FREE-RTOS

I. Création d'une tâche pour gérer l'ADC

L'idée de ce code est de créer une tâche permettant de lancer de manière récurrente un ordre de conversion pour l'ADC1.

Dans un premier temps, il faut lancer CubeMx et définir un nouveau projet (choisisse votre cible). Une fois fait, il faut choisir FREERTOS dans l'onglet MiddleWares de CubeMX. Vous devez aussi choisir une entrée sur laquelle vous pourrez mettre votre signal analogique à convertir.

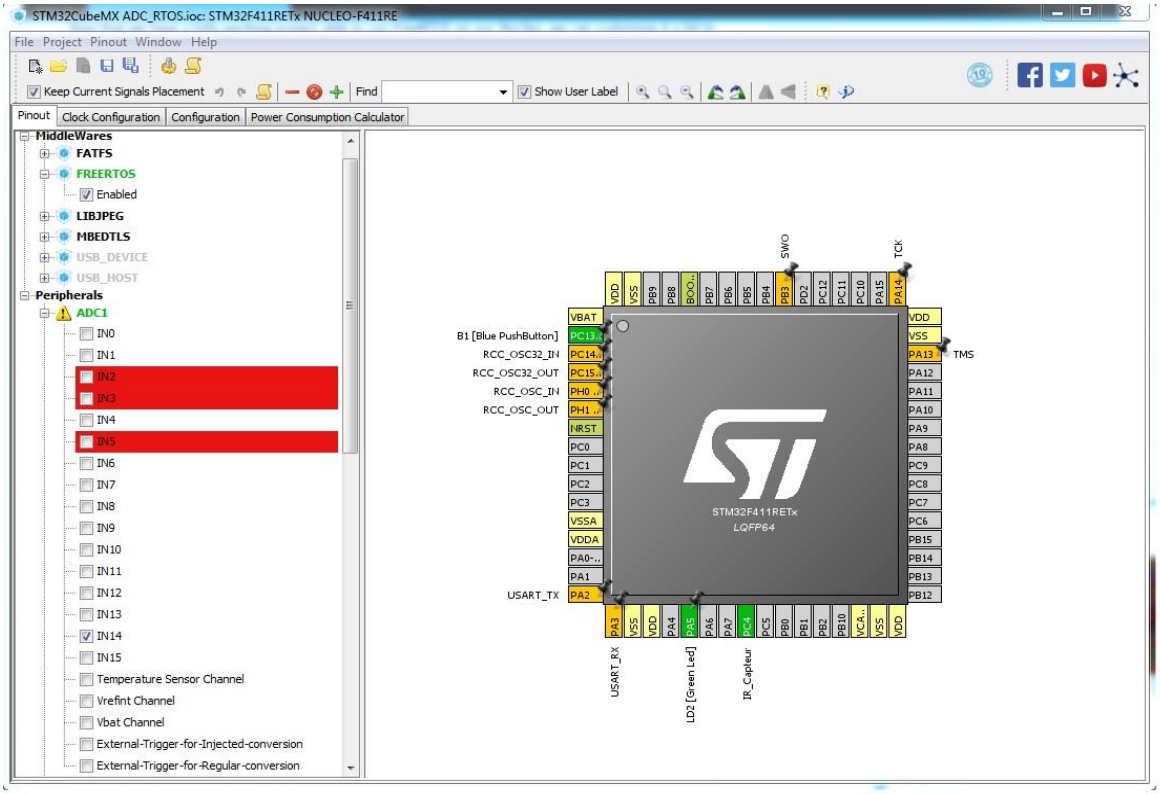

FIGURE 1: SELECTION DE FREERTOS ET DE L'ADC

Ensuite passer à l'onglet configuration.

Cliquez sur FreeRTOS pour ouvrir la fenêtre de configuration de l'OS.

