TP2

Exploration du diagramme des états

(Plateforme basée sur un pic24F)

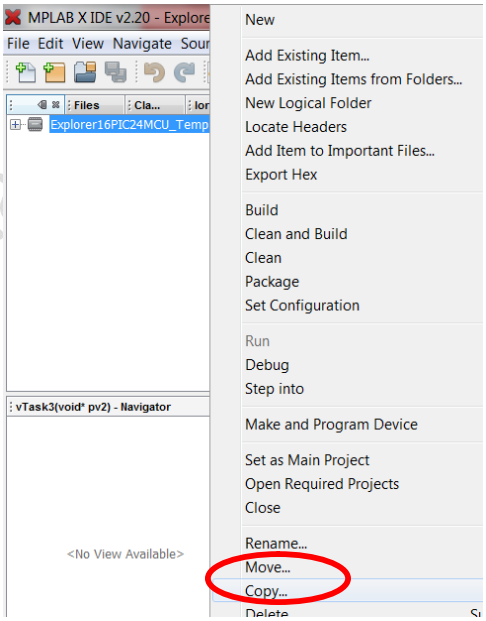
1. Objectifs :

Explorer les autres états dôune tâche :

- Blocage/déblocage, suspension/reprise de tâches
- Création de plusieurs occurrences dôune même tâche
- Initiation au débogage couplé MPLABX-ISIS

2. Mode opératoire :

2.1. Utilisation dôune copie du projet « template » créé en TP1:

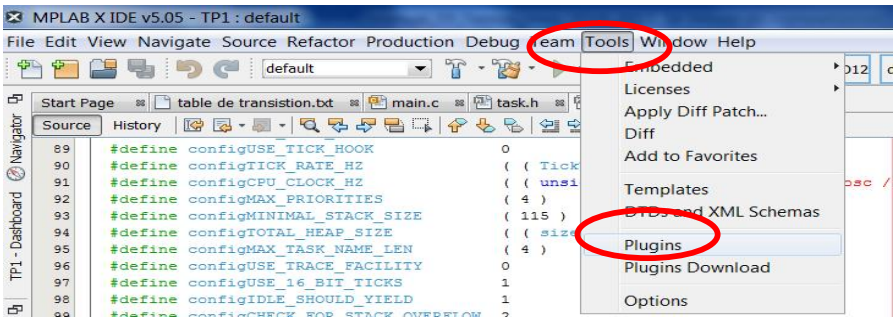


Dans MplabX, ouvrez le projet template crée en TP1 et faites une copie et **renommer** le en TP2 par exemple. **Dans la suite de ces TP, on travaillera avec cette copie.**

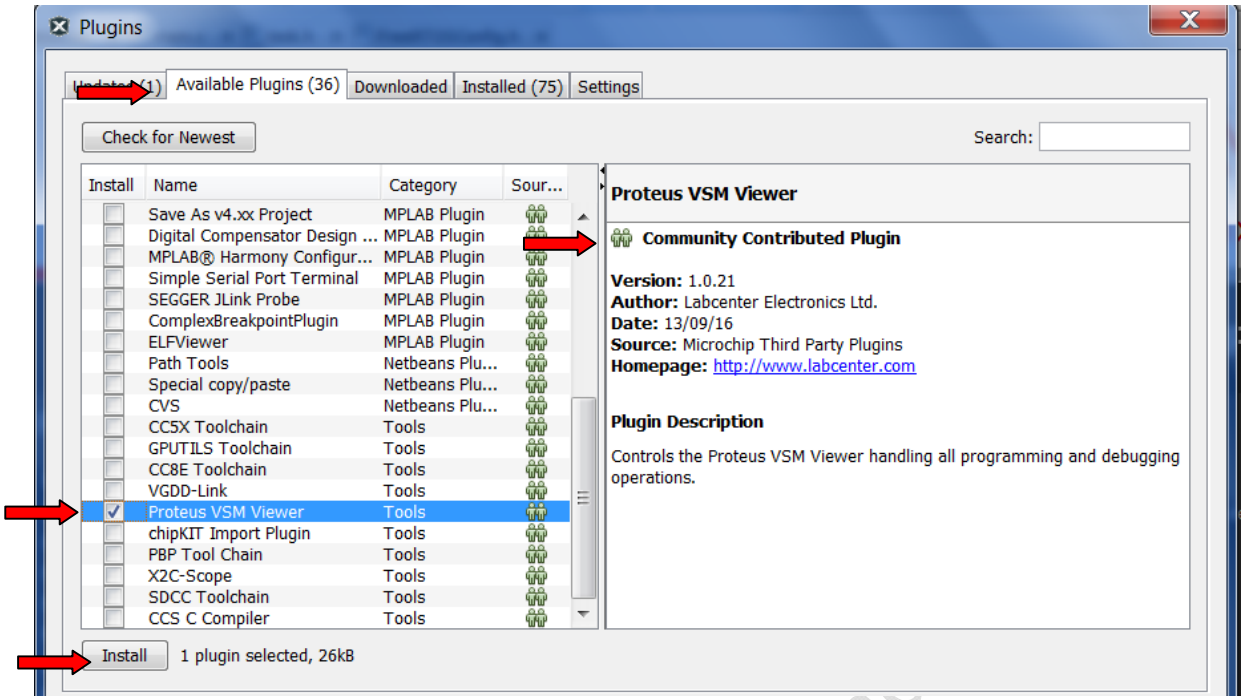
2.2. Outils de travail :

