TP5

Gestion différée des interruptions sur freertos

1. Objectif :

Ce TP a comme objectifs :

- montrer comment on peut exploiter le mécanisme de sémaphore pour différer le traitement des interruptions.
- Montrer la différence entre l'utilisation d'un sémaphore binaire et d'un sémaphore à compteur interne.
- Explorer la technique utilisée par freertos pour forcer la commutation de contexte.

2. Mode opératoire :

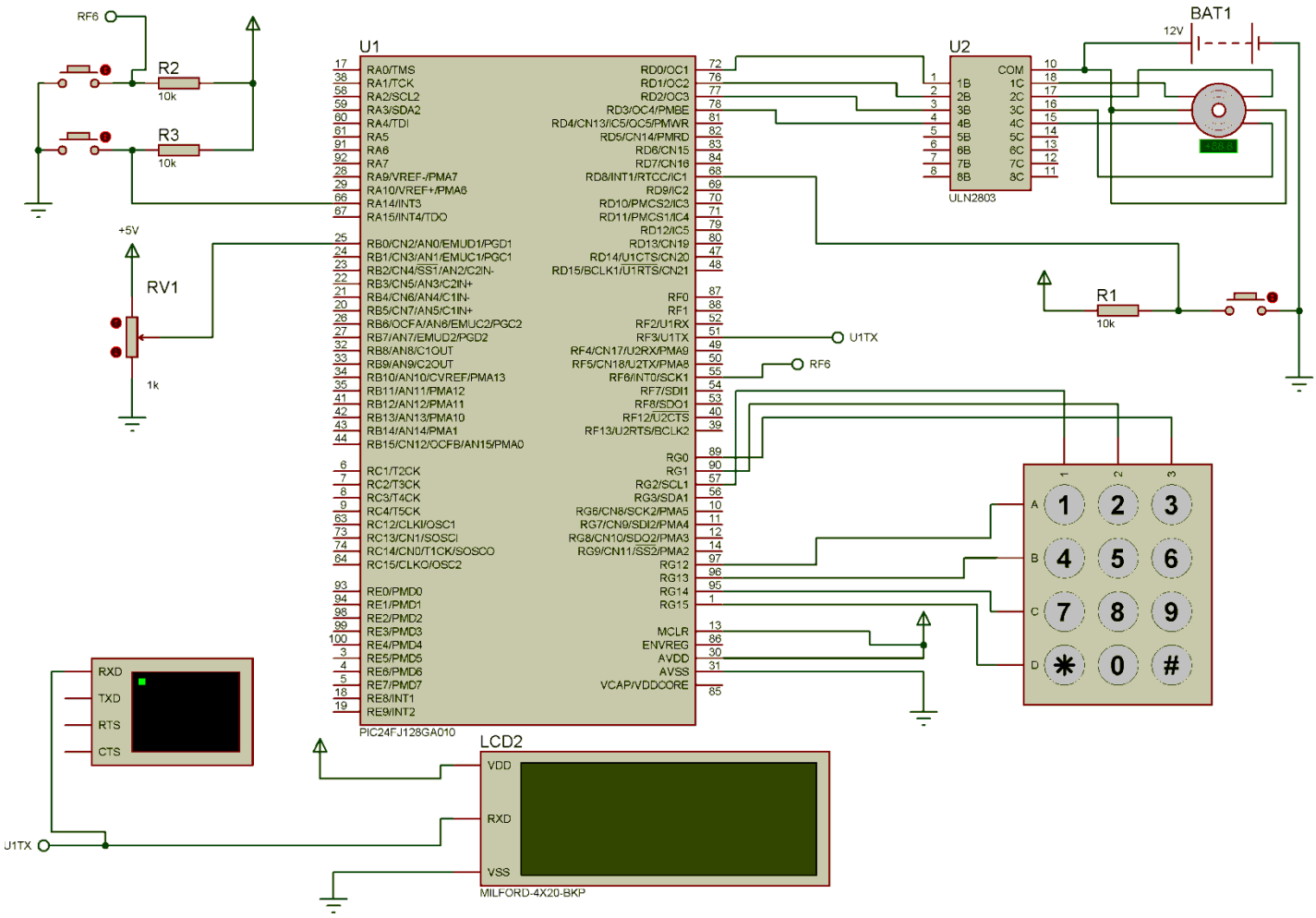
2.1. Outils de travail :

Dans ce TP vous aurez besoin, en plus de freertos que vous avez déjà téléchargé pour les premiers TP de :

- un environnement de développement intégré IDE(Integreted Developpement Environnement), dans ce TP on utilisera le dernier IDE de Microchip Mplab X v1.90.
- Un compliateur pour les pic 16 bits, on utilisera le compilateur C30 ou X16 fourni par Microchip.
- Isis proteus (si vous avez la licence !) ou une carte didactique avec un pic 24F comme par exemple **The Explorer 16 Development Board** conçue par microchip.
- Datasheet du microcôntroleur PIC24FJ128GA010 (à télécharger).