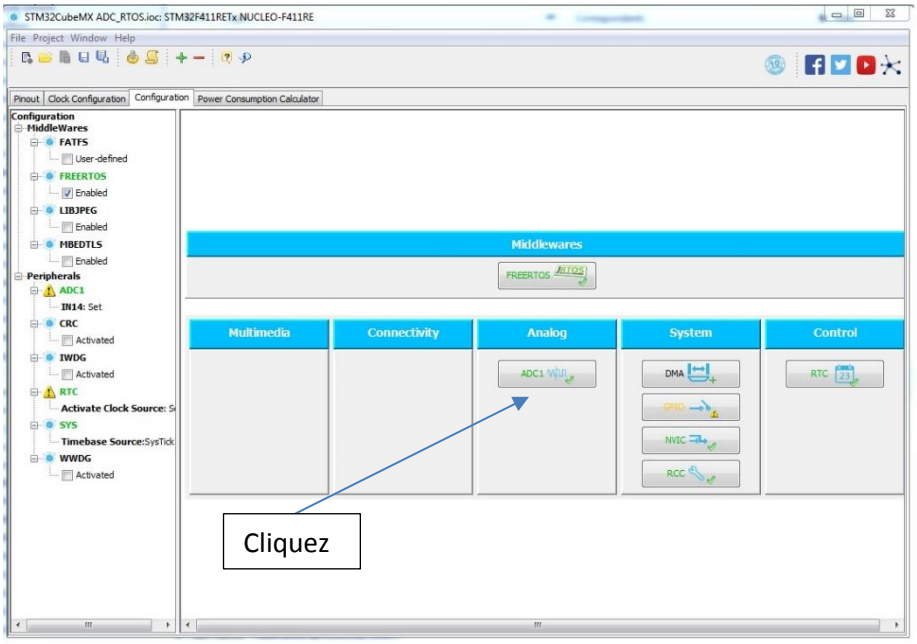

FIGURE 2: CONFIGURATION FREERTOS

Ensuite cliquez sur Tasks and Queues pour créer votre tâche.

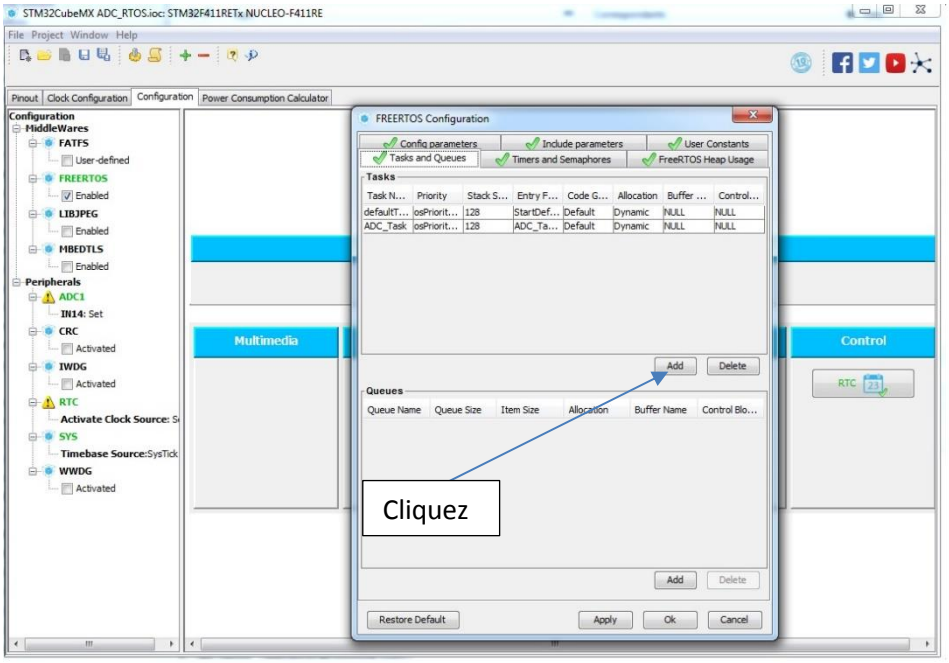

FIGURE 3: AJOUT D'UNE TACHE

Il faut maintenant créer votre propre tâche. Ici je L'appelle ADC_Task et le nom de la fonction associée est ADC_Task_Code.

