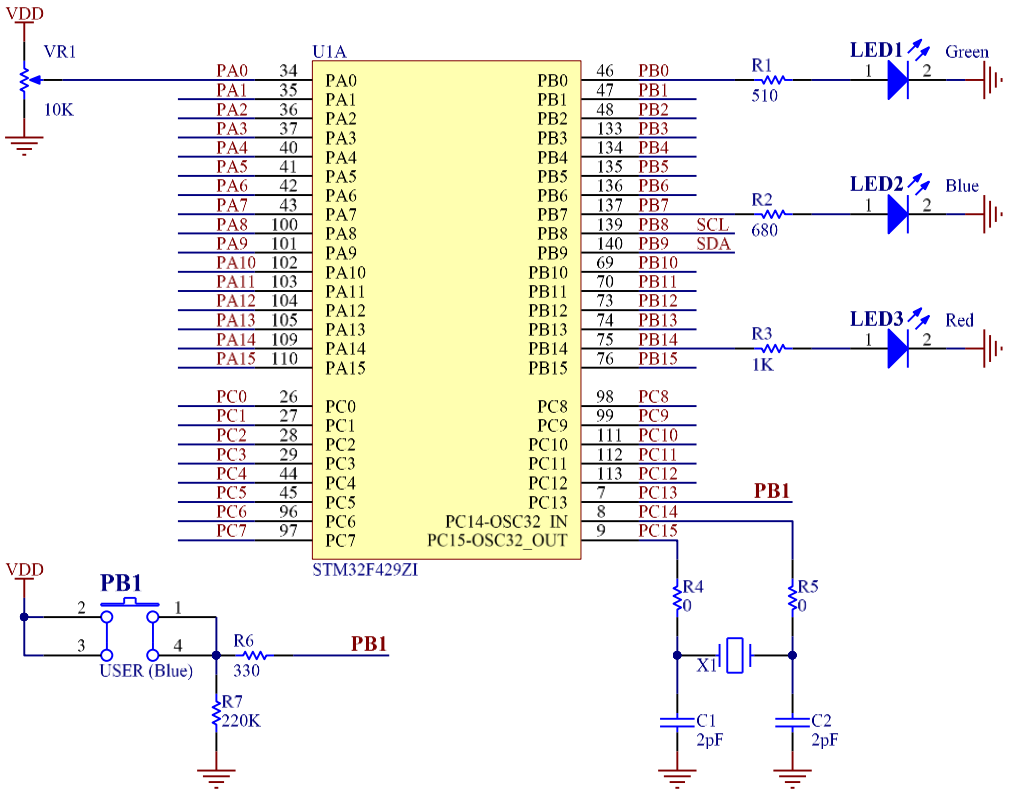
3. Thiết kế ứng dụng sử dụng hệ điều hành thời gian thực FreeRTOS trên STM32CubeMx

Ví dụ 1 : Viết chương trình sử dụng hệ điều hành FreeRTOS theo yêu cầu như sau:
- Sử dụng 3 task vụ

- Task1 điều khiển LED1 – PB0 nhấp nháy với chu kỳ 1s.

- Task2 điều khiển LED2 – PB7 nhấp nháy với chu kỳ 200ms.

- Task3 điều khiển LED3 – PB14 nhấp nháy với chu kỳ 4s.



Chương trình:

- Thiết lập chiều ra dữ liệu trên chân PB0, PB7, PB14.
- Chương trình chính

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();

  /* Create the thread(s) */
  /* definition and creation of defaultTask */
```

```
  osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0,
128);
  defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

  /* definition and creation of myTask02 */
  osThreadDef(myTask02, StartTask02, osPriorityNormal, 0, 128);
  myTask02Handle = osThreadCreate(osThread(myTask02), NULL);

  /* definition and creation of myTask03 */
  osThreadDef(myTask03, StartTask03, osPriorityNormal, 0, 128);
  myTask03Handle = osThreadCreate(osThread(myTask03), NULL);

  osKernelStart();
  while (1)
  {
  }
}
```

- Chương trình trên các Task

```
void StartDefaultTask(void const * argument){
  for(;;){
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
          HAL_Delay(500);
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
         HAL_Delay(500);
          osDelay(1);
  }
}

