Module : Systèmes temps réels

Manuel de Travaux pratiques

Préambule

Téléchargement des outils et préparation des TP :

1. Télécharger la version 9 du code sources de freertos (compressé en format zip) à partir de ce lien : https://sourceforge.net/projects/freertos/files/FreeRTOS/V9.0.0/FreeRTOSv9.0.0.zip/download
2. Décompresser le dossier compressé téléchargé dans le dossier C:\freertos. *(Ce Chemin n'est pas obligatoire mais permet d'être en phase avec les chemins absolus auxquels je fais référence dans ce document. Toutefois il est vivement recommandé d'éviter d'utiliser une arborescence trop longue pour le choix du dossier de décompression, ceci risque d'engendrer des problèmes lors de la compilation)*
3. Télécharger et installer l'environnement de développement de Microchip MPLABX : https://www.microchip.com/mplabx-ide-windows-installer
4. Télécharger et installer le compilateur XC16 de chez Microchip : https://www.microchip.com/mplab/compilers
5. Télécharger les fichiers de documentations à partir de :

https://drive.google.com/drive/folders/1H72WheK2N-ejYUPIUW-p80K2j7-1cVyC?usp=sharing

TP1

Concepts de programmation multitâches

Utilisation de freertos sur une plateforme basée sur un pic24F

1. Objectif :

Développer des applications qui utilisent le concept de la programmation multitâches, en particulier :

- Gestion des tâches : création, suppression, priorité, handle
- Les états d'une tâche : Running, Ready
- Algorithme d'ordonnancement (Scheduling Algorithm).

2. Mode opératoire :

2.1. Préparation du projet template :

Freertos a été porté (adapté) à plusieurs types de microcontrôleurs. Ce TP constitue une initiation à l'utilisation de freertos sur des PIC24F (Microcontrôleurs 16bits de chez Microchip) : notamment la procédure de création d'un projet basé sur Freertos sur l'environnement de développement intégré (IDE) MPLAB X.

2.2. Outils de travail :

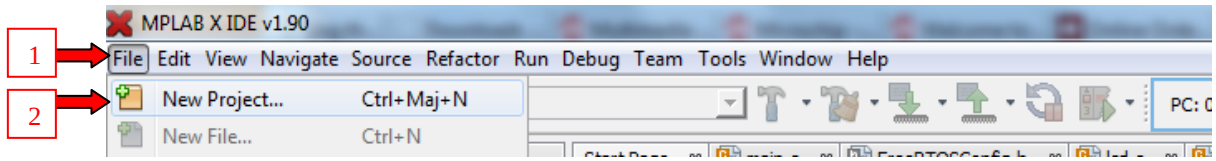
Dans ce TP vous aurez besoin, en plus de freertos que vous avez déjà téléchargé, de :

- un environnement de développement intégré IDE (Integreted Developpement Environnement), Mplab X, que vous avez aussi téléchargé et installé (de préférence sur le répertoire par défaut, soit donc C:\Microchip).
- Un compliateur pour les pic 16 bits, on utilisera le compilateur X16 déjà téléchargé et installé.
- Isis proteus (si vous avez la licence) ou une carte didactique avec un pic 24F comme par exemple 16 Explorer fourni par Microchip.

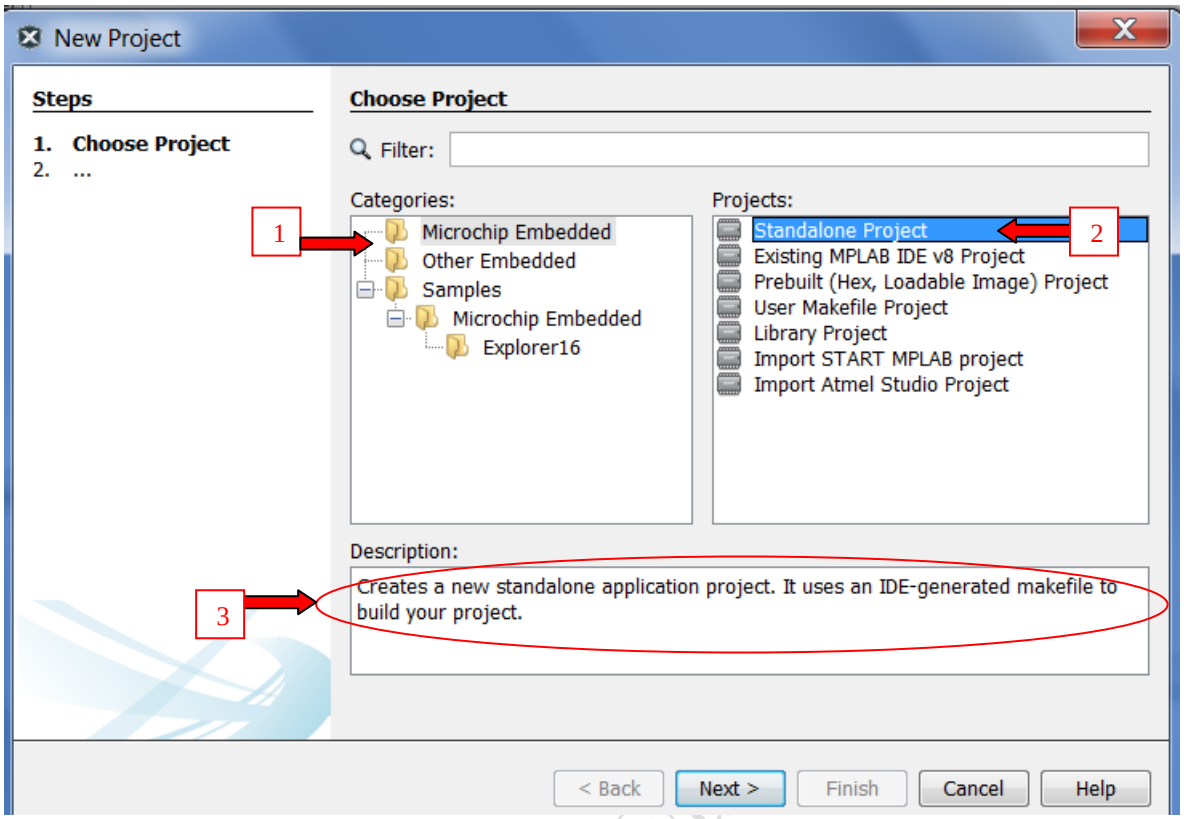
2.3. Initiation à la création d'un projet sur MPLABX

2.3.1. Conception du projet Minimal sur Mplab X :

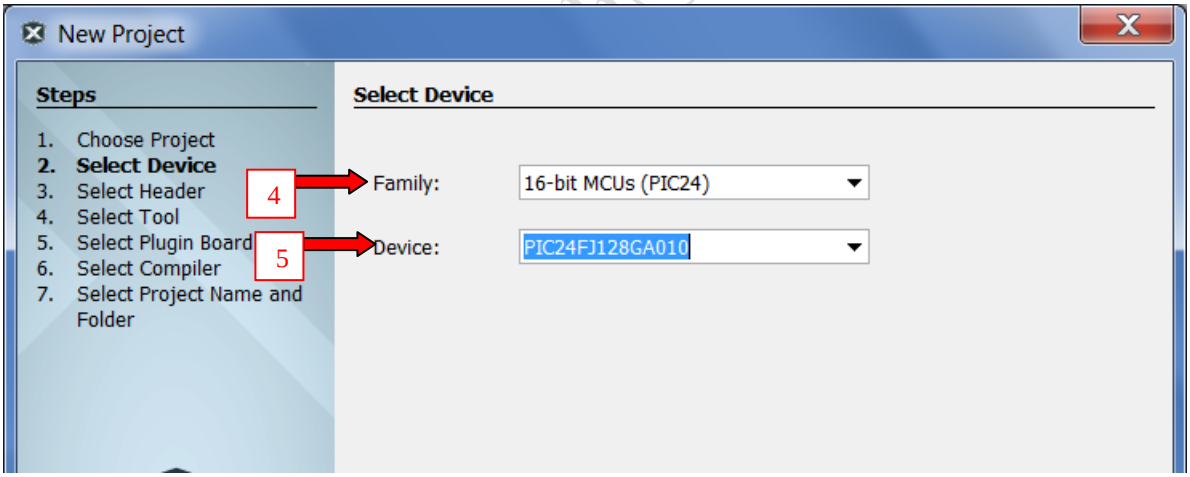
1. Démarrez Mplab X et créez un nouveau projet : A partir du menu File [1], choisissez New Project [2].