1. Dans ce TP vous aurez besoin, en plus de freertos que vous avez déjà téléchargé, de lôIDE Mplab X et ISIS, du **plugin Proteus VSM Viewer** que vous devez ajouter à MPLABX. Ce plugin permet de connecter MPLABX à ISIS et de faire un débogage sur MPLABX couplé à une simulation sur ISIS.

   Allez dans le menu Tools/Plugins

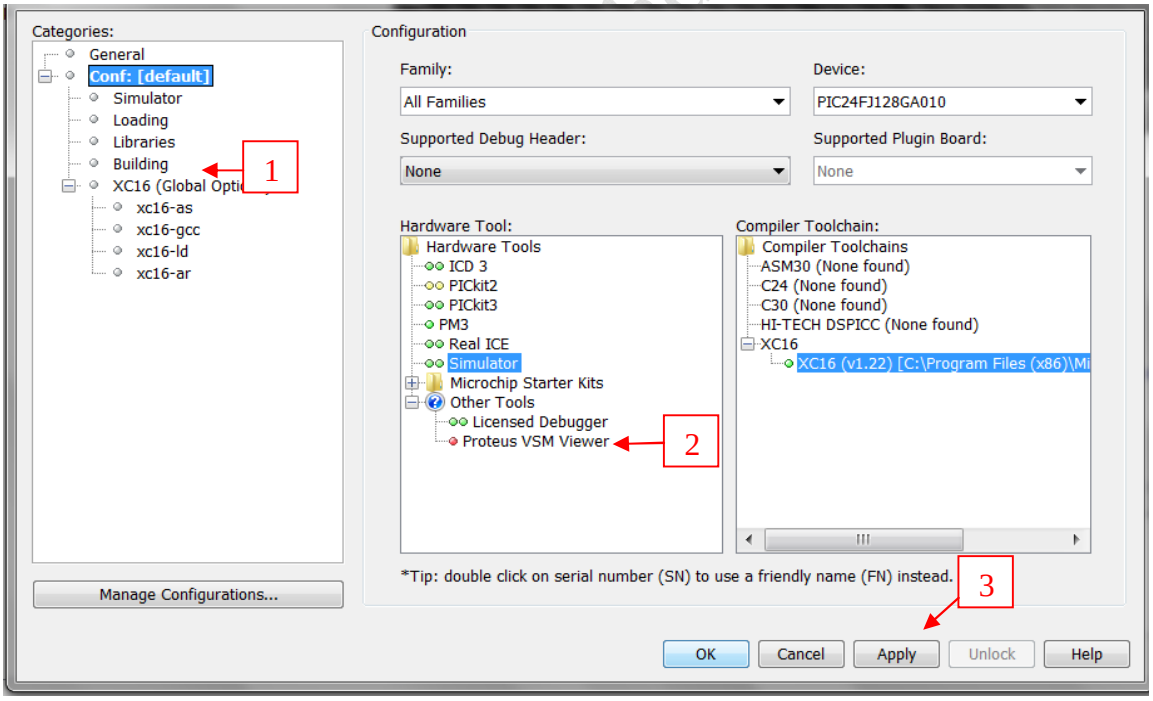


   Cochez la case pour le plugin Proteus VSM Viewer et cliquez sur install (Ceci nécessite une connexion Internet pour télécharger le plugin).