void StartTask02(void const * argument) {
  for(;;){
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_SET);
          HAL_Delay(100);
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_RESET);
         HAL_Delay(100);
          osDelay(1);
  }
}
```

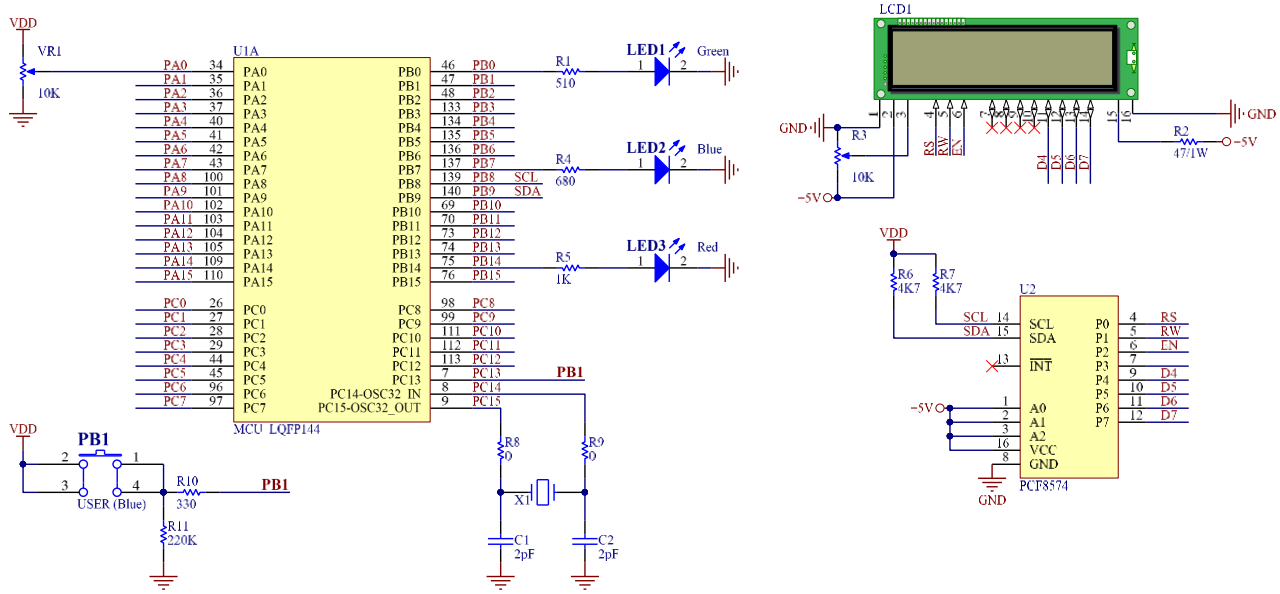
```
void StartTask03(void const * argument) {
  for(;;){
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);
          HAL_Delay(2000);
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET);
         HAL_Delay(2000);
          osDelay(1);
  }
}
```

Ví dụ 2 : : Viết chương trình sử dụng hệ điều hành FreeRTOS theo yêu cầu như sau:

- Sử dụng 2 task vụ

- Task1 điều khiển LED1 – PB0 nhấp nháy với chu kỳ 1s.

- Task2 hiển thị chuỗi kí tự và biến số lên LCD16x2 qua giao tiếp I2C



Chương trình:

```
#include "main.h"
#include "cmsis_os.h"
#include "stdio.h"
#include "string.h"

I2C_HandleTypeDef hi2c1;

osThreadId defaultTaskHandle;
osThreadId myTask02Handle;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
```

```
static void MX_I2C1_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_ADC1_Init(void);
void StartDefaultTask(void const * argument);
void StartTask02(void const * argument);
```

#define Dia_chi_LCD 0x4E // Địa chỉ mặc định của modul I2C PCF8574 giao tiếp LCD 16x2

```
char Dulieu_truyen_LCD[100];

void Lcd_Ghi_Lenh (char lenh)
{
  char data_u, data_l;
  uint8_t data_t[4];
  data_u = (lenh&0xf0);
  data_l = ((lenh<<4)&0xf0);
  data_t[0] = data_u|0x0C;  //en=1, rs=0
  data_t[1] = data_u|0x08;  //en=0, rs=0
  data_t[2] = data_l|0x0C;  //en=1, rs=0
  data_t[3] = data_l|0x08;  //en=0, rs=0
  HAL_I2C_Master_Transmit (&hi2c1, Dia_chi_LCD,
                           (uint8_t *) data_t, 4, 100);
}

void Lcd_Ghi_Dulieu (char data)
{
  char data_u, data_l;
  uint8_t data_t[4];
  data_u = (data&0xf0);
  data_l = ((data<<4)&0xf0);
  data_t[0] = data_u|0x0D;  //en=1, rs=1
  data_t[1] = data_u|0x09;  //en=0, rs=1
  data_t[2] = data_l|0x0D;  //en=1, rs=1
  data_t[3] = data_l|0x09;  //en=0, rs=1
  HAL_I2C_Master_Transmit (&hi2c1, Dia_chi_LCD,
                           (uint8_t *) data_t, 4, 100);
}

void lcd_init (void)
{
  Lcd_Ghi_Lenh (0x03);
  HAL_Delay(50);
  Lcd_Ghi_Lenh (0x02);
  HAL_Delay(50);
  Lcd_Ghi_Lenh (0x06);
  HAL_Delay(50);
  Lcd_Ghi_Lenh (0x0c);
  HAL_Delay(50);
  Lcd_Ghi_Lenh (0x28);
```