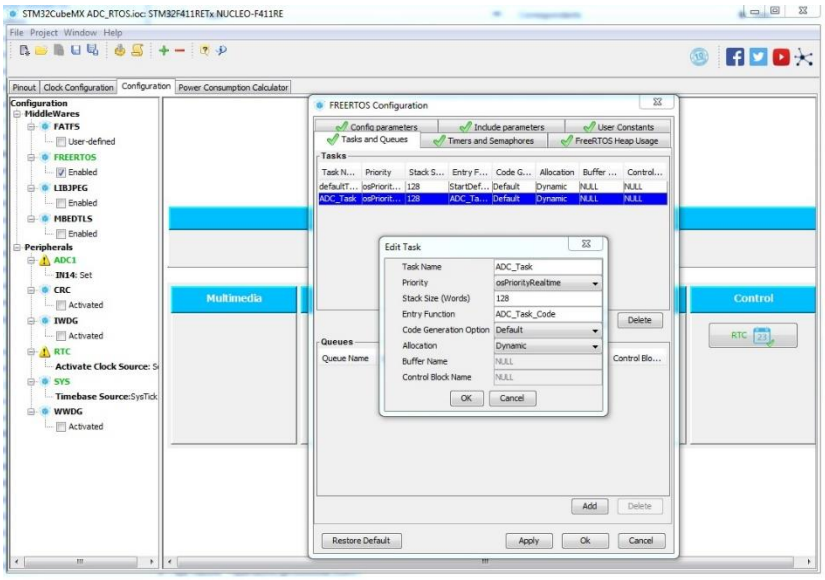

FIGURE 4: DEFINITION DE LA TACHE

Une fois la configuration de votre tâche réalisée, vous devez réaliser les configurations classiques de votre composant. Dans notre cas configurer l'ADC ainsi que le NVIC pour les interruptions.

Dès que ces tâches sont terminées, vous pouvez demander à CubeMx de générer le projet et son code associé.

L'idée de l'applicatif est de maintenant développer le code permettant de généré le clignotement de la LED (LED_2) de la carte lorsque l'interruption de l'ADC est générée. De fait à la première interruption générée, la LED sera allumée puis à la génération de l'IT suivante elle sera éteinte. Pour se faire nous utiliserons une variable globale d'état.

