

Universidade Federal de Pernambuco  
Centro de Tecnologia e Geociências  
Departamento de Engenharia Mecânica

### **Lista de Exercícios #03 - 2024.1**

O objetivo desta lista é que façamos nosso primeiro projeto com o FreeRTOS dentro da STM32 CubeIDE e comparar ele como um mesmo projeto produzido sem o auxílio do FreeRTOS.

## **1 Instruções Gerais**

- Sempre comente seu código e identifique o que ele faz e quem foi que fez (você, no caso);
- Coloque todas as pastas dos projetos criados em uma única pasta “LE03\_SEU\_NOME”, em que SEU\_NOME é o seu nome;
- Compacte a pasta e inclua como resposta da atividade.

## **2 Projeto com FreeRTOS**

Esta seção trará como configurar um projeto com o FreeRTOS, como criar uma *task* e como fazer a contagem de tempo pelo sistema operacional (SO).

### **2.1 Configurando um novo Projeto**

Nesta etapa, vamos primeiramente refazer as configurações básicas que foram feitas na aula anterior:

1. Crie um novo projeto a partir de um arquivo de configuração de outro projeto:  
**File → New → STM32 Project from a Existing STM32CubeMx Configuration file (.ioc).**
2. Coloque o nome do projeto LE03\_Q1\_SEU\_NOME, em que SEU\_NOME é o seu nome.
3. Compile (*Build*) o projeto para garantir que todas as configurações estão corretas.
4. Compile novamente e grave na placa para garantir que todas as configurações estão como desejado;

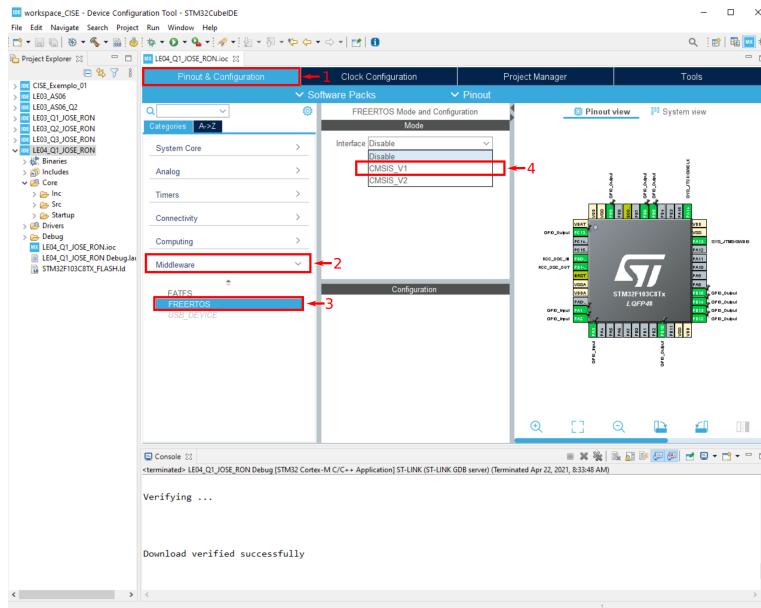
Se tiver dúvida em qualquer desses passos, revisite a LE02 e o material associado.

### **2.2 Configurando o FreeRTOS**

Para habilitar o FreeRTOS, vá para a perspectiva de configuração do projeto e acesse:

1. **Pinout & Configuration;**
2. **Middleware;**
3. **FREERTOS;**

Figura 1: Selezionando o FreeRTOS na STM32 CubeIDE.



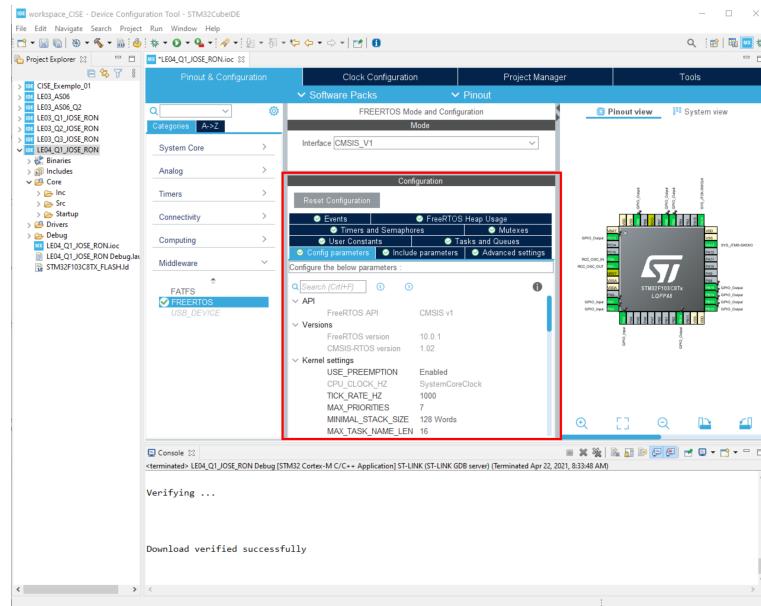
Fonte - Produzido pelo Autor.