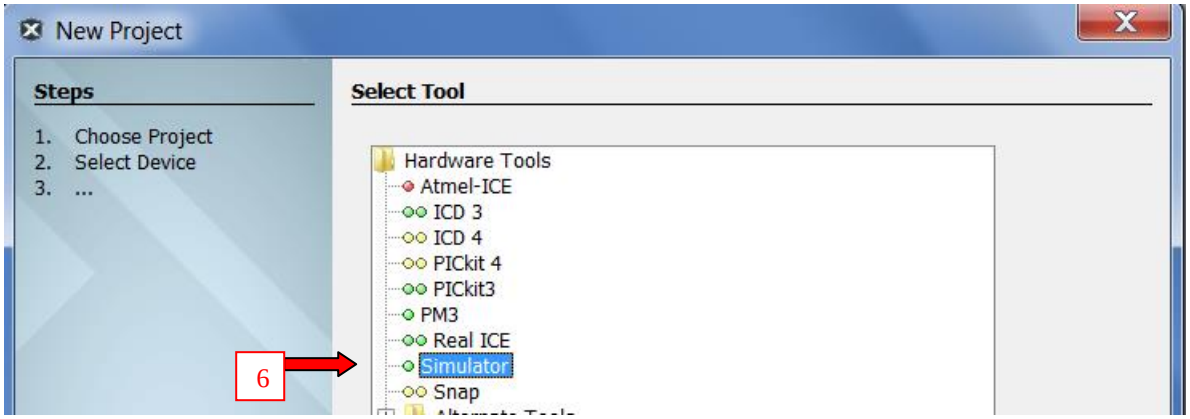
2. Dans Categories sélectionnez Microchip embedded [1], puis Projects, Standalone Project[2]



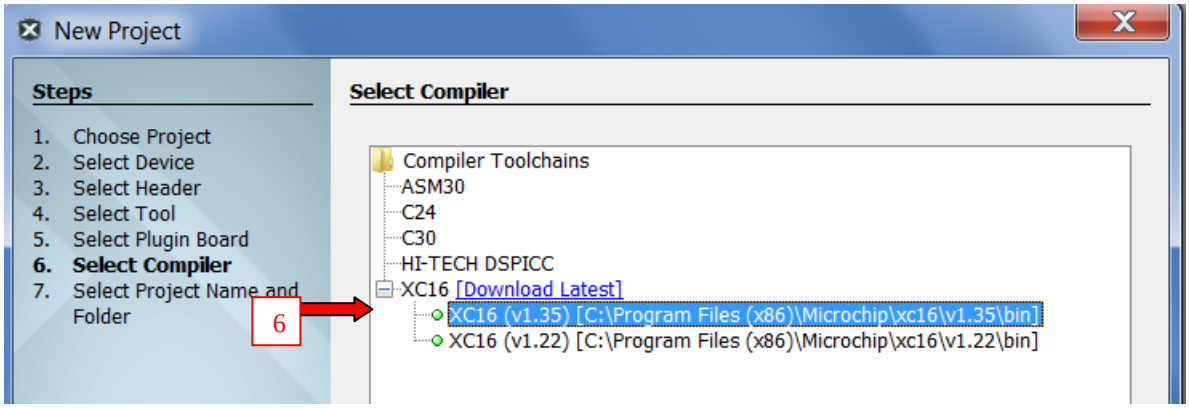
choix de la famille et du microcontroleur



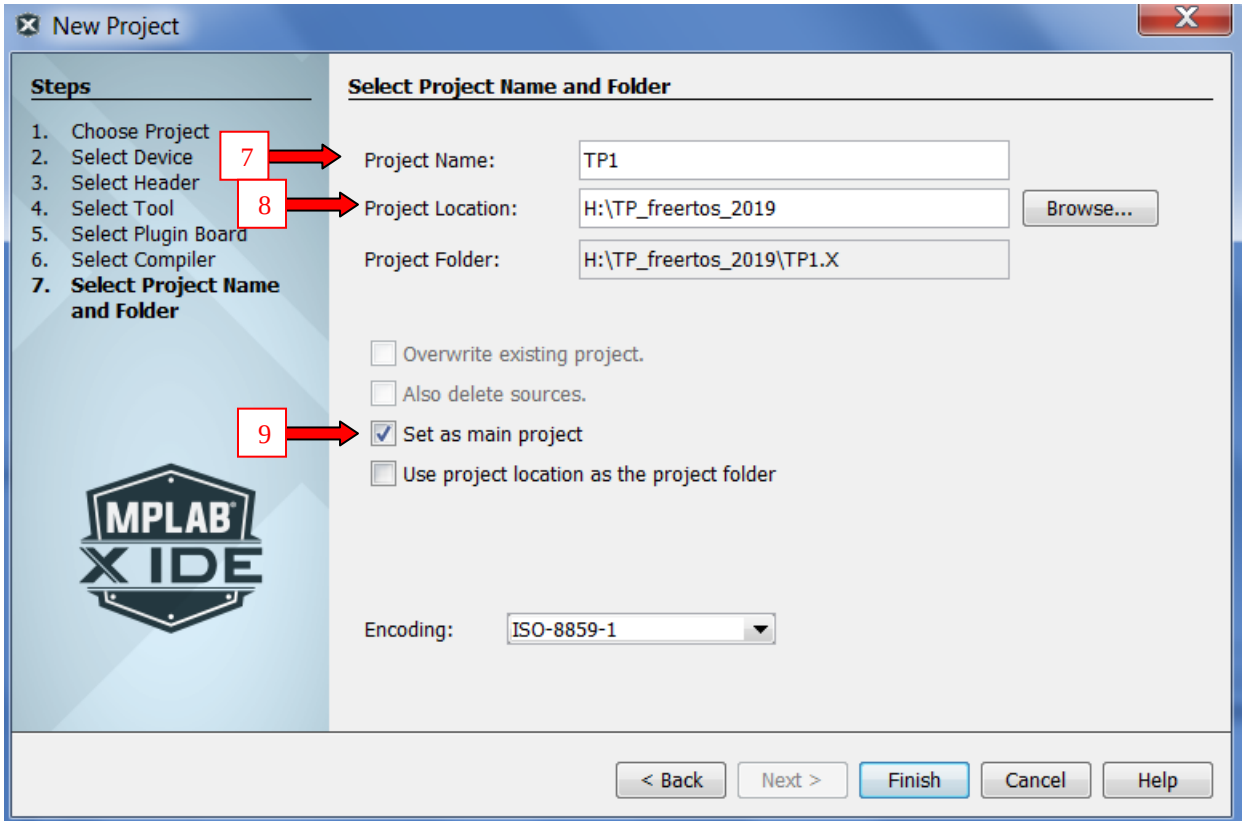
choix de l'outil du debuggage (ici simulateur)



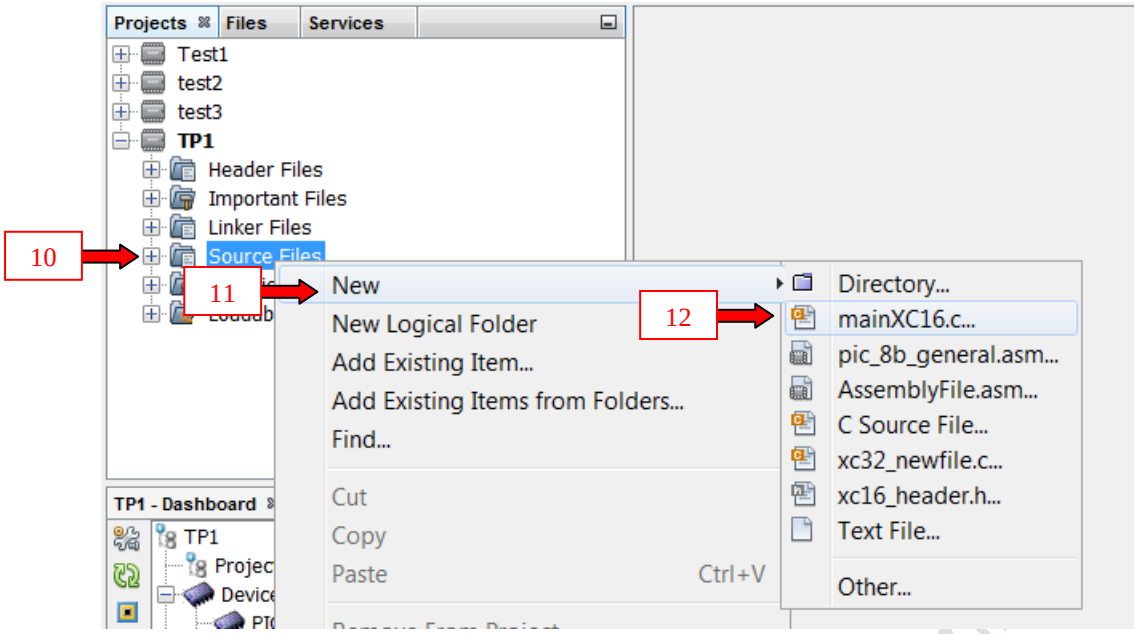
choix du compilateur



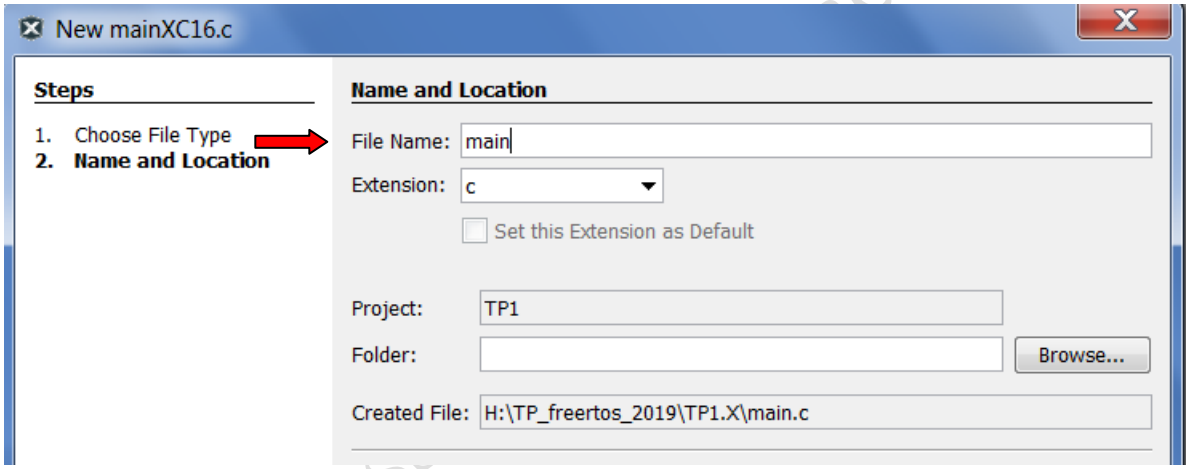
Choix du nom [7] du projet et de son emplacement[8] ; le rendre projet principal [9]



