

## **Sistemas Embarcados**

# **FreeRTOS - Event Groups -**

**Tiago Piovesan Vendruscolo**



Esta licença permite que outros remixem, adaptem e criem a partir do trabalho para fins não comerciais, desde que atribuam o devido crédito aos autores originais. [4.0 international](https://creativecommons.org/licenses/by-nc-nd/4.0/)

# Event Groups

- São flags utilizadas para informar que um determinado evento ocorreu. Esse aviso é dado pela alteração de 1 ou mais bits. Por exemplo: quando o bit estiver definido em “1” significa que uma tarefa concluiu um determinado processamento e o resultado está disponível. Se estiver “0” é porque a tarefa ainda não possui o resultado.
- Os event groups são variáveis de 16 ou 32 bits, de acordo com o valor setado em `configUSE_16_BIT_TICKS` no arquivo `FreeRTOSConfig.h`. Se o valor for 1, a variável terá 16 bits com 8 bits disponíveis para uso. Se for setada em 0, ela terá 32 bits, com 24 bits disponíveis para o usuário. Os 8 bits não disponíveis são flags utilizadas pelo sistema do FreeRTOS. Exemplo de uso para flags: (R – reservado para o sistema, D – disponível, E – Erro na comunicação serial, T – resultado da tarefa x disponível)

BIT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	R	R	R	R	R	R	R	R	D	D	D	D	D	D	E	T

# Event Groups

- Diferentemente de filas e semáforos um event group:
  - *Permite que uma tarefa bloqueada aguarde até a ocorrência de um ou mais eventos.*
  - *Permite o desbloqueio de todas as tarefas que estavam aguardando o evento.*
- Vantagens:
  - *Sincronizar várias tarefas;*
  - *Reduzem o consumo de RAM, pois trabalha com poucos bits. É possível substituir semáforos binários por um grupo de eventos.*
  - *Ativa todas as tarefas que dependem da flag, não apenas a mais prioritária, a exemplo do que acontece com o semáforo, o que pode gerar Starvation da tarefa de menor prioridade.*

# Event Groups

- É necessário adicionar `event_groups.h`
- Criando um grupo de eventos

```
EventGroupHandle_t xEventGroupCreate( void );

/* Declare a variable to hold the created event group. */
EventGroupHandle_t xCreatedEventGroup;

/* Attempt to create the event group. */
xCreatedEventGroup = xEventGroupCreate();

/* Was the event group created successfully? */
if( xCreatedEventGroup == NULL )
{
    /* The event group was not created because there was insufficient
    FreeRTOS heap available. */
}
else
{
    /* The event group was created. */
}
```

<https://www.freertos.org/xEventGroupCreate.html>


# Event Groups

- Setando uma flag (bit)

As funções para uso em interrupções são específicas para interrupções

```
EventBits_t xEventGroupSetBits( EventGroupHandle_t xEventGroup,  
                                const EventBits_t uxBitsToSet );
```

Nome do evento



```
#define BIT_0    ( 1 << 0 )  
#define BIT_4    ( 1 << 4 )
```

```
/* Set bit 0 and bit 4 in xEventGroup. */  
uxBits = xEventGroupSetBits(  
    xEventGroup,      /* The event group being updated. */  
    BIT_0 | BIT_4 ); /* The bits being set. */
```

# Event Groups

- Aguardando um evento

```
EventBits_t xEventGroupWaitBits(  
    const EventGroupHandle_t xEventGroup,  
    const EventBits_t uxBitsToWaitFor,  
    const BaseType_t xClearOnExit,  
    const BaseType_t xWaitForAllBits,  
    TickType_t xTicksToWait );
```

1. Handle do grupo de eventos
2. Quais bits está aguardando
3. Se quer apagar os bits após o tratamento
4. Aguardar por todos os bits em 2.
5. *Timeout*

- No exemplo abaixo, a tarefa irá sair do estado de bloqueio quando ambos os eventos ocorrerem, ou então, quando o tempo de espera (500ms) expirar.

```
x = xEventGroupWaitBits(xEventos, evento1 | evento2, true, true, pdMS_TO_TICKS(500));
```

- Nesse outro exemplo, a tarefa irá sair do estado de bloqueio quando um dos eventos ocorrerem, ou então, quando o tempo de espera (500ms) expirar.

```
x = xEventGroupWaitBits(xEventos, evento1 | evento2, true, false, pdMS_TO_TICKS(500));
```

- Para esperar (ficar bloqueado) indefinidamente, utilize `portMAX_DELAY`.