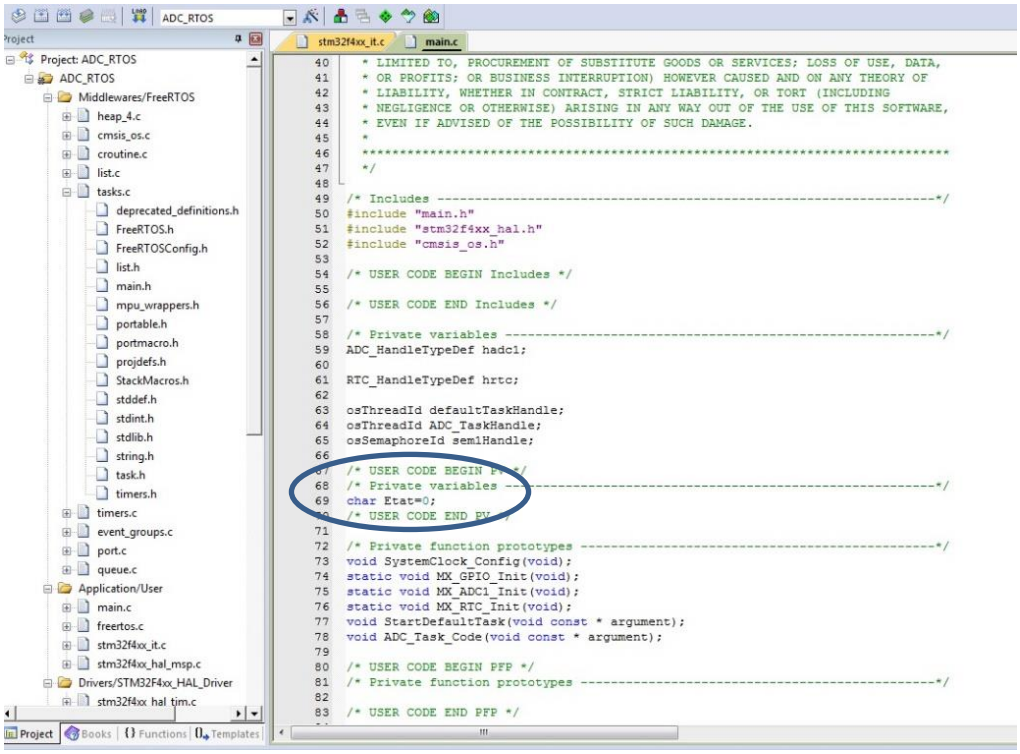

FIGURE 5: DECLARATION DE LA VARIABLE D'ETAT

Il faut maintenant écrire le code correspondant à la tâche que nous avons déclarée à savoir la tâche ADC_Task. La fonction exécutée est la fonction ADC_Task_Code. Dans cette fonction, l'idée est de seulement lancer la conversion de l'ADC en mode interruption. Pour laisser suffisamment de temps entre deux interruptions, nous rajoutons un temps d'attente gérer par la fonction osDelay().

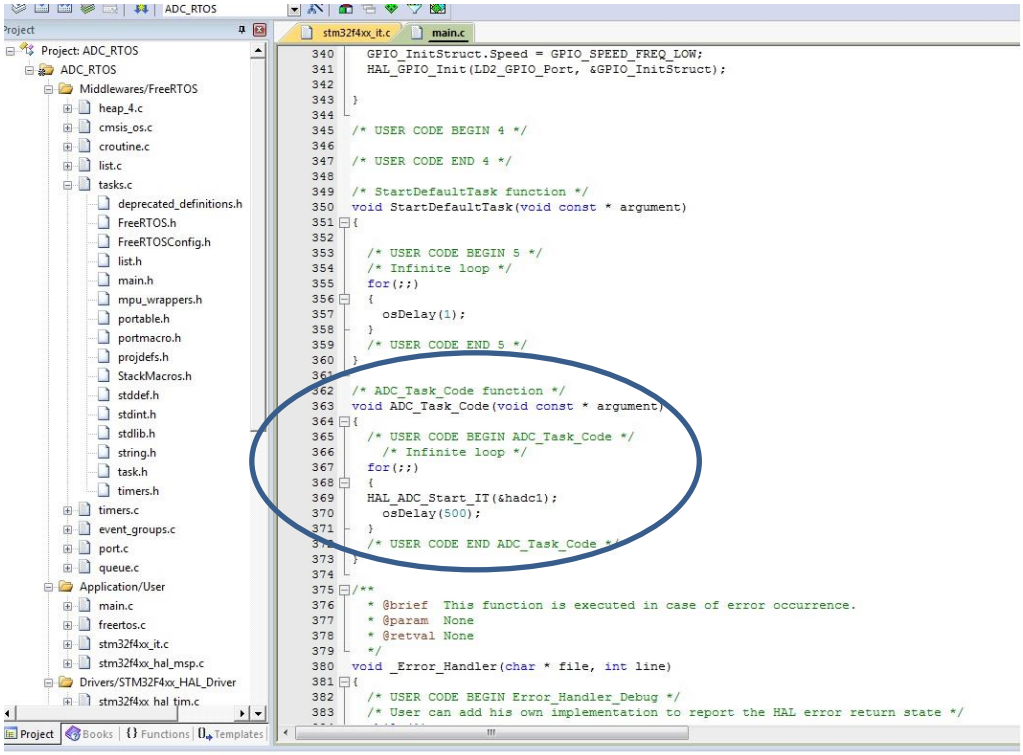

FIGURE 6: CODE DE LA FONCTION ADC_TASK_CODE

Reste maintenant à écrire le code de la fonction d'interruption de l'ADC. Dans cette fonction, nous devons seulement effacer le flag de l'interruption (normalement généré automatiquement par CubeMx) sinon l'interruption bouclera sur elle-même. Ensuite en fonction de la valeur de la variable Etat la led doit s'allumer ou s'éteindre.