#### 4. Em **FREERTOS Mode and Configuration** selecione na lista suspensa de **Interface** a opção **CMSIS\_V1**.

Como mostrado na Figura 1.

Uma vez selecionado o **CMSIS\_V1**, será disponibilizada a aba de configuração dos parâmetros do FreeRTOS, como destacado na Figura 2.

Figura 2: Ambiente de configuração do FreeRTOS na STM32 CubeIDE.



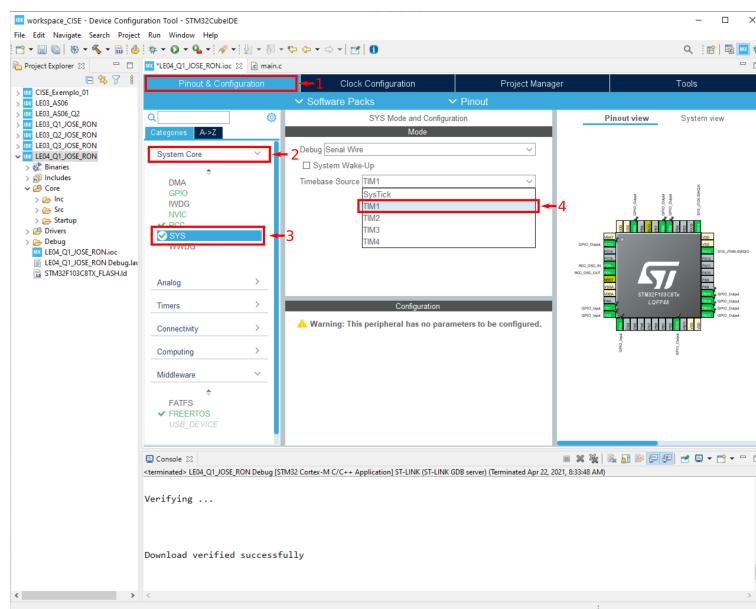
Fonte - Produzido pelo Autor.

Iremos utilizar esta área diversas vezes ao longo do semestre a medida que precisemos modificar alguma configuração ou mesmo acrescentar algum recurso. Por agora, o que precisamos é trocar qual o temporizador base utilizado pelo sistema. Para isso, siga o seguinte passo-a-passo:

1. Pinout & Configuration;
2. System Core;
3. SYS;
4. Em **SYS Mode and Configuration** selecione na lista suspensa de **Timebase Source** a opção **TIM1**.

Como mostrado na Figura 3. É possível utilizar qualquer um dos *timers*, apenas não é recomendado o uso do **SysTick**, com será indicado pela própria IDE caso você tente usá-lo.

Figura 3: Configurando o temporizador do sistema na STM32 CubeIDE.



Fonte - Produzido pelo Autor.

Figura 4: Código gerado ao adicionar o FreeRTOS a um projeto na STM32 CubeIDE.

```

/* Private variables -----
osThreadId defaultTaskHandle;
int main(void)
{
    /* Create the thread(s) */ /* definition and creation of defaultTask */
    osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
    defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

    /* Start scheduler */
    osKernelStart();

    /* We should never get here as control is now taken by the scheduler */
} /* end main

**/ StartDefaultTask function */
void StartDefaultTask(void const * argument)
{
    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    for(;;)
    {
        FOR INFO: NOT TOOL-GENERATED.
        THIS IS WHERE YOUR CODE IS INSERTED
        /* USER CODE END 5 */
    }
}

```

Fonte - [1].