# Event Groups

- Monitorando o event group

```
flag = xEventGroupWaitBits(xEventos, bit_flag, pdTRUE, pdTRUE, pdMS_TO_TICKS(50));  
if(flag)  
{  
    Faça algo aqui....  
}
```

- Outra forma...

```
if((xEventGroupWaitBits(xEventos, bit_flag, pdTRUE, pdTRUE, pdMS_TO_TICKS(50)))==pdTRUE)  
{  
    Faça algo aqui....  
}
```

# Event Groups

- Deletando um grupo de eventos – liberar memória

```
void vEventGroupDelete( EventGroupHandle_t xEventGroup );
```



- Exemplo 1: Faça um código que tenha 3 tarefas:
  - *Tarefa 1: Seja ativada pela flag “Tarefa1\_flag” e imprima na serial “Tarefa 1 ativa!”*
  - *Tarefa 2: Seja ativada pela flag “Tarefa2\_flag” e imprima na serial “Tarefa 2 ativa!”*
  - *Tarefa 3: Faz a leitura da serial, e, quando for digitado “Tarefa1” ele ative a tarefa 1, e quando for digitado “Tarefa2” ele ative a tarefa 2.*
  - *Código no moodle.*

```

#include "Arduino_FreeRTOS.h"
#include "task.h"
#include "event_groups.h"

/*mapeamento de eventos*/
//#define Tarefa1_flag (1<<0) //1
//#define Tarefa2_flag (1<<1) //10

#define Tarefa1_flag 0x01
#define Tarefa2_flag 0x02 //10

/* Variáveis para armazenamento do handle das tasks */
TaskHandle_t xTarefa1Handle;
TaskHandle_t xTarefa2Handle;
TaskHandle_t xTarefa3Handle;

EventGroupHandle_t xEventos;

/*protótipos das Tasks*/
void vTarefa1 (void *pvParameters);
void vTarefa2 (void *pvParameters);
void vTarefa3 (void *pvParameters);

void setup() {
    Serial.begin(9600);
    Serial.setTimeout(50);
    xEventos = xEventGroupCreate();
    xTaskCreate(vTarefa1, "vTarefa1", 256, NULL, 1, &xTarefa1Handle);
    xTaskCreate(vTarefa2, "vTarefa2", 256, NULL, 1, &xTarefa2Handle);
    xTaskCreate(vTarefa3, "vTarefa3", 256, NULL, 1, &xTarefa3Handle);
}

```

equivalentes

```
void loop() {
    //As funções são executadas nas tarefas
}
```

```
void vTarefa1(void *pvParameters){
    while(1)
    {
        xEventGroupWaitBits(xEventos, Tarefa1_flag, pdTRUE, pdTRUE, portMAX_DELAY);
        Serial.println("Tarefa 1 ativa!");
    }
}
```

Aguarda por todos os bits  
(nesse caso, apenas 1)

Apagar o bits após a leitura

```
void vTarefa2(void *pvParameters){
    while(1)
    {
        xEventGroupWaitBits(xEventos, Tarefa2_flag, pdTRUE, pdTRUE, portMAX_DELAY);
        Serial.println("Tarefa 2 ativa!");
    }
}
```

```
void vTarefa3(void *pvParameters){
    String x;
    while(1)
    {
        if(Serial.available() > 0){
            x = Serial.readString();
            if(x.equalsIgnoreCase("Tarefa1")){
                xEventGroupSetBits(xEventos, Tarefa1_flag);
            }
            if(x.equalsIgnoreCase("Tarefa2")){
                xEventGroupSetBits(xEventos, Tarefa2_flag);
            }
        }
    }
}
```

Não é case sensitive

- Exercício 1: Faça um código que tenha 4 tarefas:
  - *Tarefa 1: Seja ativada pela flag “Tarefa1\_flag” e imprima na serial “Tarefa 1 ativa!”*
  - *Tarefa 2: Seja ativada pela flag “Tarefa2\_flag” e imprima na serial “Tarefa 2 ativa!”*
  - *Tarefa 3: LED em 1 Hz*
  - *Tarefa 4: Faz a leitura de um botão, no momento em que ele é pressionado (nível LOW) ele ative a flag da tarefa 1, e, após 1 segundo ative a flag da tarefa 2. A ativação deve ser feita apenas 1 vez para cada vez que o botão é pressionado.*
    - Não deixe as 4 tarefas com pilha de 256 senão poderá não funcionar corretamente (falta de memória)