```
  HAL_Delay(50);
  Lcd_Ghi_Lenh (0x80);
}

void Lcd_Ghi_Chuoi (char *str)
{
  while (*str) Lcd_Ghi_Dulieu (*str++);
}

void Lcd_xoa_manhinh (void)
{

  Lcd_Ghi_Lenh (0x01); //xoa man hinh
}
int main(void)
{
  HAL_Init();
  SystemClock_Config();

  MX_GPIO_Init();
  MX_DMA_Init();
  MX_I2C1_Init();
  MX_USART1_UART_Init();
  MX_ADC1_Init();

  lcd_init();
  Lcd_xoa_manhinh();
  HAL_Delay(50);
  Lcd_Ghi_Lenh(0x80);//Dong 1
  Lcd_Ghi_Chuoi("KHOA DIEN TU");
  Lcd_Ghi_Lenh(0xC0);//Dong 2
  Lcd_Ghi_Chuoi("HAUI");
  HAL_Delay(2000);
  Lcd_xoa_manhinh();
  HAL_Delay(50);
  /* Create the thread(s) */
  /* definition and creation of defaultTask */
  osThreadDef(defaultTask,StartDefaultTask,osPriorityNormal,0, 128);
  defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

  /* definition and creation of myTask02 */
  osThreadDef(myTask02, StartTask02, osPriorityNormal, 0, 128);
  myTask02Handle = osThreadCreate(osThread(myTask02), NULL);

  /* definition and creation of myTask03 */
  osThreadDef(myTask03, StartTask03, osPriorityNormal, 0, 128);
  myTask03Handle = osThreadCreate(osThread(myTask03), NULL);
  osKernelStart();

  while (1)
```

```
  {
  }

}
```

- Chương trình trên các Task

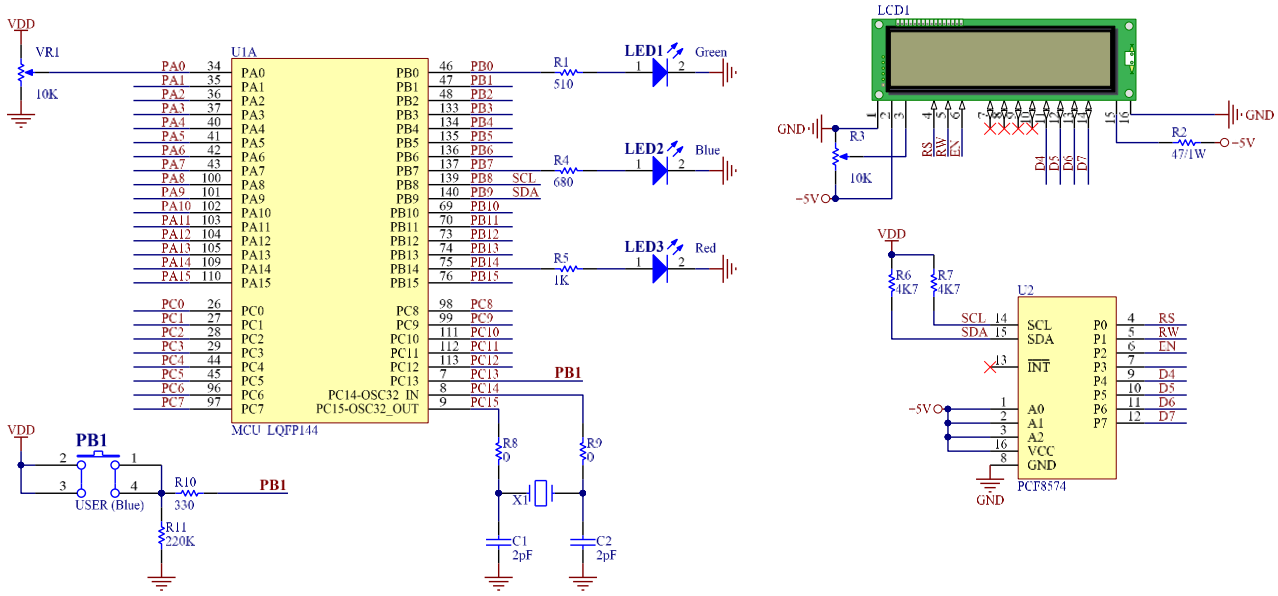
```
void StartDefaultTask(void const * argument){
  for(;;){
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
          HAL_Delay(500);
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
         HAL_Delay(500);
          osDelay(1);
  }
void StartTask03(void const * argument)
{
  for(;;)
  {
     Lcd_Ghi_Lenh(0x80);//Dong 1
     sprintf(Dulieu_truyen_LCD," Hien thi LCD  ");
     Lcd_Ghi_Chuoi(&Dulieu_truyen_LCD[0]);