Redémarrez MPLABX une fois le téléchargement et lôinstallation terminés.

2. Faites un clic droit sur le projet TP2, Sélectionnez lôoption **Set configurationé Customize**, vous obtenez la fenêtre suivante :

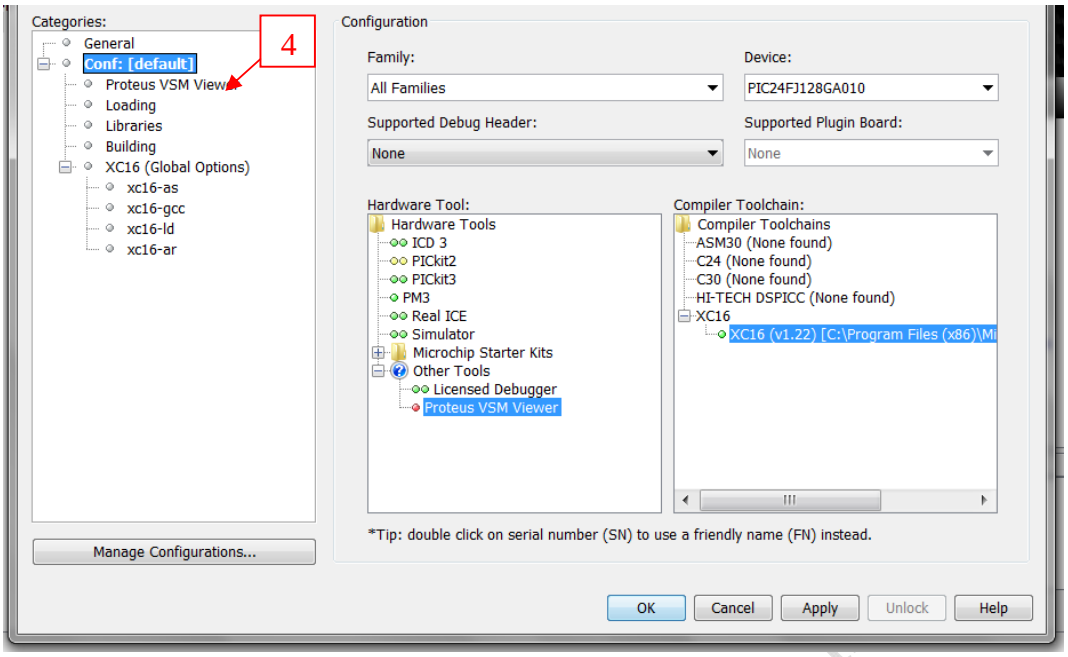


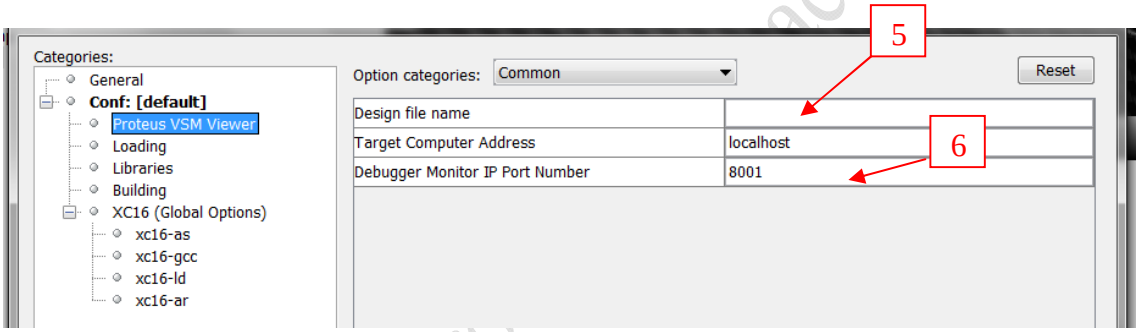
1 : Notez que par défaut lôoutil qui sera utilisé lors du débogage est le simulateur de MPLABX.