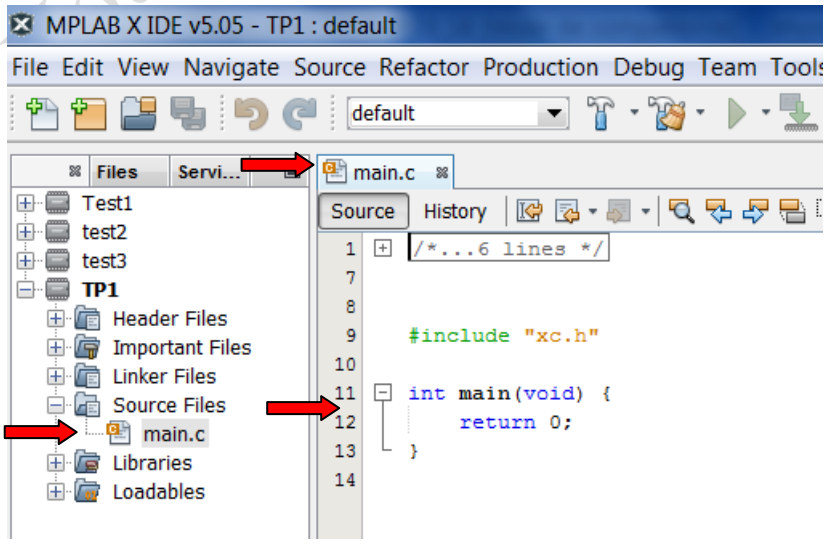
3. Un projet vide alors nommé TP1 sera créé dans IDE MPLABX.
4. Click droit sur le dossier sources pour ajouter un fichier source .c



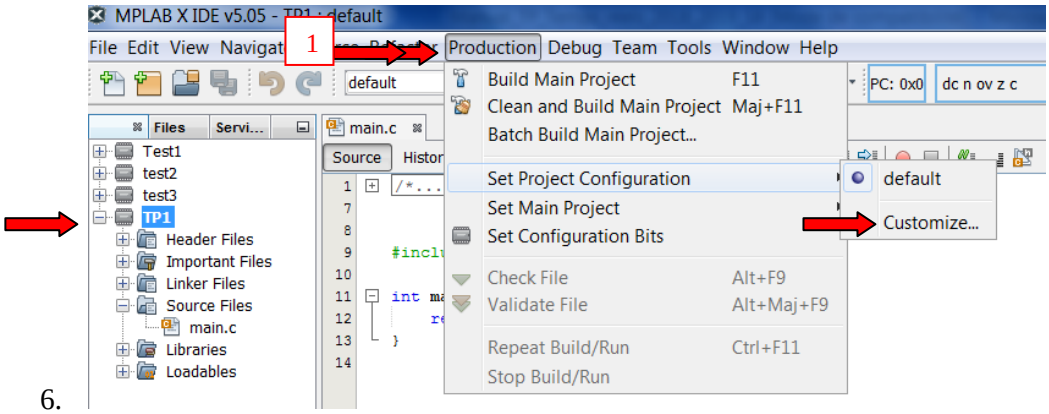
Renommez le fichier main.c



Ce fichier contient une fonction main vide :



5. Consultez les propriétés du projet comme suit ; pour cela



6.

1 : Allez au menu Production/Set Project Configuration / Customize …

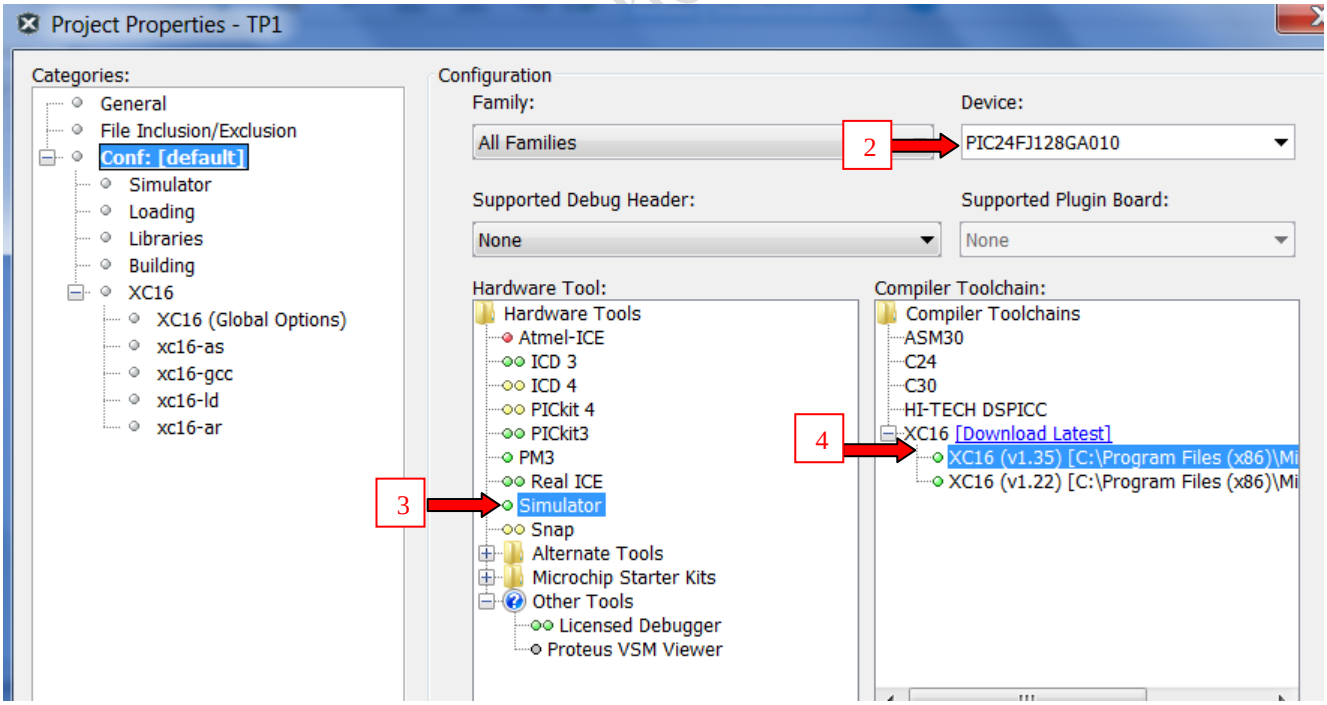
2 : Choix du microcontrôleur.

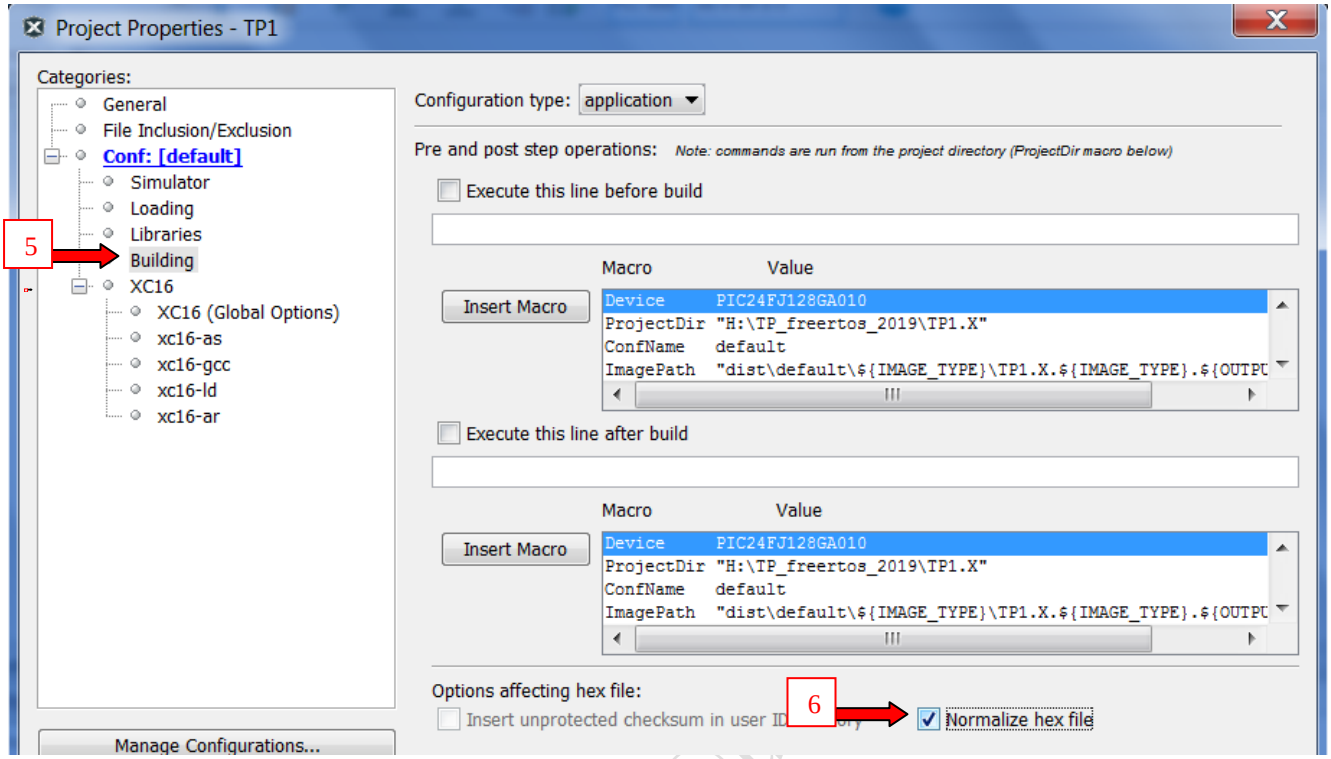
3 : Matériel utilisé pour débugger le programme sur le PIC (en mémoire flash) ; (ici on travaillera avec le simulateur de MPLABX).