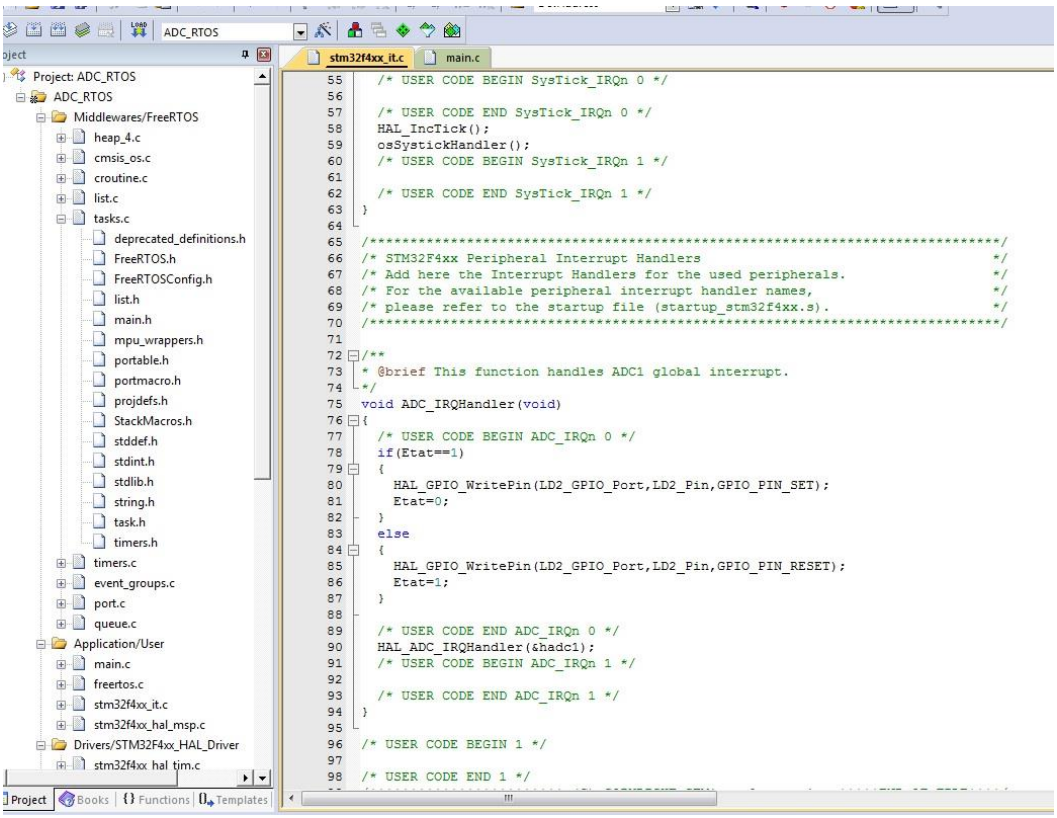

FIGURE 7: CODE DE LA FONCTION D'INTERRUPTION

Une fois le code compilé et téléchargé sur la carte, vous devez voir s'allumer et s'éteindre la led de la carte.

Le second code que nous allons écrire visera à utiliser un autre mécanisme important des OS à savoir les sémaphores. Ce code va donc créer deux tâches ; la première attendra que l'utilisateur appuie sur le bouton User de la carte (bouton bleu) dès lors cette tâche réveillera la seconde bloquée par un sémaphore en lâchant celui-ci. La seconde tâche fera clignoter la LED comme dans le code précédent.

La première chose à faire est de retourner sur CubeMX afin de créer le sémaphore (nous créerons un sémaphore binaire). Pour cela il suffit de cliquer sur « add » ensuite il faut donner un nom au sémaphore (cf. Figure 8) dans l'exemple je l'appelle « sem1 ». Il faut également créer une nouvelle tâche en suivant la même procédure que précédemment. Ici je créé la tâche LED_Task avec une priorité normale et avec comme nom de fonction LED_Task_Code.