# Event Groups

```
#include "Arduino_FreeRTOS.h"
#include "task.h"
#include "event_groups.h"

#define LED1 7
#define BOTAO 2

/*mapeamento de eventos*/
#define Tarefa1_flag (1<<0) //1
#define Tarefa2_flag (1<<1) //10

/* Variáveis para armazenamento do handle das tasks */
TaskHandle_t xTarefa1Handle;
TaskHandle_t xTarefa2Handle;
TaskHandle_t xTarefa3Handle;
TaskHandle_t xTarefa4Handle;
EventGroupHandle_t xEventos;

/*protótipos das Tasks*/
void vTarefa1 (void *pvParameters);
void vTarefa2 (void *pvParameters);
void vTarefa3 (void *pvParameters);
void vTarefa4 (void *pvParameters);

void setup() {
    Serial.begin(9600);
    pinMode(BOTAO, INPUT_PULLUP);
    xEventos = xEventGroupCreate();
    xTaskCreate(vTarefa1, "vTarefa1", 256, NULL, 1, &xTarefa1Handle);
    xTaskCreate(vTarefa2, "vTarefa2", 256, NULL, 1, &xTarefa2Handle);
    xTaskCreate(vTarefa3, "vTarefa3", 128, NULL, 1, &xTarefa3Handle);
    xTaskCreate(vTarefa4, "vTarefa3", 128, NULL, 1, &xTarefa4Handle);
}
```

# Event Groups

```
void vTarefa1(void *pvParameters){
    while(1)
    {
        xEventGroupWaitBits(xEventos, Tarefa1_flag, pdTRUE, pdTRUE, portMAX_DELAY);
        Serial.println("Tarefa 1 ativa!");
    }
}
```

Aguarda por todos os bits  
(nesse caso, apenas 1)

Apagar o bits após a leitura

```
void vTarefa2(void *pvParameters){
    while(1)
    {
        xEventGroupWaitBits(xEventos, Tarefa2_flag, pdTRUE, pdTRUE, portMAX_DELAY);
        Serial.println("Tarefa 2 ativa!");
    }
}
```

```
void vTarefa3(void *pvParameters){
    pinMode(LED1, OUTPUT);
    while(1)
    {
        digitalWrite(LED1, !digitalRead(LED1));
        vTaskDelay(pdMS_TO_TICKS(500));
    }
}
```

# Event Groups

```
void vTarefa4(void *pvParameters){
    uint8_t debouncingContagem = 0;
    uint8_t x = 0;
    while(1)
    {
        if((digitalRead(BOTAO) == LOW) && x==0)
        {
            debouncingContagem++;
            if(debouncingContagem >= 10){
                debouncingContagem = 0;
                xEventGroupSetBits(xEventos, Tarefa1_flag);
                x=1;
                vTaskDelay(pdMS_TO_TICKS(1000));
                xEventGroupSetBits(xEventos, Tarefa2_flag);
            }
            vTaskDelay(pdMS_TO_TICKS(10));
        }
        else
        {
            debouncingContagem = 0;
        }
        if((digitalRead(BOTAO) == HIGH)){
            x=0;
        }
    }
}
```

Debounce

Para não identificar vários pressionamentos caso o botão seja mantido pressionado

- Exercício 2: Faça um código que tenha 4 tarefas:
  - *Tarefa 1: Seja ativada pela flag “Tarefa1\_flag” e pisque um LED em 1Hz.*
  - *Tarefa 2: Seja ativada pela flag “Tarefa2\_flag” e “Tarefa1\_flag” e pisque um LED em 2Hz.*
  - *Tarefa 3: Seja ativada pela flag “Tarefa3\_flag”, “Tarefa2\_flag” e “Tarefa1\_flag” e imprima “Tarefa 3 ativa”. Após 2 segundos, deve desativar todas as tarefas (limpar as flags).*
  - *Tarefa 4: Faz a leitura de um botão, no momento em que ele é pressionado (nível LOW) pela primeira vez ele ative a flag da tarefa 1, na segunda vez ele ative a flag da tarefa 2 e na terceira ative a flag da tarefa 3.*
- Pode ser interessante consultar o manual:
  - <https://www.freertos.org/event-groups-API.html>