4 : Le compilateur à utiliser (pour les pic 16 bits on utilise le compilateur X16 de Microchip

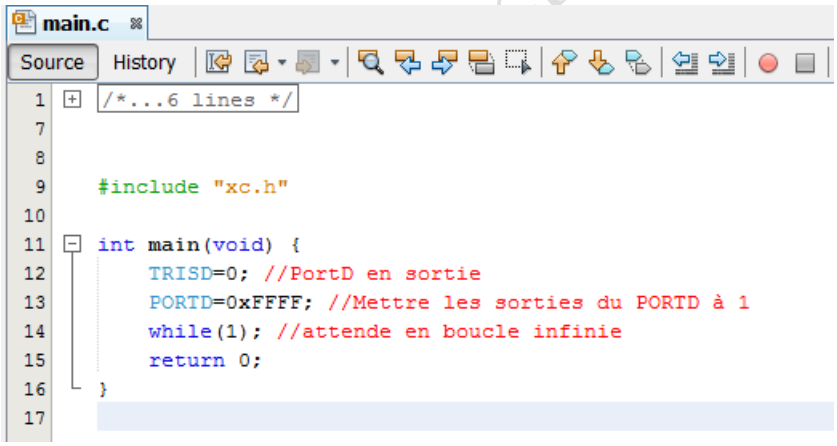
5 : Choisissez la rubrique Building

6 : Cochez la case Normalize hex file

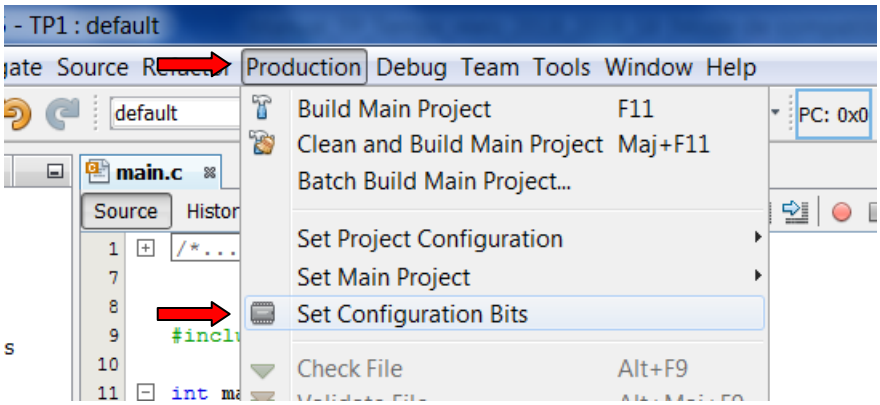


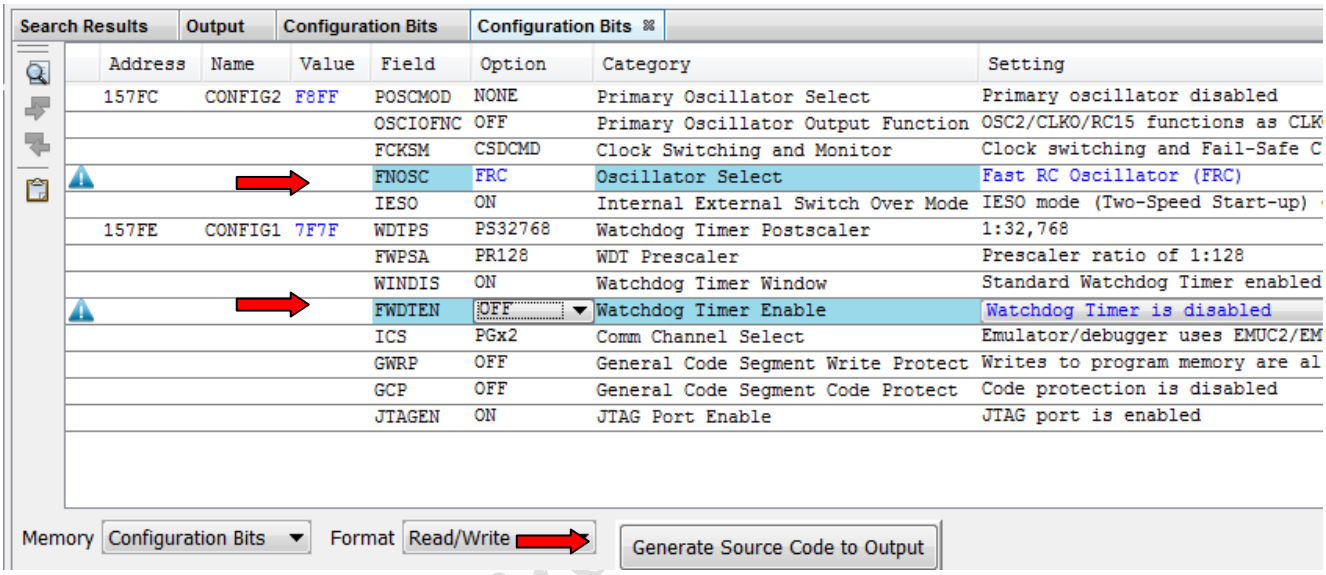


7. Confirmez les modifications et fermez la fenêtre des propriétés du projet TP1.
8. Ouvrez le fichier main.c ; personnalisez la fonction main () comme suit :

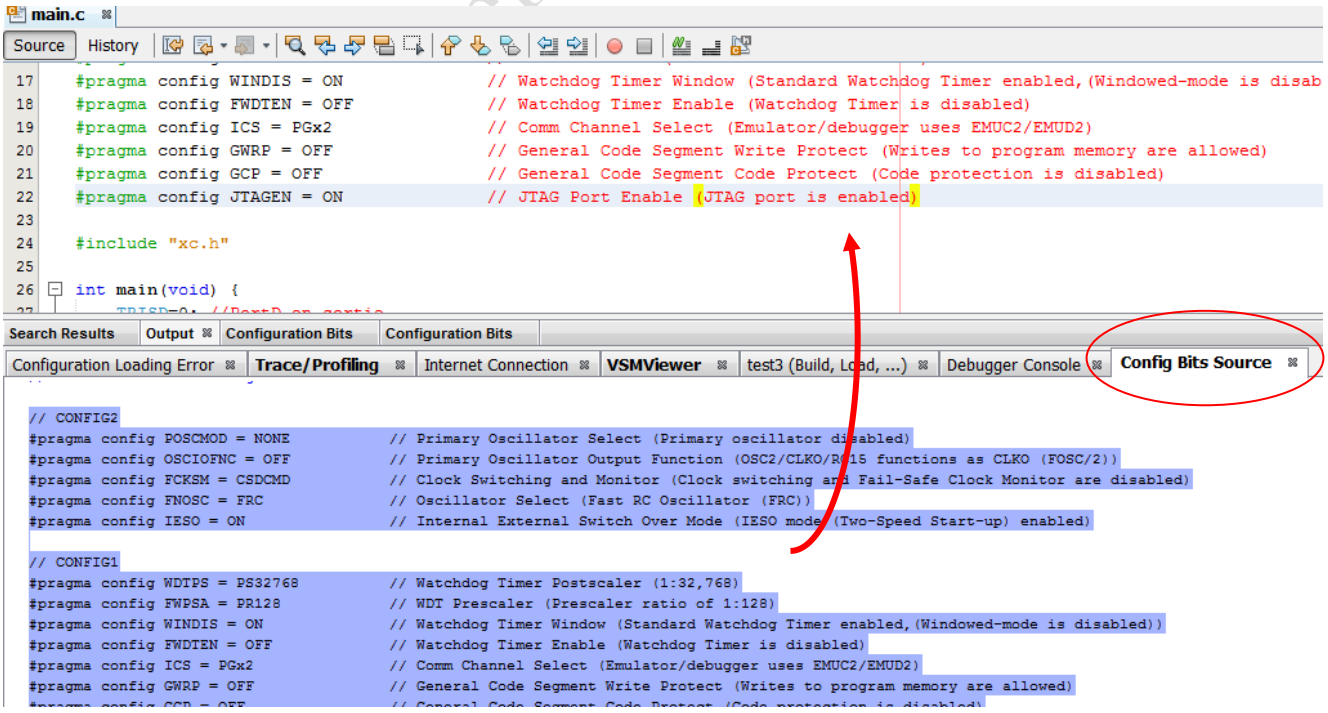


9. Modifiez les bits de configurations du microcontrôleur pour utiliser l'oscillateur interne FastRC et désactiver le watchdog , et génerer les pragma de configurations; Pour cela, suivre les étapes suivantes :