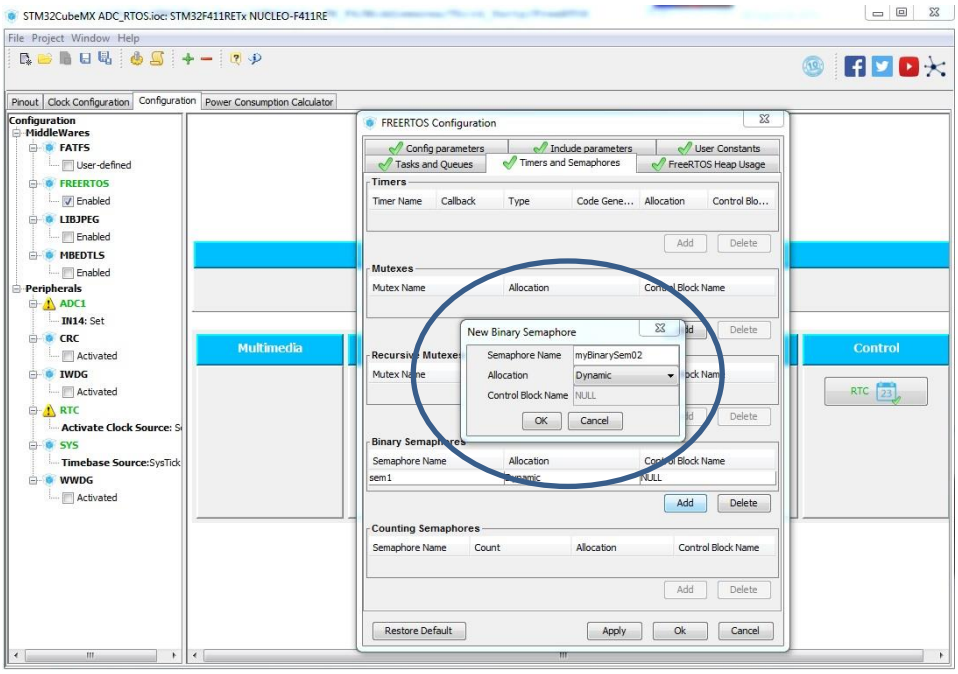

FIGURE 8: CREATION D'UN SEMAPHORE BINAIRE

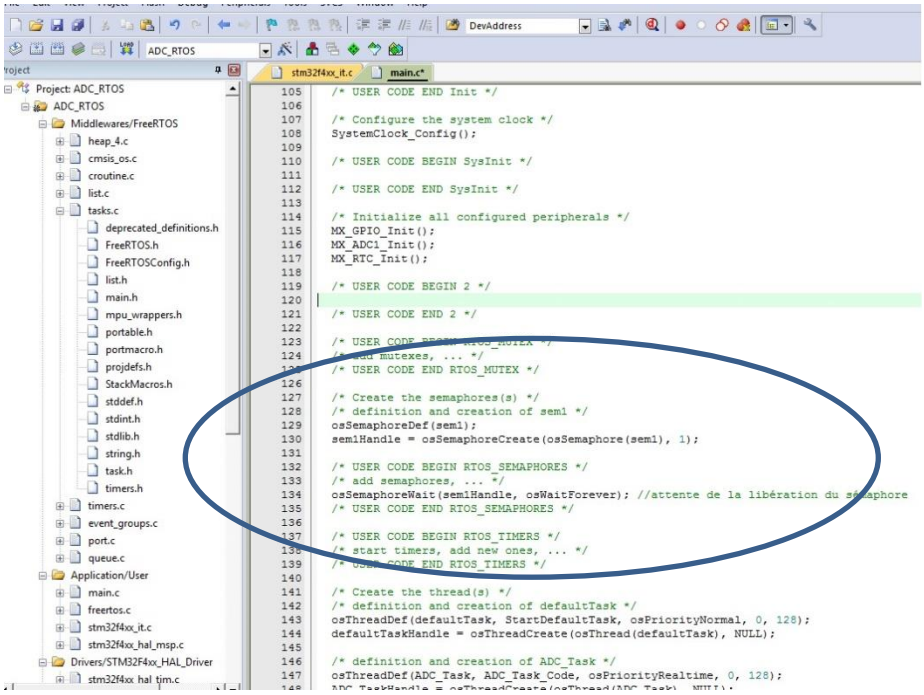

FIGURE 9: CODE DE GENERATION DU SEMAPHORE

Une fois le code généré, nous voyons bien la création du sémaphore dans le fichier main.c (cf. Figure 9). Il faut maintenant écrire le code de la fonction appelée par la tâche DefaultTask. Le but de cette tâche est de bloquer la tâche gérant la LED jusqu'à ce que l'utilisateur appuie sur le bouton USER (bouton bleu de la carte). Le code réalisant ces fonctions est présenté sur la figure 10.

```
0x0800147C 2000      MOVS        r0,#0x00
0x0800147E F000F99C  BL.W        osThreadTerminate (0x080017BA)
   366:     osDelay(1);
0x08001482 2001      MOVS        r0,#0x01
```

```
stm32f4xx_it.c | main.c* | startup_stm32f411xe.s | tasks.c

345   HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
346
347 }
348
349 /* USER CODE BEGIN 4 */
350
351 /* USER CODE END 4 */
352
353 /* StartDefaultTask function */
354 void StartDefaultTask(void const * argument)
355 {
356
357   /* USER CODE BEGIN 5 */
358   /* Infinite loop */
359   for(;;)
360   {
361     if(!HAL_GPIO_ReadPin(GPIOC, B1_Pin)) //attend l'appui sur le bouton USER
362     {
363       osSemaphoreRelease(sem1Handle); //relache le sémaphore pour bloque la tâche gérant la LED
364       osThreadTerminate(NULL); //termine la tâche
365     }
366     osDelay(1);
367   }
368   /* USER CODE END 5 */
369 }
370
371 /* ADC_Task_Code function */