# Event Groups

```
#include "Arduino_FreeRTOS.h"
#include "task.h"
#include "event_groups.h"

#define LED1 7
#define LED2 8
#define BOTAO 2

/*mapeamento de eventos*/
#define Tarefa1_flag (1<<0) //1
#define Tarefa2_flag (1<<1) //10
#define Tarefa3_flag (1<<2) //100

/* Variáveis para armazenamento do handle das tasks */
TaskHandle_t xTarefa1Handle;
TaskHandle_t xTarefa2Handle;
TaskHandle_t xTarefa3Handle;
TaskHandle_t xTarefa4Handle;
EventGroupHandle_t xEventos;

/*protótipos das Tasks*/
void vTarefa1 (void *pvParameters);
void vTarefa2 (void *pvParameters);
void vTarefa3 (void *pvParameters);
void vTarefa4 (void *pvParameters);
```

# Event Groups

```
void setup() {
    Serial.begin(9600);
    pinMode(BOTAO, INPUT_PULLUP);
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
    xEventos = xEventGroupCreate();
    xTaskCreate(vTarefa1, "vTarefa1", 128, NULL, 1, &xTarefa1Handle);
    xTaskCreate(vTarefa2, "vTarefa2", 128, NULL, 1, &xTarefa2Handle);
    xTaskCreate(vTarefa3, "vTarefa3", 128, NULL, 1, &xTarefa3Handle);
    xTaskCreate(vTarefa4, "vTarefa4", 128, NULL, 1, &xTarefa4Handle);
}

void loop() {
    //As funções são executadas nas tarefas
}

void vTarefa1(void *pvParameters){
    while(1)
    {
        xEventGroupWaitBits(xEventos, Tarefa1_flag, pdFALSE, pdTRUE, portMAX_DELAY);
        digitalWrite(LED1, !digitalRead(LED1));
        vTaskDelay(pdMS_TO_TICKS(500));
    }
}
```

Não limpa a flag



# Event Groups

```
void vTarefa2(void *pvParameters){
    while(1)
    {
        xEventGroupWaitBits(xEventos, Tarefa1_flag | Tarefa2_flag, pdFALSE, pdTRUE, portMAX_DELAY);
        digitalWrite(LED2, !digitalRead(LED2));
        vTaskDelay(pdMS_TO_TICKS(250));
    }
}

void vTarefa3(void *pvParameters){
    while(1)
    {
        xEventGroupWaitBits(xEventos, Tarefa1_flag | Tarefa2_flag | Tarefa3_flag, pdFALSE, pdTRUE, portMAX_DELAY);
        Serial.println("Tarefa 3 ativa!");
        vTaskDelay(pdMS_TO_TICKS(2000));
        Serial.println("Desativando as 3 tarefas");
        xEventGroupClearBits(xEventos, 0x07);
    }
}
```

Não limpa a flag

Limpa as 3 flags (que estão nas 3 últimas posições)

```

void vTarefa4(void *pvParameters){
    uint8_t debouncingContagem = 0;
    uint8_t x = 0;
    uint8_t cont = 0;
    while(1)
    {
        if((digitalRead(BOTAO) == LOW) && x==0)
        {
            debouncingContagem++;
            if(debouncingContagem >= 10){
                debouncingContagem = 0;
                x=1;
                cont++;
            }
            vTaskDelay(pdMS_TO_TICKS(15));
        }
        else
        {
            debouncingContagem = 0;
        }
        if((digitalRead(BOTAO) == HIGH)){
            x=0;
        }
        // Serial.println(cont);
        if(cont==1) {xEventGroupSetBits(xEventos, Tarefa1_flag);}
        if(cont==2) {xEventGroupSetBits(xEventos, Tarefa2_flag);}
        if(cont==3) {
            xEventGroupSetBits(xEventos, Tarefa3_flag);
            cont=0;
        }
    }
}