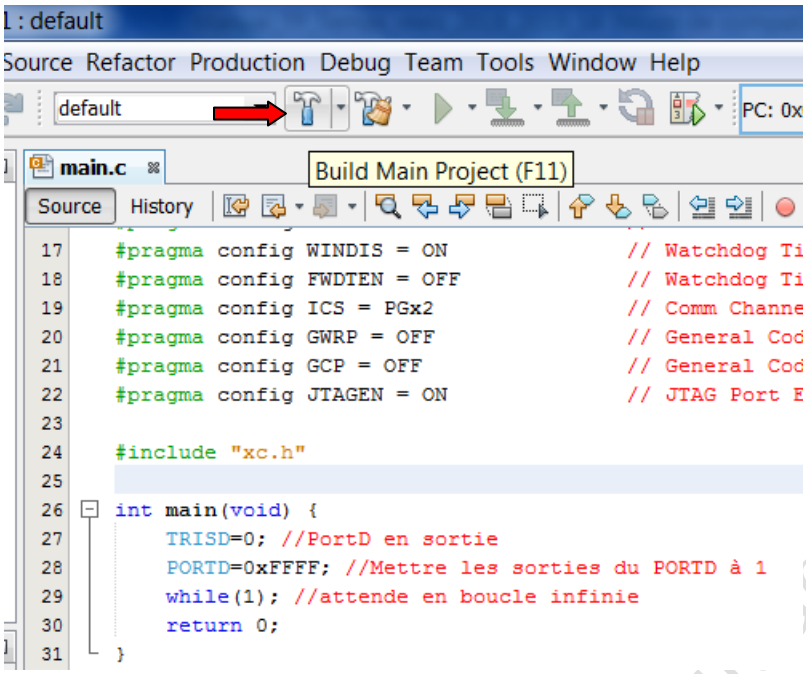


Copiez les pragmas générées et les copiez dans main.c avant #include <xc.h>

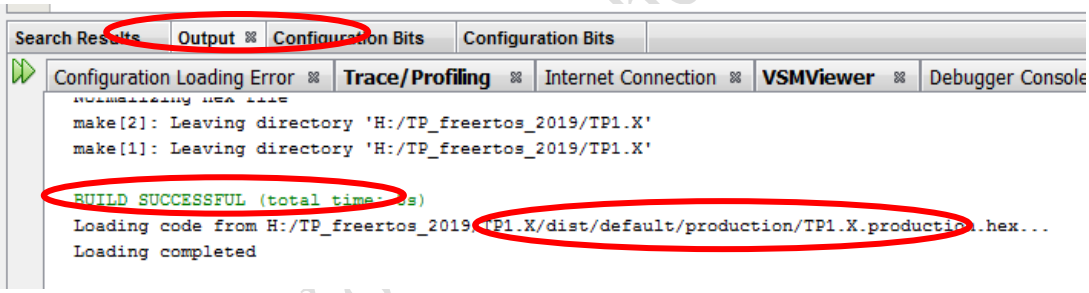


2.3.2. Construction et test du projet Minimal sur ISIS :

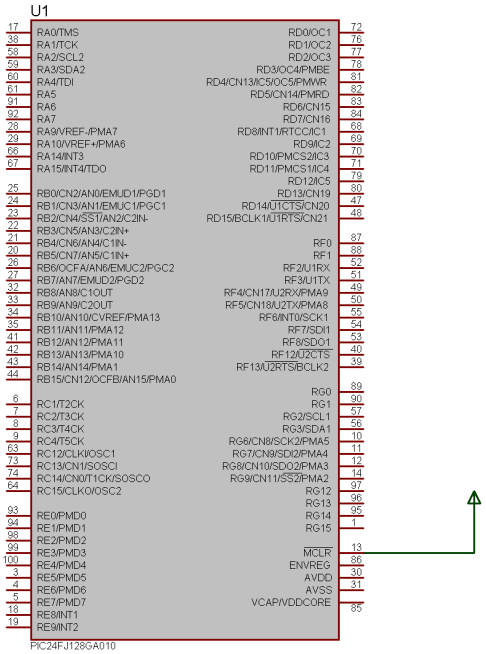
10. Construisez (Build) le fichier .hex à charger dans le PIC, en utilisant le raccourci ci_dessous



11. Cette construction de projet doit en principe réussir. Le message qu'on obtient dans la fenêtre de sortie doit ressembler à ce qui suit :



Avec l'indication du chemin ou le code .hex a été créé (**dossier_du_projet/dist/default/production/Nom_projet.X.production.hex**).



12. Démarrez ISIS et concevez un design minimal avec le microcontrôleur PIC24FJ128GA010 et n'oubliez pas de mettre le MCLR à 5V (voir figure ci-contre) ;

13. chargez dans le microcontrôleur le fichier .hex obtenu dans la question précédente, dont le chemin d'accès est :

dossier_du_projet/dist/default/production/Nom_projet.X.production.hex. Testez le bon fonctionnement de l'application (le PORTD doit se mettre à 1).

2.4. Initiation à la création d'un projet basé sur freertos

L'objectif maintenant est d'intégrer au projet précédent, le système d'exploitation temps réel freertos.

14. Commencez par vider la fonction main( ) et de remplacer son contenu par le code suivant , qui crée deux tâches puis démarre le scheduler:

```
int main(int argc, char** argv) {
// créer une tâche basée sur la fonction vTask1 de priorité 1 utilisant une pile de profondeur 150
 xTaskCreate(vTask1, "Task1", 150, NULL,  1,   NULL  );
// créer une tâche basée sur la fonction vTask1 de priorité 1 utilisant une pile de profondeur 150
 xTaskCreate( vTask2,  "Task2", 150, NULL, 1,  NULL  );
// Lancer le scheduler
 vTaskStartScheduler ();
// On atteint ce point si le scheduler s'est arrêté à cause d'un problème ou il a été arrêté explicitement par le code
 return (0);
}
```

15. Déclarez les prototypes des fonctions vTask1 et celui de vTask2 avant la fonction main () , comme suit:

```
void vTask1(void *pv1); // NB : le type de l'argument et le type retourné (voir cours)
void vTask2(void *pv2);
```

16. Définissez les fonctions vTask1 et celui de vTask2 après la fonction main ( ) , comme suit:

```
void vTask1(void *pv1){    // Blinking  PortA
TRISA=0;  // Setup PortA as digital Outputs
 while(1){    // une tâche est une boucle infinie
       PORTA=0x00;  delay();
       PORTA = 0xFF; delay(); }
  // ne retourne rien
}
  // ***********************************************
void vTask2(void *pv2) {    // Blinking  PortD
TRISD=0;
while(1){    // une tâche est une boucle infinie
       PORTD=0x00;       delay();
       PORTD = 0xFF;       delay(); }
  // ne retourne rien
}
```

17. Ajoutez le prototype de la fonction delay( ) avant main( ), et sa définition après main() comme suit:

```
void delay(void); //déclaration avant main( )
void delay(void){
unsigned int Compt=0X0FFF ;
while(Compt--) ;
}
```

18. Ajoutez le prototype de la fonction vApplicationIdleHook( ) avant main( ), et sa définition après main() comme suit:

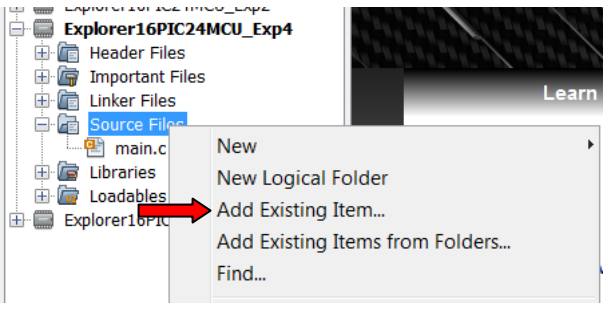
```
void vApplicationIdleHook(void); //déclaration avant main( )
….
//définition après main( )
void vApplicationIdleHook(void) {
// si on souhaite que la tâche idle execute un code on le met ici
}
```

19. Ajoutez la définition de la fonction vApplicationStackOverflowHook ( ) après main( ), et sa déclaration avant main() comme suit :

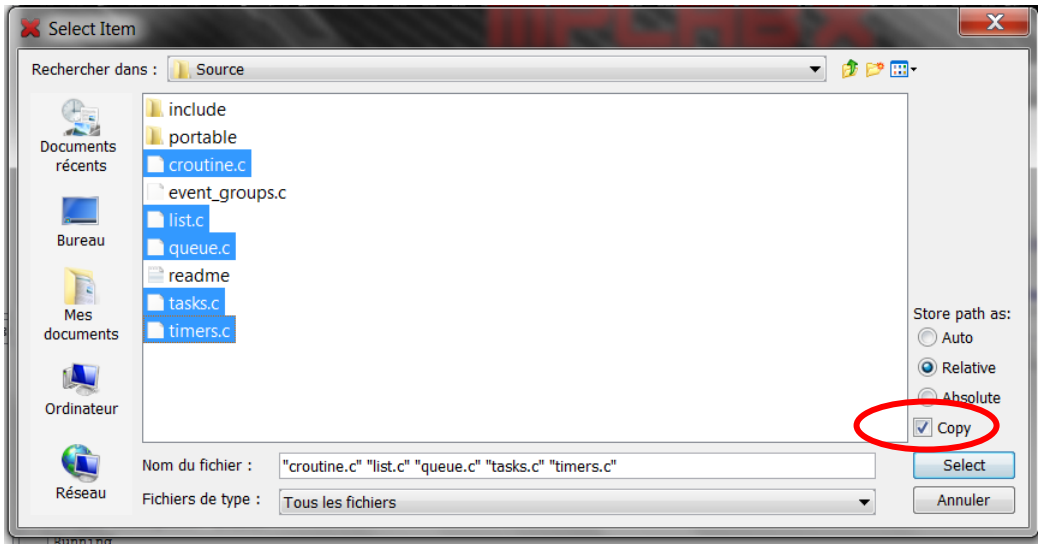
```
void vApplicationStackOverflowHook( TaskHandle_t xTask, char *pcTaskName ); //declaration avant main()
void vApplicationStackOverflowHook( TaskHandle_t xTask, char *pcTaskName ){
   //Que faire en cas de débordement de pile ?
}
```

NB : dans la suite du TP " freertos_doss " indique le dossier où vous avez placez freertos lors de sa décompression après son téléchargement.