Após a configuração da base de tempo do sistema, salve todo o projeto para que a CubeIDE gere o código necessário. Se observarmos o código gerado pela IDE, é possível identificar a mudança nos quatro pontos mostrados na Figura 4. Na Figura 4:

1. `osThreadId defaultTaskHandle;` → cria um ponteiro que receberá a *thread* que será criada;
2. `osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);` → define os parâmetros da *thread* e  
`defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);` → cria a *thread* e faz o ponteiro criado anteriormente apontar para ela;
3. `osKernelStart();` → inicializa o gerenciador de *threads* (o *scheduler*, veremos mais detalhes sobre ele na próxima aula).
4. Definição da “função” que será chamada sempre que a *thread* for ser executada. Cada *thread* irá possuir duas partes, como mostrado na Figura 5:
  - (a) Parte inicial da *thread*: só irá executar na primeira vez que a *thread* for chamada pelo *scheduler*. Esta parte da *thread* é utilizada para definição de variáveis e configurações iniciais desta parte do código;
  - (b) *Loop infinito*: esta parte do código será executada em um laço infinito enquanto a *thread* existir, da mesma forma que o nosso conhecido “`while(1)`” de quando estamos programando em *bare-metal*.

Mais sobre cada uma dessas funções pode ser visto na documentação do FreeRTOS [2] e na documentação desenvolvida pela ST [3].

## 2.3 Uma única Task

Utilizando a *task* que já nos foi criada, crie um código que fará com que o led P13 da Blue Pill fique 100 ms ligado a cada 2 segundos. Para isso utilize a função `osDelay(TEMPO)`, em que `TEMPO` é o tempo em ms que você quer o SO espere sem fazer nada.

Figura 5: Código base de uma *thread* no FreeRTOS.

```

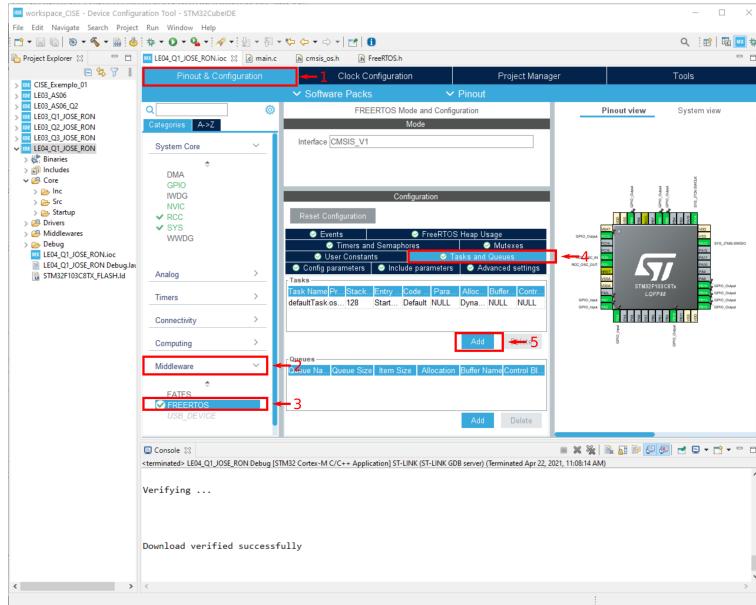
229 /* USER CODE END Header_StartDefaultTask */
230 void StartDefaultTask(void const * argument)
231 {
232     /* USER CODE BEGIN 5 */
233     /*
234     *
235     * Parte inicial
236     *
237     */
238     /* Infinite loop */
239     for(;;)
240     {
241         /*
242         *
243         * Loop infinito
244         *
245         */
246     }
247     /* USER CODE END 5 */
248 }
```

Fonte - Produzido pelo autor.

## 2.4 Trabalhando com múltiplas *tasks*

Vamos criar agora uma outra *thread*, para isso, mude para perspectiva de configuração na CubeIDE e siga o passo-a-passo mostrado na Figura 6:

Figura 6: Como criar uma *thread* na STM32 CubeIDE.



Fonte - Produzido pelo autor.