2.2. Projet initial et Réalisation sur ISIS de la plateforme de travail :

Nous allons utiliser le même design que celui du TP4 avec une légère modification :



* ULN2803 est un driver pour le moteur pas à pas (voir datasheet).

le projet initial s'inspire de l'exercice de synthèse du TP2 où chaque touche de 0 à 3 permet de lancer ou arrêter le basculement d'une sortie du portA avec des fréquences différentes et rapport cycliques (0 pour RA0, 1 pour RA1, …).

2.3. Ajout de la gestion différée de l'interruption externe INT1 en utilisant un sémaphore binaire:

Cette partie de TP reprend l'exemple étudié en séance de cours qui permet de faire avancer ou reculer (alternativement) un moteur pas à pas à 4 phases de 4 pas, à chaque fois qu'on appuie sur un bouton poussoir relié à la broche INT1 (RD8), en utilisant le mode interruptible.

1. Ajoutez au projet le fichier semphr.h à partir du code source freertos, pour pouvoir utiliser les semaphores.
2. Ajoutez au projet un nouveau fichier que vous intitulez stepmotor.c.
3. Ajoutez à ce fichier le code suivant qui implémente :
   - le sous-programme d'interruption (ISR) pour l'interruption INT1 et qui permet de donner (to give) un sémaphore pour débloquer la tâche de traitement de cet événement externe vTaskINT1Handler().

- la tâche vTaskINT1Handler( ) qui en boucle attend l'apparition d'un sémaphore pour passer de l'état bloqué à l'état prêt et par la suite quand elle est passée par le scheduler à l'état en exécution (Running), elle fait avancer ou reculer le moteur pas à pas de 4 pas.

```c
#include <xc.h>
#include "FreeRTOS.h"
#include "semphr.h"
#include "task.h"


//le semaphores pour differer le traitement de l'interruption INT1,
xSemaphoreHandle xSemaphore1;
#define DelaiPas 500    //500ticks soit 500ms si le tick vaut 1 ms

void _ISR _INT1Interrupt(void) {
    signed portBASE_TYPE pxHigherPriorityTaskWoken = pdFALSE;
    _INT1IF = 0; //effacer le flag
    xSemaphoreGiveFromISR(xSemaphore1, &pxHigherPriorityTaskWoken); //inserer un jeton dans le semaphore1
    if (pxHigherPriorityTaskWoken == pdTRUE) taskYIELD();
    //si cette insertion debloque une tache plus prioritaire que la tache qui etait interrompue
    // faire une commutation de contexte
}
```

```c
void vTaskINT1Handler(void *p) {
    unsigned int cmdword; //mot de commande du moteur pas a  pas
    unsigned char direction = 'f'; //direction de rotation
    TRISD = 0xFFF0; //RD3 a  RD0 sorties les autres entrÃ©es
    PORTD = 0x0001; //init PORTD a  1 : phase 1 alimentee du moteur pas Ã  pas
    vSemaphoreCreateBinary(xSemaphore1); //creation d'un semaphore binaire
    xSemaphoreTake(xSemaphore1, portMAX_DELAY); //retirer le jeton cree par defaut
    _INT1EP = 1; //interruption sur front positif
    _INT1IP = 1; //priorite 1 la meme que celle du kernel de freertos
    _INT1IF = 0; //effacer le flag
    _INT1IE = 1; //activer l'interrupt INT1
```

```c
    while (1) {
        xSemaphoreTake(xSemaphore1, portMAX_DELAY); //blocage jusqu'a  la reception d'un jeton
        _INT1IE = 0; //desactiver les interruptions sur INT1
        if (direction == 'f') {
            cmdword = 0x1;
            while (cmdword < 8) {
                cmdword = cmdword << 1;
                PORTD = (PORTD & 0xFFF0) | cmdword;
                vTaskDelay(DelaiPas);
            }
            cmdword = 0x01;
            PORTD = (PORTD & 0xFFF0) | cmdword;
            vTaskDelay(DelaiPas);
            direction = 'b';
        } else {
            cmdword = 0x08;
            PORTD = (PORTD & 0xFFF0) | cmdword;
            vTaskDelay(DelaiPas);
            while (cmdword > 1) {
                cmdword = cmdword >> 1;
                PORTD = (PORTD & 0xFFF0) | cmdword;
                vTaskDelay(DelaiPas);
            }
            direction = 'f';
        }
        //_INT1IF=0;
        _INT1IE = 1; //reactiver les interruptions

    }
}
```