20. Ajoutez au projet, le code source et les fichiers d'entête de freertos, qui sont indépendants du matériel ; dans l'arborescence du projet et au niveau du dossier **Source Files** faites un click droit puis **Add Existing Item… ,** allez dans le dossier où vous avez installé freertos :

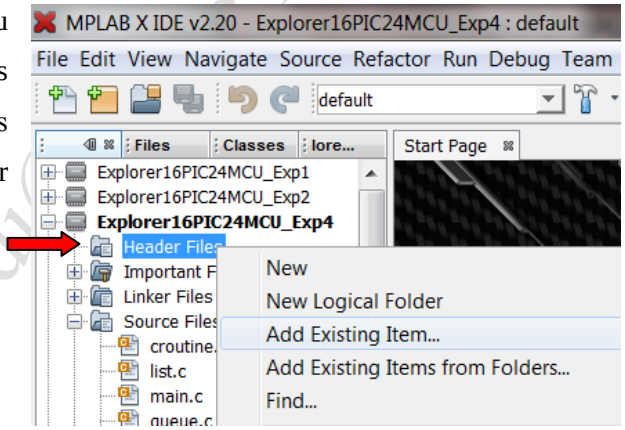


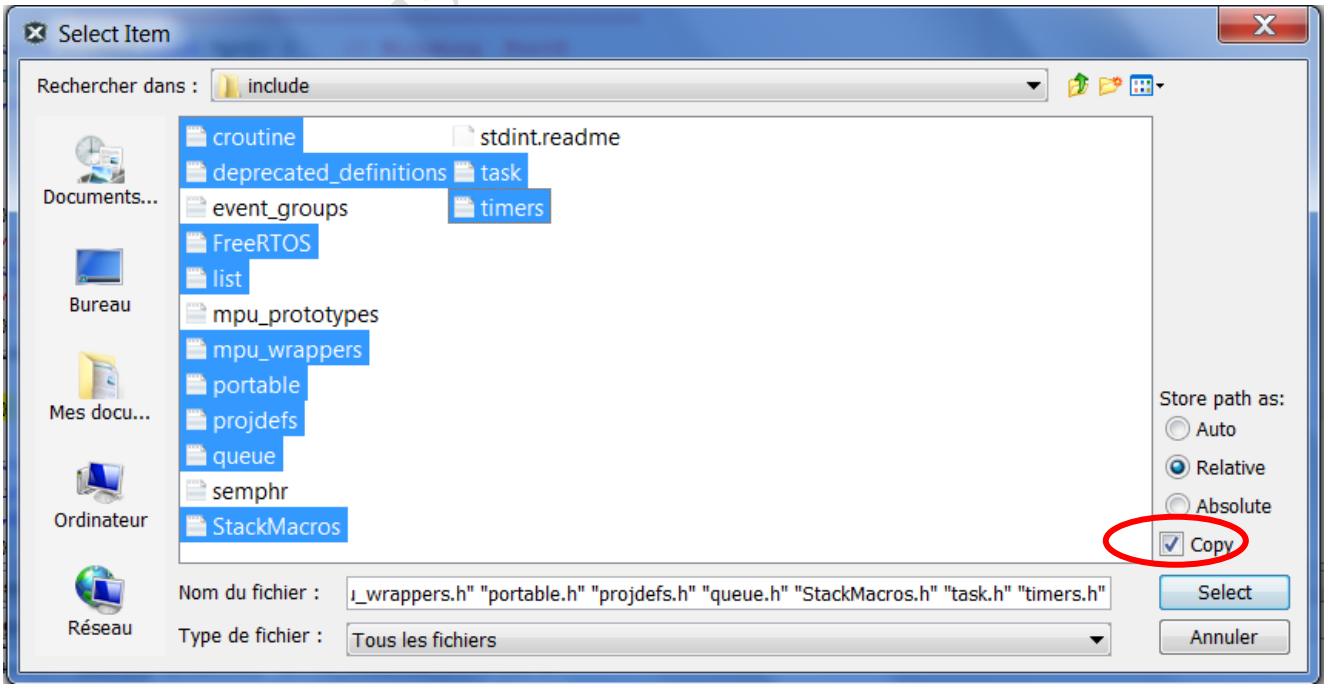
freertos_doss \FreeRTOS\Source , sélectionnez les fichiers comme le montre la figure suivante, cochez la **case copy** pour faire une copie de ces fichiers dans le dossier de votre projet, et valider par **Select.**



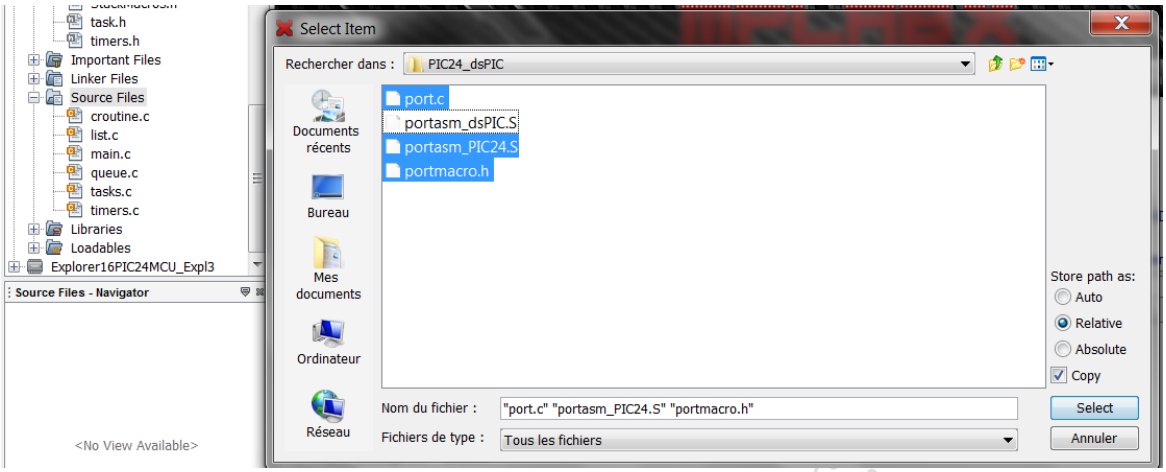
21. Refaites la même procédure pour ajouter, au niveau du dossier Header Files, les fichiers d'entête de freertos correspondants aux fichiers source ajoutés précédemment et d'autres, à partir du répertoire :

freertos_doss\FreeRTOS\Source\include

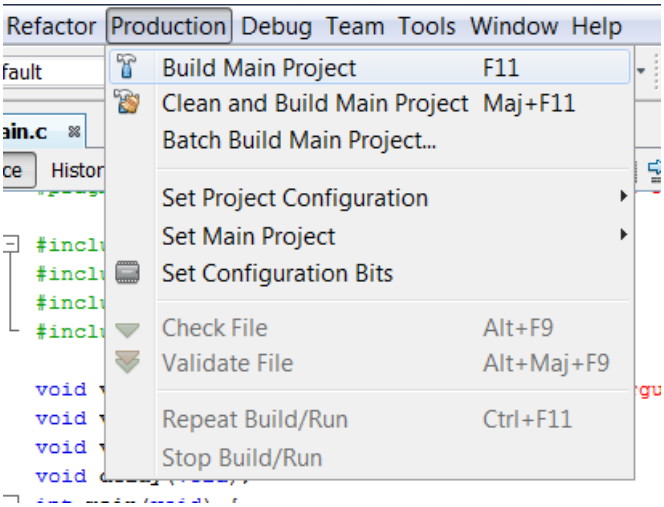




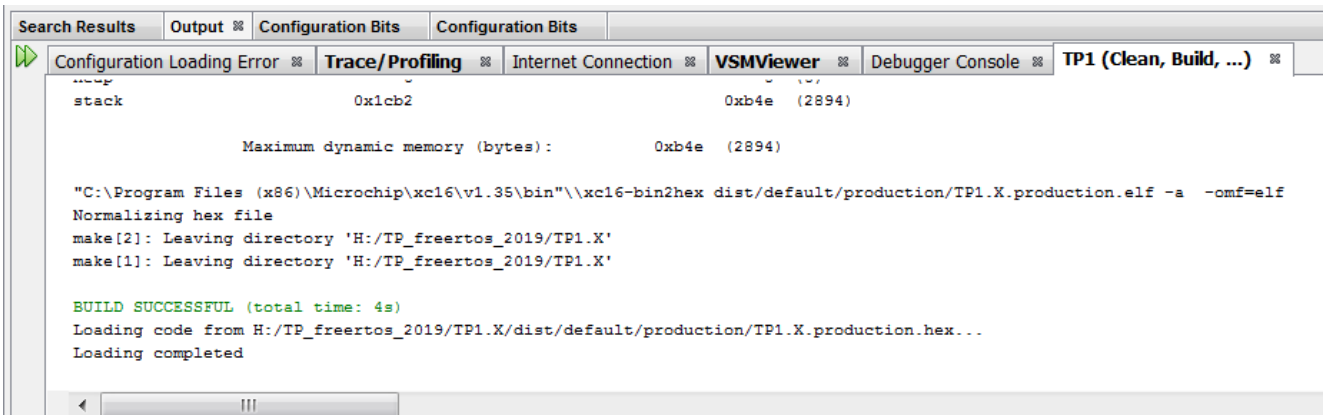
22. Refaites la même procédure pour ajouter, les fichiers de freertos correspondants aux fichiers source (C et assembleur) et d'entête qui dépendent du matériel (ici pic24F de chez Microchip) et qui sont "port.c" "portasm_PIC24.S" "portmacro.h", à partir du répertoire :
    freertos_doss\FreeRTOS\Source\portable\MPLAB\PIC24_dsPIC