```

- Exercício 3: Refaça o exercício 2, de forma que quando um botão (interrupção – pino 3) seja pressionado, informe na serial os bits setados no grupo de eventos:
  - *Tarefa 1: Seja ativada pela flag “Tarefa1\_flag” e pisque um LED em 1Hz.*
  - *Tarefa 2: Seja ativada pela flag “Tarefa2\_flag” e “Tarefa1\_flag” e pisque um LED em 2Hz.*
  - *Tarefa 3: Seja ativada pela flag “Tarefa3\_flag”, “Tarefa2\_flag” e “Tarefa1\_flag” e imprima “Tarefa 3 ativa”. Após 2 segundos, deve desativar todas as tarefas.*
  - *Tarefa 4: Faz a leitura de um botão (pino 2), no momento em que ele é pressionado (nível LOW) pela primeira vez ele ative a flag da tarefa 1, na segunda vez ele ative a flag da tarefa 2 e na terceira ative a flag da tarefa 3.*
- Pode ser interessante consultar o manual:
  - <https://www.freertos.org/event-groups-API.html>

# Event Groups

```
void botao_ISR(void) {  
    Serial.println("bits setados: " +String(xEventGroupGetBitsFromISR(xEventos)));  
}
```

-> bits setados: 0

-> bits setados: 1

-> bits setados: 3

-> Tarefa 3 ativa!