4. dans le programme principal main( ) ajoutez la création de la tâche vTaskINT1Handler( ) avec une priorité égale à celle des autres tâches :

```
xTaskCreate(vTaskINT1Handler, "INT1H", 200, NULL, 1, NULL);
```

5. Clean and build votre projet ; si aucune erreur n'est signalée vous pouvez passer à l'étape de test sur ISIS.
6. Testez le bon fonctionnement du projet :
   - une impulsion sur un bouton poussoir, doit provoquer une séquence {1,2,4,8,1} ou {1,8,4,2,1} sur les pins RD3→RD0.
   - Une autre impulsion sur le bouton poussoir alors que le moteur est en train de tourner sera traitée à la fin de ce déplacement et provoquera un déplacement dans le sens contraire.
   - Si on effectue plusieurs impulsions alors que le moteur est en train de tourner une seule sera traitée les autres seront ignorées (le sémaphore est binaire : au plus un).

NB : si vous n'arrivez pas à générer plusieurs impulsions alors que le moteur n'a pas encore terminé les 4 pas vous pouvez ralentir sa vitesse en augmentant le délai d'attente sur chaque pas : au lieu de 500ms vous pouvez prendre 1000ms=1s.

7. Pour ignorer les impulsions qui apparaissent alors que le moteur tourne, ajoutez l'instruction INT1F=0 juste avant l'instruction INT1E=1 à la fin de la boucle while(1) dans vTaskINT1Handler() : avant donc de réactivez l'interruption INT1 on efface le flag INT1F qui a avait été positionné par les impulsions précédentes. Refaites le test pour vérifier que toutes les impulsions survenues alors que le moteur était en train de tourner sont ignorées.

8. Supprimez l'instruction ajoutée dans la question 6 (INT1F=0). Le comportement précédent (une seule impulsion est prise en compte alors que le moteur tourne), reste valable même si on ne désactive pas l'interruption INT1 dans la tâche handler : mettez les instructions INT1E=0 et INT1E=1 dans la boucle while(1) en commentaire et refaites le test ; dans ce cas chaque impulsion qui apparait alors que le moteur est commandé déclenche une interruption _INT1Interrupt() qui interrompe vTaskINT1Handler() la tâche handler (ou toute autre tâche), et qui ajoute un jeton dans le semaphore. Mais quel que soit le nombre d'interruption déclenché au plus un jeton sera dans le semaphore (binaire).
9. Tout en gardant les deux instructions INT1E=0 et INT1E=1 en commentaires, ajoutez INT1F=0 à la fin de la boucle while(1) dans vTaskINT1Handler(). Testez à nouveau combien d'impulsions sont prises en compte alors que le moteur est en train de tourner ? Expliquez ce comportement …

2.4. Ajout de la gestion différée de l'interruption externe INT1 en utilisant un sémaphore à compteur interne:

On souhaite maintenant prendre en considération plusieurs impulsions qui sont effectuées alors que le moteur est déjà en mouvement, pour cela on utilisera un sémaphore à compteur binaire.

10. Dans le code précédent de vTaskINT1Handler (), remplacez la création du sémaphore binaire :

```
vSemaphoreCreateBinary(xSemaphore); // Create the semaphore
xSemaphoreTake(xSemaphore, portMAX_DELAY); // to ignore the initial semaphore
```