### 1. Pinout & Configuration;

### 2. Middleware;

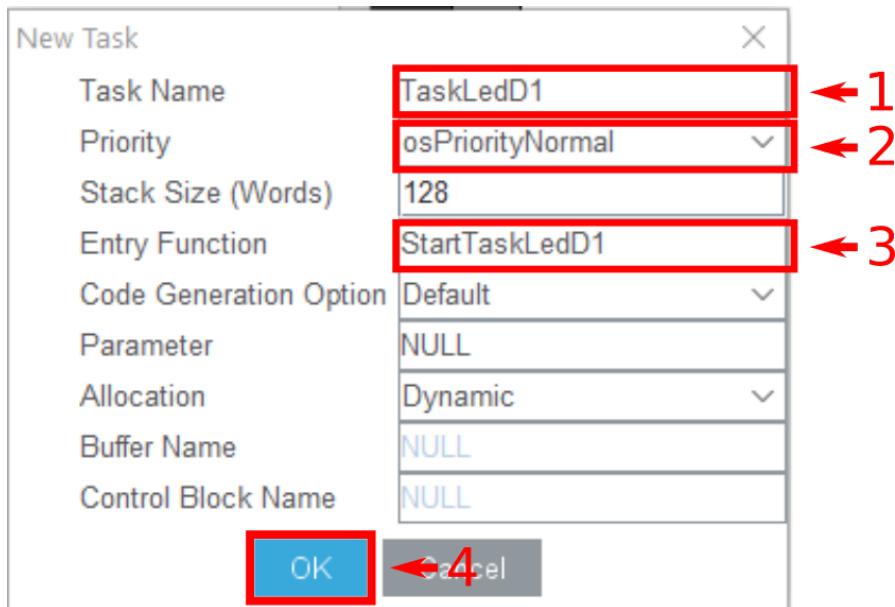
### 3. FREERTOS;

### 4. Tasks and Queues;

### 5. Add.

Uma janela pop-up irá abrir para que as configurações da *task* seja feita, deixe como mostrado na Figura 7

Figura 7: Configurando uma nova *thread* na STM32 CubeIDE.



Fonte - Produzido pelo autor.

1. **Task Name:** TaskLedD1;
2. **Priority:** osPriorityNormal;
3. **Entry Function:** StartTaskLedD1;
4. Salve as configurações clicando em **Ok**;

Salve o projeto para que o código seja gerado. Observe o código gerado seguindo mesmo padrão mostrando na Figura 4.

#### 2.4.1 Exercícios Propostos

- Utilize a TaskLedD1 para fazer o led D1 piscar invertendo de valor a cada 1000ms;
- Seguindo o mesmo passo-a-passo da Seção 2.4, crie uma nova *task* para fazer o led D2 piscar ficando ligado 50ms a cada 450 ms e o led D3 piscar ficando desligado 50ms a cada 450 ms
- Seguindo o mesmo passo-a-passo da Seção 2.4, crie uma nova *task* para fazer o led D4 piscar ficando ligado 100ms a cada 2s<sup>1</sup>;

<sup>1</sup>**OBS:** Na configuração que estamos usando para o FreeRTOS, só podemos criar quatro *threads*. Veremos com mais detalhes em aulas posteriores os modos e o espaço que o FreeRTOS separa para cada *task* e como aumentar o número de *threads*.

## 2.5 Delay não blocante no FreeRTOS

Como ocorre com as funções HAL\_Delay( ) e HAL\_GetTick(), em que a primeira gera um tempo em que o microcontrolador fica sem executar nenhuma instrução e a segunda pode ser usada para que a contagem de tempo não seja blocante. No FreeRTOS, a função osDelay() é blocante, pode não parecer no cenário que estamos mostrando graças ao *scheduler* do SO e ao fato que as funções que mudam o valor dos leds consomem um tempo irrisório comparado pelos tempos de espera dos *delays* impostos.

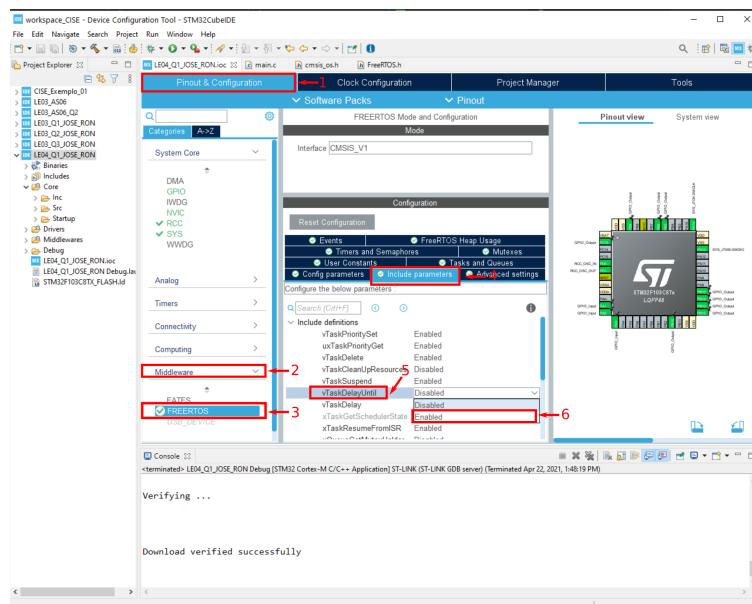
No entanto, para códigos que precisam ser chamados em tempos precisos e que levam um tempo que precisa ser considerado para executar, o comportamento do sistema pode ficar longe do ideal.