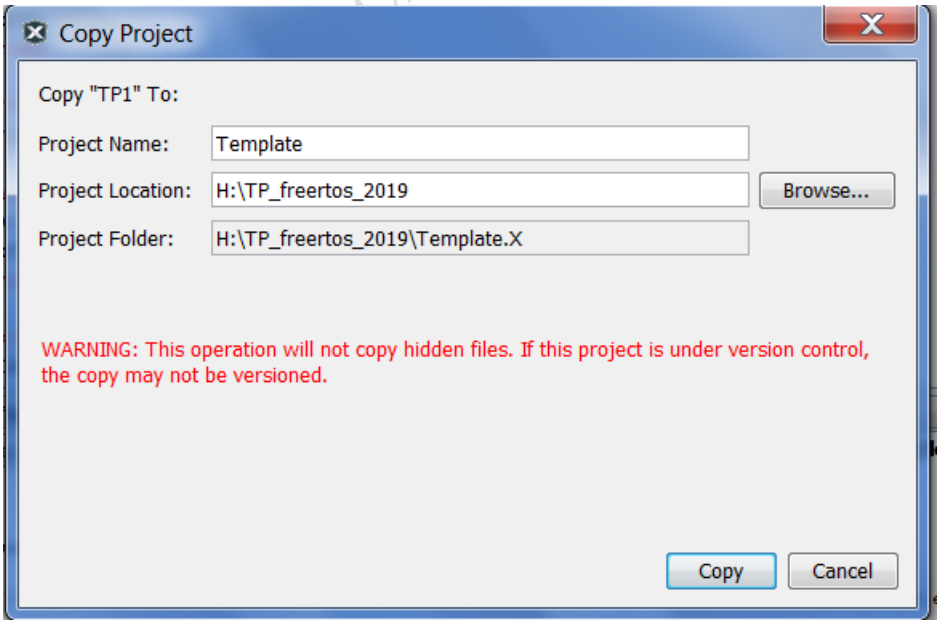
23. Freertos gère la mémoire en utilisant un fichier code source qui dépend de l'architecture matérielle ; ajoutez le fichier "heap_1.c" qu'on trouve dans le répertoire :
    freertos_doss\FreeRTOS\Source\portable\MemMang
24. Chaque application a besoin d'un fichier de configuration freeRTOSConfig.h. Ajoutez au projet le fichier freeRTOSConfig.h à partir du projet démo qui se trouve dans le répertoire : **freertos_doss \FreeRTOS\Demo\PIC24_MPLAB**
25. Dans le fichier main.c, ajoutez après #include <xc.h>, les directives de compilation suivantes:
    #include "FreeRTOS.h"
    #include "task.h"
    #include "list.h"
26. Construisez (Build) le projet :



A ce stade la construction du projet doit se faire avec succès, et vous devez obtenir un message comme le suivant :



27. Testez le bon fonctionnement à l'aide de Isis: les octets de poids faible des deux ports A et D doivent clignoter " simultanément".
28. *Le projet construit jusqu'à maintenant est un projet minimal qui intègre freertos ; il sera le point de départ pour les prochains TP puis sera modifié et adapté selon les besoins des prochaines applications, ainsi il est préférable d'en garder une copie qui servira comme template pour les prochains TP. Pour cela faites un click droit sur le nom du projet dans MPLAB X puis choisissez l'option* ***Copy…***
    Renommez le projet en lui donnant comme nom Template pour activer le bouton copy, et validez la copie. Vous disposez alors d'un projet Template qui servira dans les prochains TP.



2.5. Gestion des tâches

L'objectif maintenant est d'étudier la gestion des tâches : utilisation des handles des tâches, modification des priorités, tâche idle, suppression des tâches,….etc

29. Dans le projet construit jusqu'à maintenant les deux tâches Task1 et Task2 avaient la même priorité (1) et qui est supérieure à celle de la tâche idle (de priorité 0) : la tâche idle ne passe jamais à l'état Running et Task1 et Task2 se partagent le temps processeurs. Dans le code source, donnez à Task1 la priorité 2, et reconstruisez le projet. Testez l'application sur ISIS ; justifiez le nouveau comportement de l'application.
30. Donnez à Task1 et Task2, la même **priorité 0** que la tâche idle. Dans ce cas les trois tâches se partagent le temps processeur, pour vérifier, implémentez la fonction vApplicationIdleHook( ) de la manière suivante :

```
void vApplicationIdleHook(void) {
   // Blinking  Pin RC1
TRISC=0;
while(1){
      _RC1=0;    delay();
      _RC1=1;    delay(); }
}
```

Le code de vApplicationIdleHook() s'exécute si la tâche Idle passe à l'état Running. Testez votre application sur ISIS et remarquez que RC1 clignote.

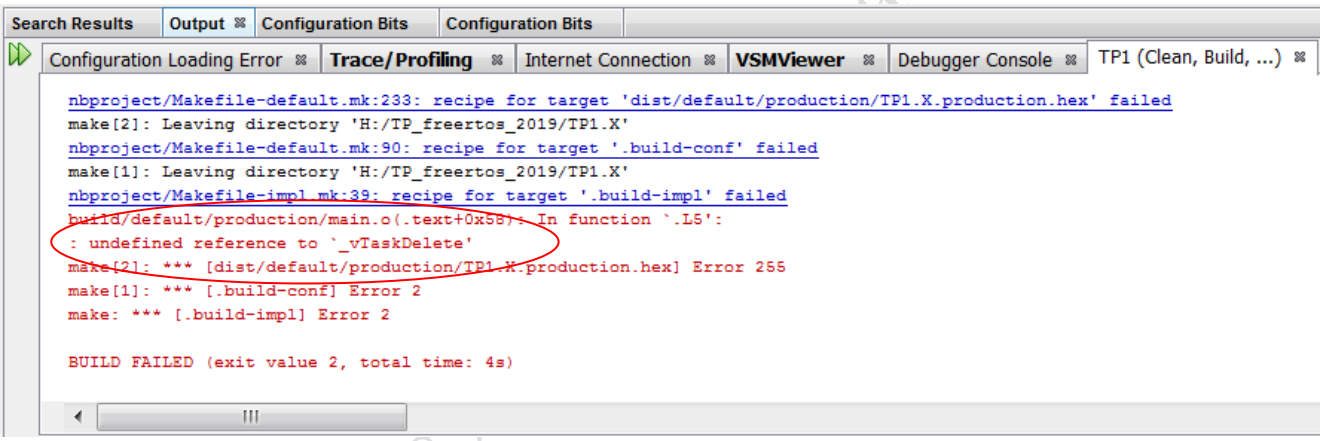
31. Remettez les priorités de Task1 et Task2 à la valeur 1. Testez l'application à nouveau et justifiez pourquoi RC1 ne clignote plus ?
32. Quand une tâche n'est plus utile elle peut être supprimée définitivement par la fonction de l'API freertos vTaskDelete( ). Modifiez les tâches précédentes de telle sorte qu'elles font clignoter le port correspondant 25 fois puis s'auto-détruisent :

```
void vTask1(void *pv1){
int i=25; //nombre de clignotemment
      // Blinking  PORTA
TRISA=0;
 // Setup PortA IOs as digital
 while(i){    // tant que i>0, boucle 5 fois
      PORTA=0x00;    delay();
      PORTA = 0xFF;  delay(); i--;
 }
  vTaskDelete(NULL);    // autodestruction de la tâche
}
```

```
// ************************************************
void vTask2(void *pv2) {
int i=25;
    // Blinking PORTD
TRISD=0;
while(i){    // tant que i>0, boucle 5 fois
      PORTD=0x00;   delay();
      PORTD = 0xFF; delay(); i--;
 }
vTaskDelete(NULL); // autodestruction de la tâche
}
```

Après que les deux tâches s'auto-détruisent seule la tâche Idle subsiste dans la liste de tâches Ready, et elle exécutera le code de la fonction vApplicationIdleHook( ) : ce qui fait clignoter RC1.

Essayez de construire le projet. Lisez l'erreur obtenue et déterminez la source qui empêche l'obtention du .hex.



Consultez le fichier donnant les détails sur la fonction API vTaskDelete( ).

Pour résoudre le problème, ouvrez le fichier freeRTOSConfig.h et changez le paramétrage :

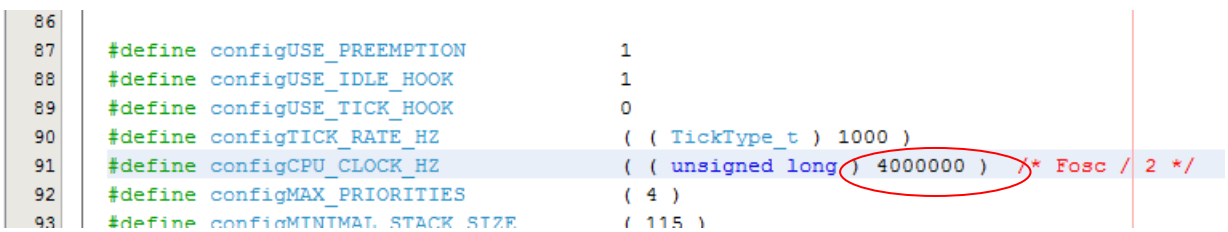
#define INCLUDE_vTaskDelete 0

en

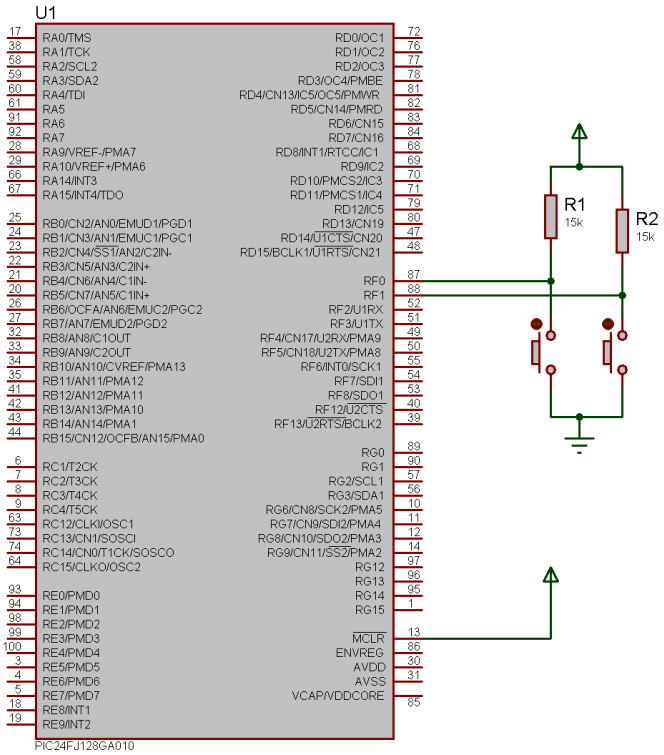
#define INCLUDE_vTaskDelete 1

Construisez le projet et Testez alors ce nouveau comportement de l'application.