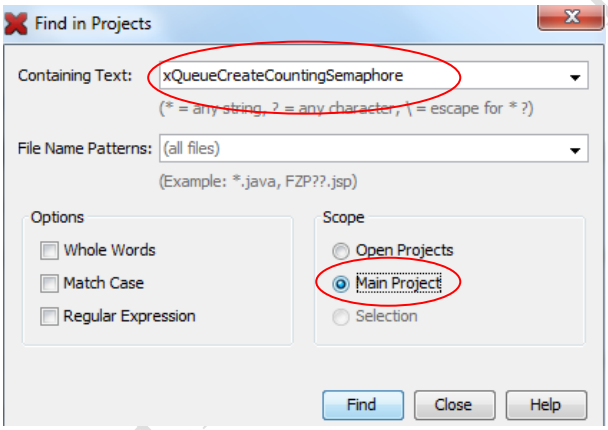
par le code créant un sémaphore à compteur interne avec 0 jetons au début et pouvant contenir jusqu'à 3 jetons.

```
xSemaphore = xSemaphoreCreateCounting(3,0); // Create the semaphore
```

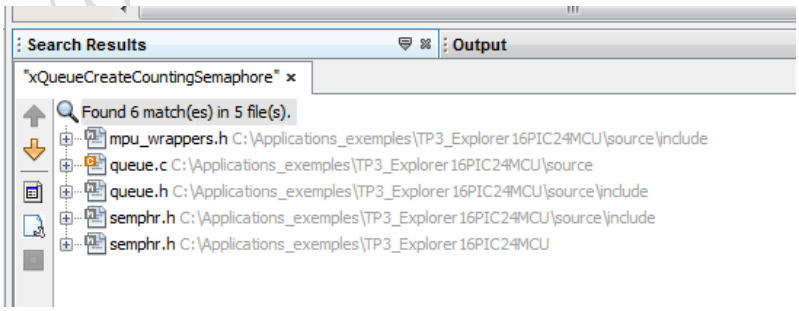
11. 'Clean and build' votre projet, vous obtenez l'erreur suivante :

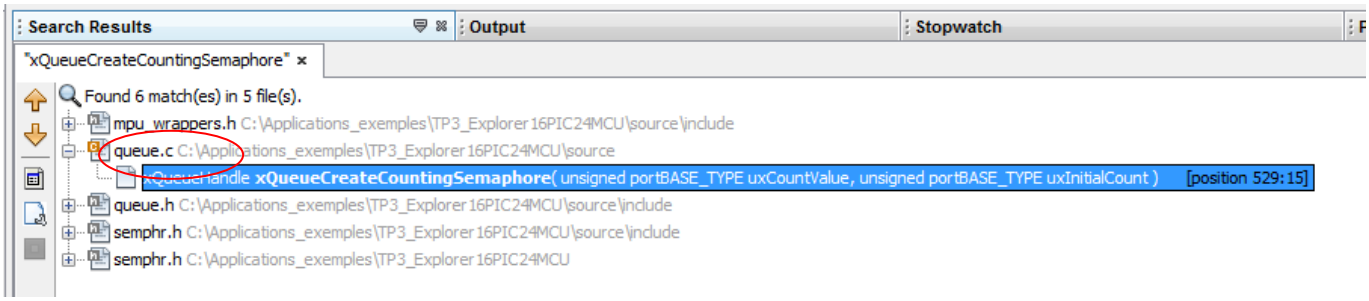
: undefined reference to `_xQueueCreateCountingSemaphore'

copiez "xQueueCreateCountingSemaphore" sans le _ , et allez dans le menu Edit/Find in projects pour chercher la définition dans le main project, comme illustré ci-dessous :

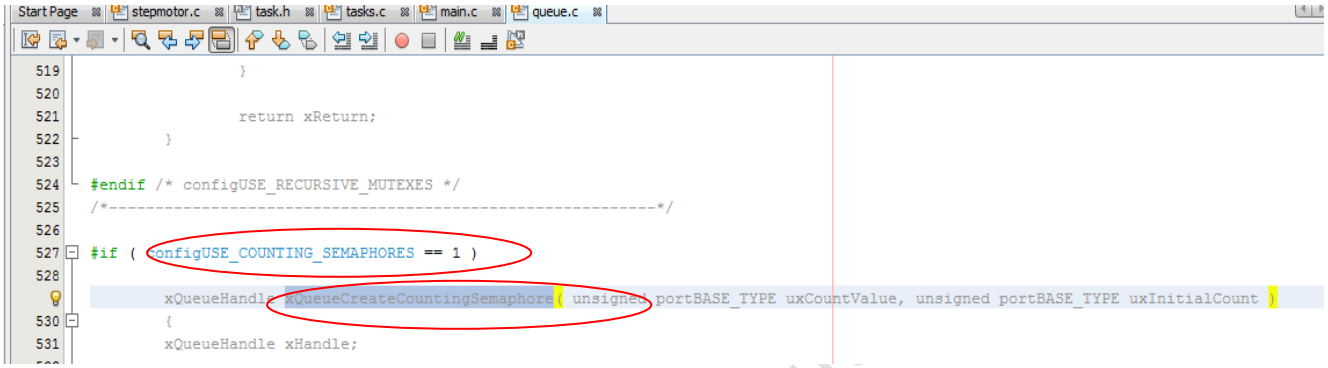


Dans les résultats de recherche, cherchez la définition de cette fonction ; on la trouve dans queue.c, mais elle est désactivée (grisée) :



