
Programmation en C des microcontrôleurs, Exercices Sujet0quater__c__micro

Licence EPOCS *(ECUE23)

Pierre Pardo

2020-2021

Ce sujet contient 2 exercice(s).



Comment répondre ?

Ce sujet comporte des exercices, les réponses seront données sous formes de fichier sources (*.c et/ou *.h) ou de fichier texte (*.txt, *.odt) quand il y a des réponses associées soit:

- Sur un support numérique (clef USB) à la fin du cours ou du TP.
- Par email à pierre.pardo@univ-amu.fr, au maximum le lendemain du cours ou du TP.

Il y aura souvent plusieurs fichiers aux noms identiques, il faudra transférer les arborescences complètes des fichiers. Le plus simple est de compresser les dossiers.

*Licence professionnelle Électronique Pour Objets Connectés et Smart-Grids

Nous allons analyser le fonctionnement des `EventGroup_t`.

- (a) — Importer le projet `demoHommeMort_master` dans STM32CubeIDE
 - Renommer le immédiatement en `demoHommeMort_master_Event`.
- (b) A partir de ce projet, ajouter les initialisations suivantes (dans le `mainc.`) :
 - Créer un `EventGroupHandle_t` `MesEvents`
 Et ajouter les tâches suivantes :
 - `TacheGen1`
 - Une tâche qui attend 500 ms
 - Active le `Bit0` du `MesEvents`
 - Recommence
 - `TacheGen2`
 - Une tâche qui attend 750 ms
 - Active le `Bit1` du `MesEvents`
 - Et recommence
 - `TacheGen3`
 - Une tâche qui attend 950 ms
 - Active le `Bit2` du `MesEvents`
 - Et recommence
 - `TacheWaitBits`
 - Cette tâche attend au maximum 100 ms qu'un `Bit` soit présent dans `MesEvents` et ne les remet pas à zéro.
 - Affiche les bits de `MesEvents` qui sont présents (éventuellement)
 - Et recommence
 - `TacheWaitResetBits`
 - Cette tâche attend (durée infinie) que tous les bits soient présents dans `MesEvents` et les remet à zéro quand cela est vrai.
 - Affiche un message
 - Et recommence l'attente

API des EventGroup

```
EventGroupHandle_t xEventGroupCreate( void );
EventBits_t xEventGroupWaitBits( const EventGroupHandle_t xEventGroup, const
    ↳ EventBits_t uxBitsToWaitFor, const BaseType_t xClearOnExit, const
    ↳ BaseType_t xWaitForAllBits, TickType_t xTicksToWait );
EventBits_t xEventGroupSetBits( EventGroupHandle_t xEventGroup, const
    ↳ EventBits_t uxBitsToSet );
```

- (c) Les tâches `TacheGen1` à `TacheGen3` sont similaires, vous pouvez utiliser la même tâche (et passer un paramètre).

Nous allons utiliser un timer logiciel pour simuler un mécanisme de l'homme mort.

- (a) Vous pouvez ajouter directement les lignes de gestion de l'homme mort dans le programme **Gatekeeper**. **Ne pas utiliser le projet demoHommeMort_master.**

- Importer le projet **Gatekeeper** dans STM32CubeIDE
- Renommer le immédiatement en **Gatekeeper_Homme_Mort**.

- (b) Créer un timer logiciel, les commandes nécessaires sont ci-dessous.

En global

```
TimerHandle_t xAlert = NULL;
```

Dans la configuration :

```
//...
xAlert = xTimerCreate("Alert", pdMS_TO_TICKS(10000), pdFALSE, 0,
    ↪ vAlertTimerCallback);
xTimerStart (xAlert, 0);
//... tester les valeurs de retour...
```

Ajouter dans l'interruption le redémarrage du timer :

```
xTimerStartFromISR( xAlert, &xHigherPriorityTaskWoken);
portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
```

Et ajouter la fonction de gestion du timer, vous pouvez mettre un affichage à l'intérieur (utiliser le **Gatekeeper**) :

```
void vAlertTimerCallback( TimerHandle_t pxTimer )
{
    // déclenchement de l'alerte !
}
```

Compiler et transférer ce programme sur la carte Nucleo.

Lancer un terminal (**Putty** ou **Termite**) (115200,8,N,1 sans contrôle).

- Les leds doivent clignoter
- Des messages doivent apparaître régulièrement (tâche **Alive**)
- Un appui sur le bouton bleu doit changer le mode de clignotement, et afficher un message
- Un message d'alerte doit apparaître au bout de 10 secondes si il n'y a pas d'appui

Les messages renvoyés par la carte sont sur des canaux différents (identifiés par des caractères en début de ligne) :

- (c) Modifier le paramètre **Autoreload** de la création du timer pour avoir des alertes qui se répètent.