2: Changez dôoutil de débogage en sélectionnant Proteus VSM Viewer.

3 : puis cliquez sur Apply pour appliquer la modification

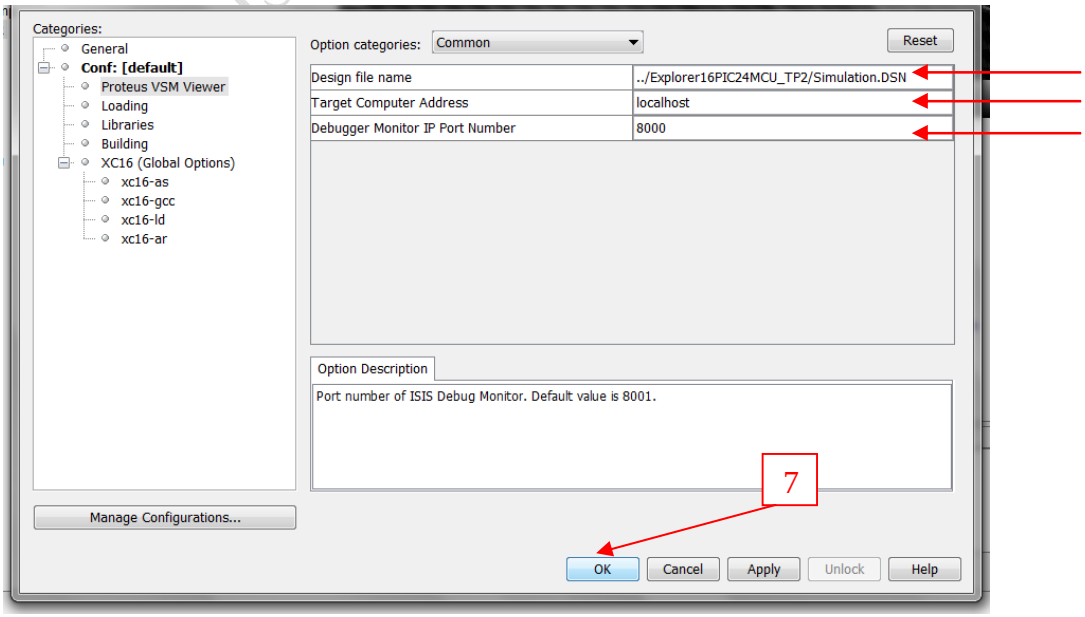
4 : la fenêtre se transforme comme suit, sélectionnez Proteus VSM Viewer pour montrer ces propriétés





5 : Faites un clic pour parcourir lôarborescence des fichiers à la recherche du fichier design conçu sous ISIS avec lequel se fera la simulation.