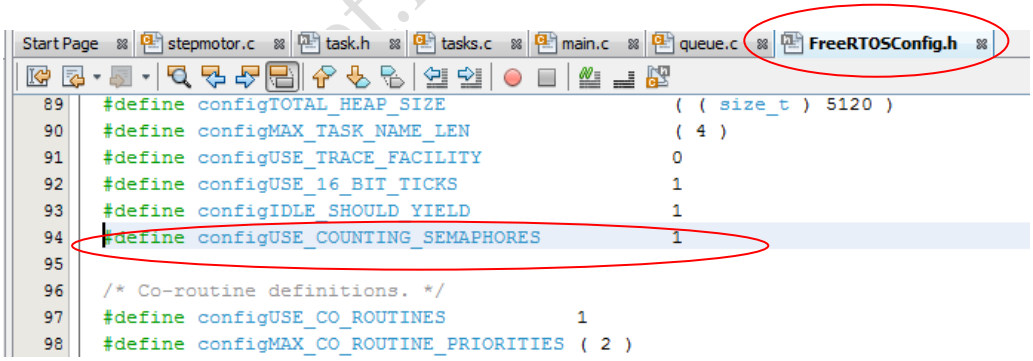


Double-cliquez sur cette définition le fichier queue.c s'ouvre et le curseur pointe sur la définition recherchée :



La source d'erreur est donc le fait que la macro config_COUNTING_SEMAPHORS n'a pas été définie à 1 pour activer la compilation de xQueueCreateCountingSemaphore( ) .

12. Maintenez la touche control appuiée et cliquez sur la macro configUSE_COUNTING_SEMAPHORES pour retrouver où cette macro est définie, ou allez au fichier FreeRtosConfig.h et ajoutez alors la définition de cette macro comme suit :



13. Clean and Buil votre projet : cette fois ci l'erreur doit disparaitre.
14. Faites le test avec les instructions INT1E=0 et INT1E=1 mises en commentaires dans la boucle while(1) dans vTaskINT1Handler( ). Vous devez constater que l'application ne traite qu'au maximum 3 impulsions apparaissant alors que le moteur est en rotation.
15. Faites le test avec les instructions INT1E=0 et INT1E=1 non mises en commentaires dans la boucle while(1) dans vTaskINT1Handler( ). Vous devez constater que l'application ne traite qu'au maximum une impulsion

apparaissant alors que le moteur est en rotation, même si on a un sémaphore pouvant contenir jusqu'à 3 jetons. Pourquoi ?

Exercice de Synthèse :

16. Ajoutez au design deux boutons poussoirs BPI connecté à RF6/INT0 et BPD connecté à RA14/INT3. On souhaite afficher une consigne sur le LCD sous forme : « Consigne : 0 » sur la première ligne et que :
    - une impulsion sur BPI incrémente cette consigne et la met à jour sur l'affichage.
    - une impulsion sur BPD décrémente cette consigne et la met à jour sur l'affichage.

    NB : la consigne ne peut être incrémentée ou décrémentée que dans la plage 0 à 20.

    Utilisez deux tâches vTaskINT0Handler( ) et vTaskINT3Handler( ) pour traiter les interruptions provoquées par les impulsions sur les deux boutons BPI et BPD, de manière différée en utilisant des sémaphores binaires.

    Remarque : La tâche d'affichage a déjà été utilisée dans le TP4.
17. Faites de telle sorte que le délai d'attente d'un pas du moteur soit calculé de la manière suivante :

    delai (en ms)=consigne*10+100.

    pour que le fait de changer la consigne fait varier la vitesse de rotation du moteur pas à pas :

    une consigne=0 donne une vitesse maximale (délai minimal=100ms).

    une consigne =20 donne une vitesse minimale (délai maximal=300ms).