Toujours dans freertosconfig.h , modifier la fréquence du CPU (Fosc/2) qui est par défaut à 16 000 000 Hz , à la valeur de 4 000 000 Hz, car Fosc=8MHz , correspond au type d'oscillateur qu'on a choisi au début (Fast RC interne).



33. Construisez à nouveau le projet et testez son bon fonctionnement : après 25 clignotements les ports ne clignotent plus, alors que _RC1 commence à clignoter. Interpréter ce résultat.
34. Remettez les deux tâches au fonctionnement initial soit un clignotement infini ; Modifiez le design d'ISIS pour rajoutez deux boutons poussoirs (voir figure ci-dessous) :



Une impulsion sur BP1 (relié à RF0) supprime Task1 et une impulsion sur BP2 (relié à RF1) supprime Task2. Ajoutez une 3ème tâche définie par la fonction vTask3() suivante :

```
void vTask3(void *pv2) {
TRISF=0x03; //RF0 et RF1 en entrée
short task1deleted=0, task2deleted=0;    // deux var pour mémoriser l'état de chaque tâche supprimée ou pas ?
while(1){   // boucle infinie
   if((_RF0==0)&& (task1deleted==0)) { vTaskDelete(Htask1); task1deleted=1;} // supprimer Task1
   if((_RF1==0)&& (task2deleted==0)) {vTaskDelete(Htask2); task2deleted=1;} // supprimer Task2
    if((task1deleted==1)&& (task2deleted==1))  break; // Task1 et Task2 supprimées quitter la boucle
   }
vTaskDelete(NULL); // supprimer Task3
}
```

La création des tâches doit préciser les handles des tâches :

```
xTaskCreate( vTask1, "Task1", 150, NULL, 1,    &Htask1 ); //Hask1 est le handle (identificateur) de la tâche 1
xTaskCreate( vTask2,  "Task2", 150, NULL, ,   &Htask2  ); //passage par adresse !!!
 xTaskCreate(vTask3, "Task3", 150, NULL, 1,   NULL  );// on n'a pas besoin de handle pour cette tâche
```

N'oubliez pas la déclaration des variables Htask1 et Htask2 avant la fonction main ()

```
xTaskHandle  Htask1, Htask2  ;
```

35. Construisez à nouveau votre projet et testez son bon fonctionnement. Quand est ce que la broche _RC1 se mette à clignoter ?

36. La priorité d'une tâche peut être lue ou modifiée après la création de la tâche, respectivement par les deux fonctions de l'API freertos :

```
unsigned portBASE_TYPE uxTaskPriorityGet( xTaskHandle pxTask );
```

et

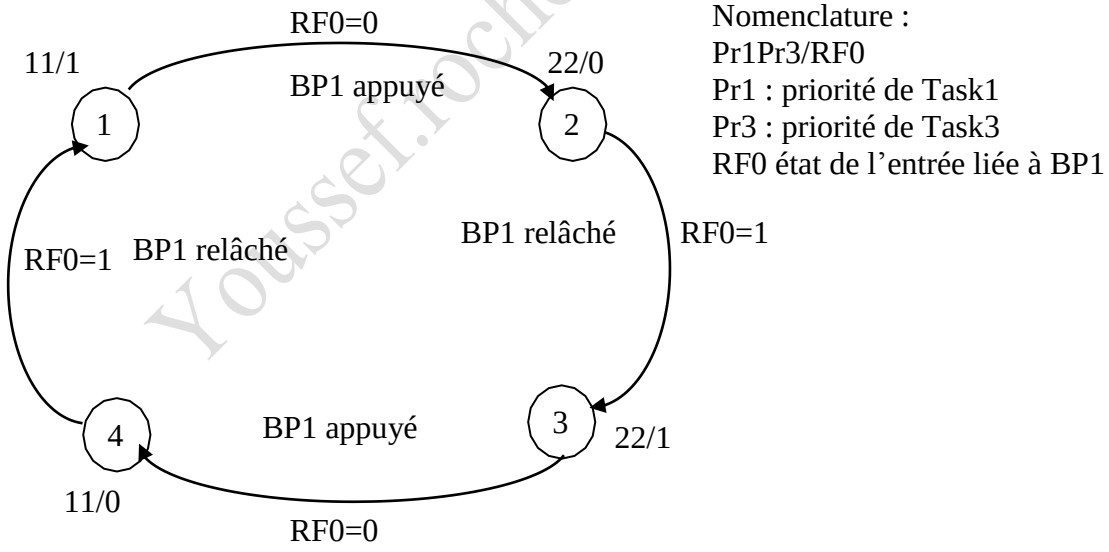
```
void vTaskPrioritySet( xTaskHandle pxTask, unsigned portBASE_TYPE uxNewPriority );
```

à condition que le deux macros suivantes soient définies toutes les deux à 1 dans le fichier freeRTOSConfig.h :

```
#define INCLUDE_vTaskPrioritySet 1
#define INCLUDE_uxTaskPriorityGet 1
```

Donnez comme priorité aux trois tâches la valeur 1. Modifiez le code de la fonction vTask3 pour que lorsqu'on appuie sur BP1 la priorité de Task1 est modifiée et prend la valeur 2, une autre impulsion la remet à 1. Le fonctionnement de la tâche 3 est décrit par le diagramme d'états suivant (machine à états finie):

Modélisation du fonctionnement de la tâche 3 par une machine à états finis



Analysez le code de vTask3, construisez le projet et expliquez le changement du comportement de l'application quand on appuie sur BP1 une seule fois puis deux fois.

```
void vTask3(void *pv2) {
  TRISF = 0x03; //RF0 et RF1 en entrée
  short state = 1; //cas var qui indique l'état
  while (1) { // boucle infinie
    switch (state) {
```

```
//state=1 Task1 a comme priorité 1 et BP1 non appuyé pour la première fois
            case 1: //state=1
                if (_RF0 == 0) //si BP0 appuyé RF0==0
               //alors donnez à la tâche courante la priorité 2 d'abord ensuite faite de même pour la tâche 1
              //on passe à l'état 2
                 {vTaskPrioritySet(NULL, 2); vTaskPrioritySet(Htask1, 2);state = 2;} break;
//state=2  Task1 a comme priority 2 et BP1 a été appuyé une première fois et n'est pas encore relâché
          case 2: if (_RF0 == 1) state = 3;break;  //si BP0 relâché RF0==1 passer à l'état 3
//state=3  Task2 a comme priority 1 et BP1 a été appuyé une première fois et relâché
          case 3: if  (_RF0 == 0)
//mettre d'abord la priorité de tâche 1 à 1 puis faire de même pour la tâche courante
          { vTaskPrioritySet(Htask1, 1); vTaskPrioritySet(NULL,1), state = 4;}
             break;
//state=4  Task1 a comme priority 1 et BP1 a été appuyé une deuxième fois et n'est pas encore relâché
          case 4:  if (_RF0 == 1) state = 1; break;
       }
   }
   vTaskDelete(NULL);
}
```

Exercice de synthèse :
Complétez vTask3 pour faire la même chose avec BP2 et Task2.
Etablir d'abord le diagramme des états. Chaque état sera caractérisé par les priorités des 3 tâches Pr1Pr2Pr3 et l'état des deux entrées RF0RF1.
Nombre d'états possibles :

Pr1Pr2Pr3/RF0RF1 (chaque variables deux états → 2^5=32 états) mais il y a des états impossibles :

221/XY (4états), 211/XY (4états), 121/XY (4 états) , 112/XY(4états)→ 16 états à retrancher, il nous reste 16 états .

Pour simplifier le diagramme on suppose que jamais RF0 et RF1 changent d'état au même temps : c'est-à-dire si par exemple RF0RF1=11 , on peut passer vers états caractérisés par RF0RF1=10 ou RF0RF1=01 et jamais vers l'état RF0RF1=00.

On peut modéliser la machine à états finis par une table de transition (à compléter) :

	1 111/00	2 111/10	3 111/01	4 111/11	5 212/00	6 212/10	7 212/01	8 212/11	9 122/00	10 122/10	11 122/01	12 122/11	13	14	14	16
1 111/00	→	→	→	X				X				X				
2 111/10		→	X	→	→		X				X					
3 111/01		X	→	→	→	X										
4 111/11				→	X		→			→						
5																

212/00																
6 212/10																
7 212/01																
8 212/11																
9 122/00																
10 122/01																
11 122/10																
12 122/11																
13 222/00																
14 222/01																
15 222/10																
16 222/11																