     Lcd_Ghi_Lenh(0xC0);//Dong 2
     sprintf(Dulieu_truyen_LCD," FreeRTOS ");
     Lcd_Ghi_Chuoi(&Dulieu_truyen_LCD[0]);
    osDelay(1);
  }
}
```

Ví dụ 3 :

Viết chương trình sử dụng hệ điều hành FreeRTOS theo yêu cầu như sau:

- Sử dụng mạch điện của ví dụ 2
- Sử dụng ngắt ngoài đếm số lần nhấn của nút PB1
- Sử dụng 2 task vụ;
    + Task1 điều khiển LED1 – PB0 nhấp nháy với chu kỳ 1s.
    + Task2 hiển thị số lần nhấn của nút PB1 lên LCD16x2 qua giao tiếp I2C



Chương trình:

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();

  MX_GPIO_Init();
  MX_DMA_Init();
  MX_I2C1_Init();
  MX_USART1_UART_Init();
  MX_ADC1_Init();

  lcd_init();
  Lcd_xoa_manhinh();
  HAL_Delay(50);
  Lcd_Ghi_Lenh(0x80);//Dong 1
  Lcd_Ghi_Chuoi("KHOA DIEN TU");
  Lcd_Ghi_Lenh(0xC0);//Dong 2
  Lcd_Ghi_Chuoi("HAUI");
  HAL_Delay(2000);
  Lcd_xoa_manhinh();
  HAL_Delay(50);
  /* Create the thread(s) */
  /* definition and creation of defaultTask */
  osThreadDef(defaultTask,StartDefaultTask,osPriorityNormal,0, 128);
  defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

  /* definition and creation of myTask02 */
  osThreadDef(myTask02, StartTask02, osPriorityNormal, 0, 128);
  myTask02Handle = osThreadCreate(osThread(myTask02), NULL);

  /* definition and creation of myTask03 */
  osThreadDef(myTask03, StartTask03, osPriorityNormal, 0, 128);
```

```
  myTask03Handle = osThreadCreate(osThread(myTask03), NULL);
  osKernelStart();

  while (1)
  {
  }

}
```

- Chương trình đọc nút nhấn sử dụng ngắt ngoài

```
  int dem;
  void EXTI0_IRQHandler(void)
  {
      dem=dem+1;
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  }
```

- Chương trình trên các Task

```
void StartDefaultTask(void const * argument){
  for(;;){
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
          HAL_Delay(500);
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
         HAL_Delay(500);
          osDelay(1);
  }
void StartTask03(void const * argument)
{
  for(;;)
  {
     Lcd_Ghi_Lenh(0x80);//Dong 1
     sprintf(Dulieu_truyen_LCD," So lan nhan PB1");
     Lcd_Ghi_Chuoi(&Dulieu_truyen_LCD[0]);

     Lcd_Ghi_Lenh(0xC0);//Dong 2
     sprintf(Dulieu_truyen_LCD,"%d", dem);
     Lcd_Ghi_Chuoi(&Dulieu_truyen_LCD[0]);
    osDelay(1);
  }
}
```

- Các phần khởi tạo và thiết lập vào ra dữ liệu giống như các bài trước.

4. BÀI TẬP

Bài tập 1

Lập trình sử dụng hệ điều hành FreeRTOS: Điều khiển LED1 – PB0 sáng tắt chu kì 0,5s; đọc tín hiệu ADC trên ADC 1 – Kênh 0 (ADC1_IN0 – PA0). Hiển thị giá trị Volt của đầu vào ADC lên LCD 16x2.

Bài tập 2

Lập trình sử dụng hệ điều hành FreeRTOS: Sử dụng ngắt ngoài đọc số lần nhấn của nút PB1 – PC13; Điều khiển LED1 – PB0 sáng tắt chu kì 0,5s; đọc tín hiệu ADC trên ADC 1 – Kênh 0 (ADC1_IN0 – PA0). Hiển thị dòng thứ 1 giá trị Volt của đầu vào ADC lên LCD 16x2. Hiển thị dòng thứ 2 số lần nhấn của nút PB1 lên LCD 16x2.