Para ver isso, adicione na *task* que lida com o led D1 um *for* que não faça nada, apenas introduza um *delay* no código:

```
1 /* USER CODE BEGIN StartTaskLedD1 */
2     uint32_t i = 0;
3     /* Infinite loop */
4     for (;;)
5     {
6         HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_15); // inverte o valor do led;
7         for (i = 0; i < 1000000; ++i)
8         {
9             // loop fazendo nada;
10        }
11        osDelay(100); // espera 100ms;
12    }
13 /* USER CODE END StartTaskLedD1 */
```

Veja que o código já não se comporta como o desejado. Para podermos chamar uma *task* com um período mais preciso e que independe do tempo de execução do código da *task* nós usamos a função osDelayUntil(), mas para poder utilizar essa função é necessário habilitar seu uso nas configurações do FreeRTOS, para isso, siga o passo-a-passo mostrado na Figura 8:

Figura 8: Configurando a função vTaskUntil na STM32 CubeIDE.



Fonte - Produzido pelo autor.

## 1. Pinout & Configuration;

## 2. Middleware;

## 3. FREERTOS;

## 4. Include parameters;

## 5. vTaskDelayUntil;

## 6. Selecionar Enabled na lista suspensa;

Tanto faz usar a função osDelayUntil() ou a função vTaskDelayUntil(), na verdade a primeira só chama a segunda.

### 2.5.1 Exercício Proposto

- Crie um novo projeto a partir de um arquivo de configuração de outro projeto LE03\_Q1\_SEU\_NOME: **File → New → STM32 Project from a Existing STM32CubeMx Configuration file (.ioc)**.
- Coloque o nome do projeto LE03\_Q2\_SEU\_NOME, em que SEU\_NOME é o seu nome.
- Refaça todas as *tasks* do exercício anterior, só que a agora usando a função osDelayUntil(). Por exemplo, para o código da TaskLedD1 teríamos:

```
1 /* USER CODE BEGIN StartTaskLedD1 */
2     TickType_t TaskTimeStamp;           // Guarda o Tempo
3     TaskTimeStamp = xTaskGetTickCount(); // pega o valor atual do contador
4     /* Infinite loop */
5     for(;;)
6     {
7         HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_15); // inverte o valor do led;
8         osDelayUntil(&TaskTimeStamp, 100);
9     }
/* USER CODE END StartTaskLedD1 */
```

- Adicione o `for()` em alguma das *tasks* e veja o que acontece. Brinque com os valores de parada do `for`.

## 3 De volta ao Bare-Metal

Refaça o projeto anterior sem utilizar o FreeRTOS e veja se é mais simples ou não de garantir as temporizações. Chame o projeto de LE03\_Q3\_SEU\_NOME, em que SEU\_NOME é o seu nome.

## Referências

- [1] J. Cooling, *Real-time Operating Systems: Book 2 - The Practice (Using STM Cube, FreeRTOS and the STM32 Discovery Board)*. Lindentree Associates, 2018.
- [2] FreeRTOS, “Api reference.” Disponível em: <https://www.freertos.org/a00106.html>, 2021. Acessado em 22/04/2021.
- [3] ST, *UM1722 User manual: Developing applications on STM32Cube with RTOS*, rev 3 ed., Oct 2019. Disponível em: [https://www.st.com/resource/en/user\\_manual/dm00105262-developing-applications-on-stm32cube-with-rtos-stmicroelectronics.pdf](https://www.st.com/resource/en/user_manual/dm00105262-developing-applications-on-stm32cube-with-rtos-stmicroelectronics.pdf).