372 void ADC_Task_Code(void const * argument)
373 {
374   /* USER CODE BEGIN ADC_Task_Code */
375     /* Infinite loop */
376   for(;;)
377   {
```

FIGURE 10: CODE DE LA FONCTION D'ATTENTE

Il ne reste plus qu'à écrire le code gérant le clignotement de la LED. Dans ce code, il faut seulement attendre que le sémaphore soit libéré et ensuite générer le clignotement à une fréquence d'une seconde. Ce code est présenté sur la figure 11.

```
364:                    osThreadTerminate(NULL);
365:                }
0x0800147C 2000      MOVS        r0,#0x00
0x0800147E F000F99C  BL.W        osThreadTerminate (0x080017BA)
   366:     osDelay(1);
0x08001482 2001      MOVS        r0,#0x01
```

```
stm32f4xx_it.c | main.c* | startup_stm32f411xe.s | tasks.c

369 }
370
371 /* ADC_Task_Code function */
372 void ADC_Task_Code(void const * argument)
373 {
374   /* USER CODE BEGIN ADC_Task_Code */
375     /* Infinite loop */
376   for(;;)
377   {
378   HAL_ADC_Start_IT(&hadc1);
379     osDelay(500);
380   }
381   /* USER CODE END ADC_Task_Code */
382 }
383
384 /* LED_Task_Code function */
385 void LED_Task_Code(void const * argument)
386 {
387   /* USER CODE BEGIN LED_Task_Code */
388   /* Infinite loop */
389   if(osSemaphoreWait(sem1Handle, osWaitForever) ==osOK)
390   {
391     for(;;)
392     {
393       HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
394       osDelay(500);
395     }
396   }
397   /* USER CODE END LED_Task_Code */
398 }
399
400 /**
```

FIGURE 11: CODE GERANT LE CLIGNOTEMENT DE LA LED