-> bits setados: 7

Após iniciar a tarefa 1

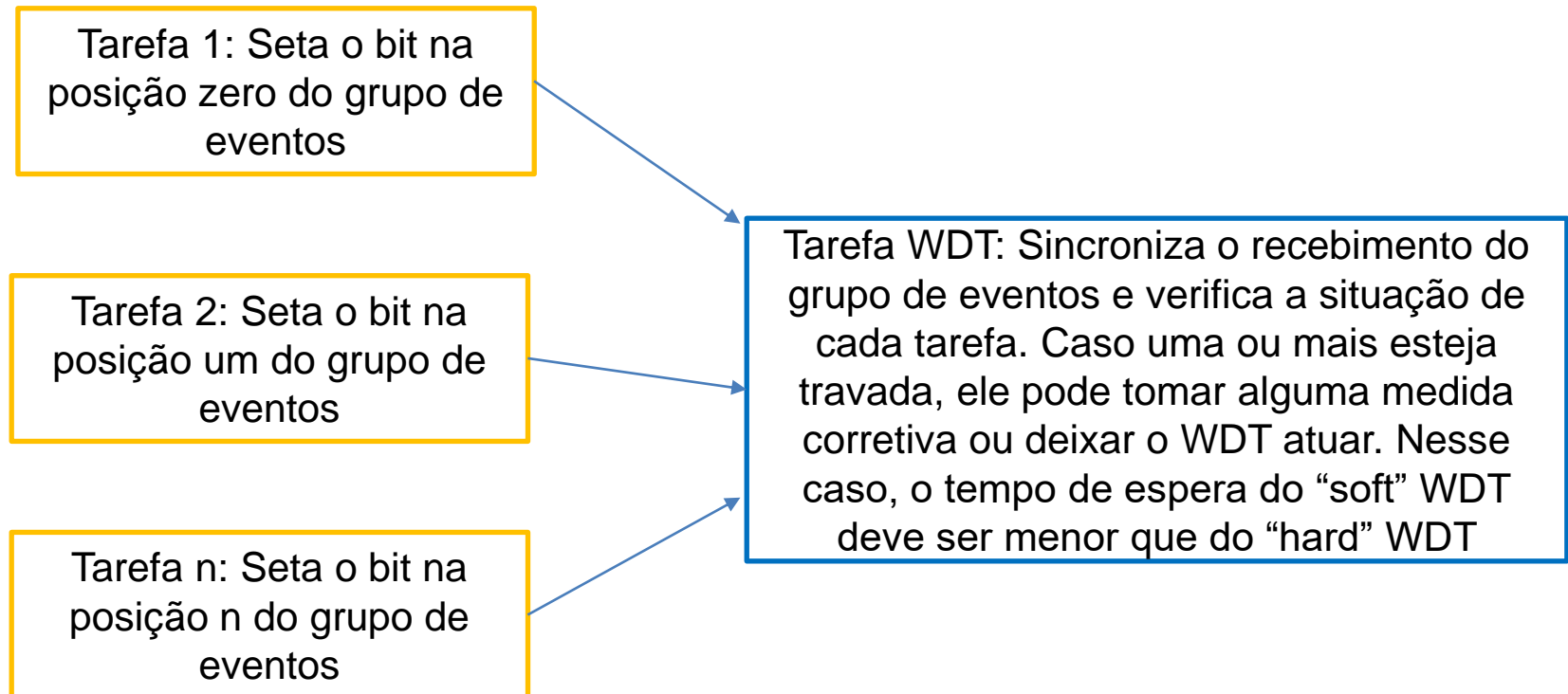
Após iniciar a tarefa 2

Após iniciar a tarefa 3

- Como utilizar o WDT em um sistema multitarefas?
  - *Colocando dentro de uma tarefa?*
    - Apenas ela será monitorada, e ela poderá estourar o WDT caso seja excessivamente preemptada – ou bloqueada devido ao mau compartilhamento de recurso – inversão de prioridade.
  - *Dentro de todas as tarefas?*
    - Não vai funcionar, pois se apenas uma estiver funcionando, irá resetar o WDT mesmo que as outras estejam travadas.
  - *O que fazer?*
    - Utilizar uma tarefa que monitora as outras tarefas. Sincronizar o envio de sinais que sinalizam que a tarefa está funcionando corretamente através de um grupo de eventos.

# WatchDog Timer - Software e Hardware

- Como utilizar o WDT em um sistema multitarefas?





# WatchDog Timer

- Exemplo 2: Crie duas tarefas que pisquem um LED cada com uma frequência de 0,5 Hz. Uma terceira tarefa servirá para realizar o monitoramento.

- Obs: O WDT não funcionará no framework Arduino, pois ele é utilizado para fazer a contagem de tempo –*

*Ticks do sistema.*

```
#include <avr/wdt.h>
#include "Arduino_FreeRTOS.h"
#include "task.h"
#include "event_groups.h"

#define LED1 7
#define LED2 8

/*mapeamento de eventos*/
#define Tarefal_flag (1<<0) //1
#define Tarefa2_flag (1<<1) //10

/* Variáveis para armazenamento do handle das tasks */
TaskHandle_t xTarefalHandle;
TaskHandle_t xTarefa2Handle;
TaskHandle_t xTarefaWDTHandle;
EventGroupHandle_t xEventosWDT;

/*protótipos das Tasks*/
void vTarefal(void *pvParameters);
void vTarefa2(void *pvParameters);
void vTarefaWDT(void *pvParameters);
```

Arquivo no moodle

# WatchDog Timer

```
void setup() {
    wdt_enable(WDTO_2S);
    Serial.begin(9600);
    xEventosWDT = xEventGroupCreate();
    xTaskCreate(vTarefal, "Tarefal", 128, NULL, 1, &xTarefalHandle);
    xTaskCreate(vTarefa2, "Tarefa2", 128, NULL, 1, &xTarefa2Handle);
    xTaskCreate(vTarefaWDT, "vTarefaWDT", 256, NULL, 2, &xTarefaWDTHandle);
    Serial.println("Iniciando o sistema...");
    //vTaskStartScheduler();
}

void loop() {
    //As funções são executadas nas tarefas
}

void vTarefal(void *pvParameters){
    pinMode(LED1, OUTPUT);
    while(1) {
        //Tarefa 1 seta o bit para avisar que está ok!
        xEventGroupSetBits(xEventosWDT, Tarefal_flag);
        digitalWrite(LED1, !digitalRead(LED1));
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

void vTarefa2(void *pvParameters){
    pinMode(LED2, OUTPUT);
    while(1) {
        //Tarefa 2 seta o bit para avisar que está ok!
        xEventGroupSetBits(xEventosWDT, Tarefa2_flag);
        digitalWrite(LED2, !digitalRead(LED2));
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

Teste comentando essa linha

# WatchDog Timer

```
void vTarefaWDT(void *pvParameters){
    EventBits_t uxBitsTarefas;
    TickType_t xUltimaLeituraTempo;
    // A variável xUltimaLeituraTempo deve ser inicializada com o primeiro tempo a ser utilizado
    // após a primeira execução, ela é atualizada pela função vTaskDelayUntil
    xUltimaLeituraTempo = xTaskGetTickCount();
    for(;;){
        // vTaskDelayUntil serve para sincronizar a leitura sempre a cada 1000ms
        // Essa função é útil para manter uma tarefa com execução periódica
        vTaskDelayUntil(&xUltimaLeituraTempo, pdMS_TO_TICKS(1000));
        // 0x03 = monitora os dois ultimos bits
        uxBitsTarefas = xEventGroupWaitBits(xEventosWDT, 0x03, pdTRUE, pdTRUE, 0);
        if (( uxBitsTarefas & ( 0x03 )) == ( 0x03 )){
            // Tarefas ok!
            Serial.println("Tarefas ok!");
        }else if(( uxBitsTarefas & 1 << 0) != 0){
            Serial.println("Tarefas 2 não retornou");
            // Apenas a tarefa 1 retornou ok
            // Faça aqui uma tentativa de reparo da tarefa 2
        }else if(( uxBitsTarefas & 1 << 1) != 0){
            Serial.println("Tarefas 1 não retornou");
            // Apenas a tarefa 2 retornou ok
            // Faça aqui uma tentativa de reparo da tarefa 1
        }else{
            // Nenhuma tarefa retornou, aguarda o WDT atuar
            Serial.println("Nenhuma tarefa retornou, aguardando WDT atuar....");
            for(;;);
        }
        // Reseta o WDT.
        // Nesse caso o WDT só iria atuar se as duas tarefas travassem
        wdt_reset();
    }
}
```

A variável `UltimaLeituraTempo` é atualizada automaticamente pela função `vTaskDelayUntil`

Aguarda por todos os bits

Apagar o bits após a leitura

`vTaskDelete(...);`  
`xTaskCreate(...);`

# WatchDog Timer

## Exemplo 3: Simule um WDT com a função resetFunc();

```
void(* resetFunc) (void) = 0;
```

Declare a função

```
void vTarefaWDT(void *pvParameters){
    EventBits_t uxBitsTarefas;
    TickType_t xUltimaLeituraTempo;
    // A variável xUltimaLeituraTempo deve ser inicializada com o primeiro tempo a ser utilizado
    // após a primeira execução, ela é atualizada pela função vTaskDelayUntil
    xUltimaLeituraTempo = xTaskGetTickCount();
    for(;;){
        // vTaskDelayUntil serve para sincronizar a leitura sempre a cada 1000ms
        // Essa função é útil para manter uma tarefa com execução periódica
        vTaskDelayUntil(&xUltimaLeituraTempo, pdMS_TO_TICKS(1000));
        // 0x03 = monitora os dois ultimos bits
        uxBitsTarefas = xEventGroupWaitBits(xEventosWDT, 0x03, pdTRUE, pdTRUE, 0);
        if (( uxBitsTarefas & ( 0x03 )) == ( 0x03 )){
            // Tarefas ok!
            Serial.println("Tarefas ok!");
        }else if(( uxBitsTarefas & 1 << 0) != 0){
            Serial.println("Tarefas 2 não retornou");
            // Apenas a tarefa 1 retornou ok
            // Faça aqui uma tentativa de reparo da tarefa 2
        }else if(( uxBitsTarefas & 1 << 1) != 0){
            Serial.println("Tarefas 1 não retornou");
            // Apenas a tarefa 2 retornou ok
            // Faça aqui uma tentativa de reparo da tarefa 1
        }else{
            // Nenhuma tarefa retornou, aguarda o WDT atuar
            Serial.println("Nenhuma tarefa retornou, aguardando WDT atuar....");
            for(;;){
                vTaskDelay(pdMS_TO_TICKS(100));
                resetFunc();
            }
        }
        // Reseta o WDT.
        // Nesse caso o WDT só iria atuar se as duas tarefas travassem
        wdt_reset();
    }
}
```

Teste comentando o xEventGroupSetBits da tarefa 1 e 2

Função utilizada para reiniciar o microcontrolador. Não funciona como um WDT real, pois se a tarefa de monitoramento travar (ou o uC travar totalmente), ela não será acionada.

- FREERTOS – Task Notifications

# Referências

- <https://www.freertos.org/>
- [https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS\\_Reference\\_Manual\\_V10.0.0.pdf](https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf)
- [https://www.freertos.org/fr-content-src/uploads/2018/07/161204\\_Mastering\\_the\\_FreeRTOS\\_Real\\_Time\\_Kernel-A\\_Hands-On\\_Tutorial\\_Guide.pdf](https://www.freertos.org/fr-content-src/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf)
- <https://www.embarcados.com.br/>