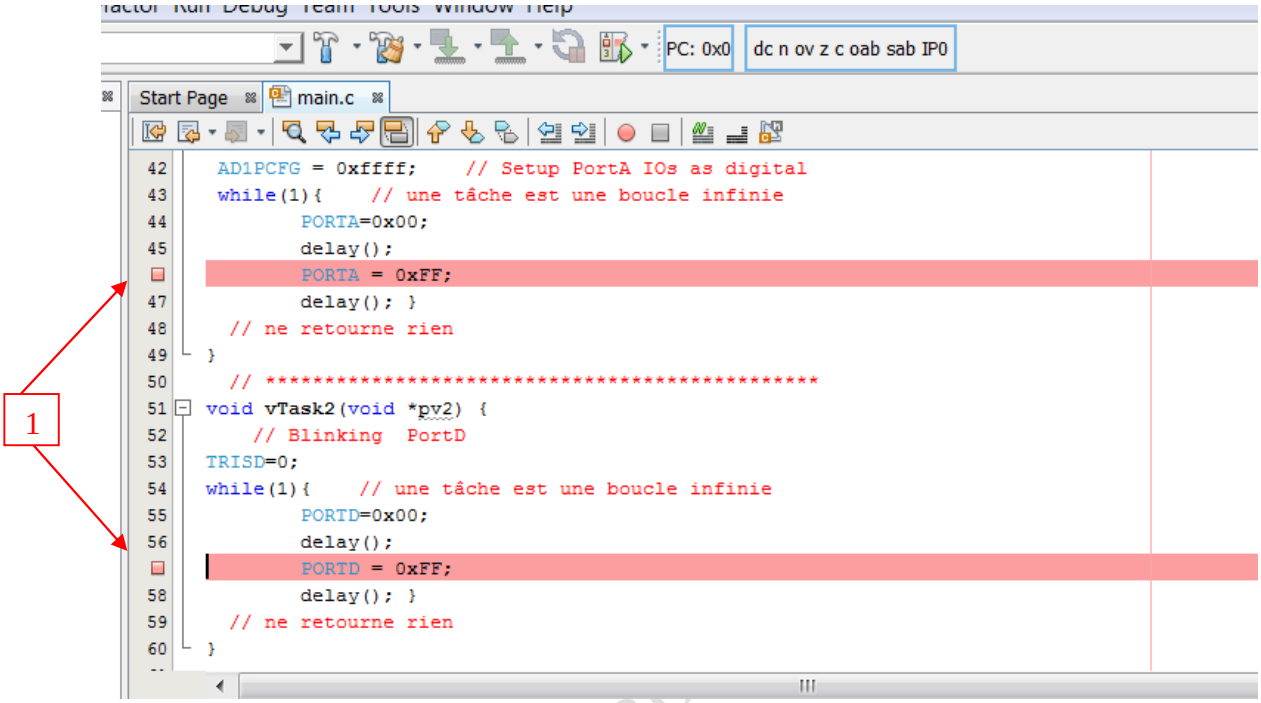
6 : Changez la valeur du PORT en 8000



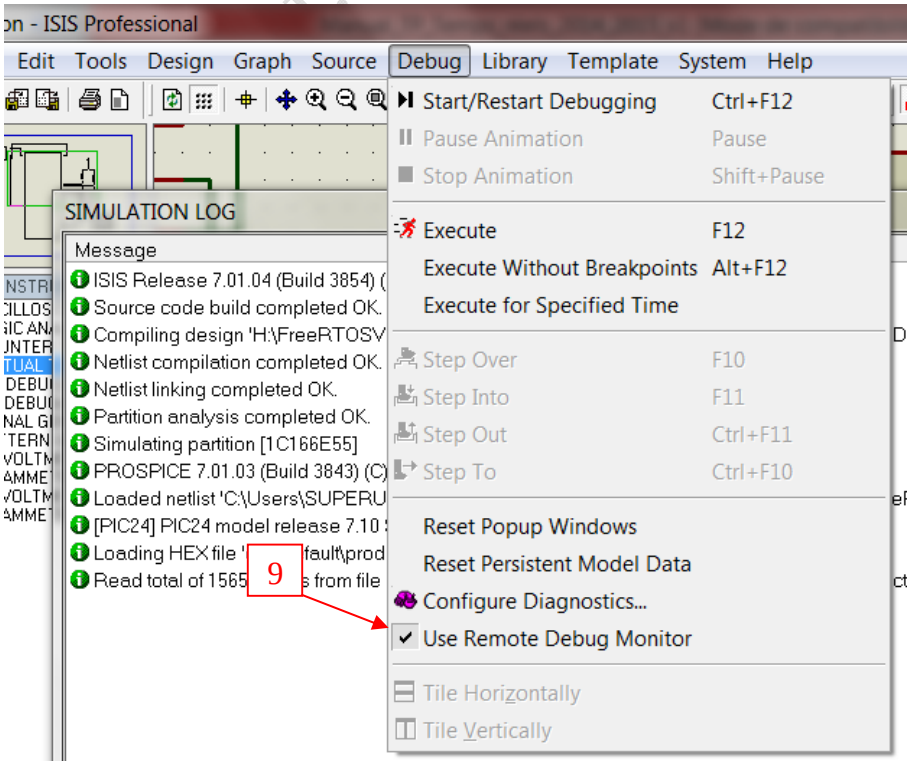
7 : validez par OK

3. Créez une application contenant deux tâches qui font clignoter les deux ports A et D comme dans le TP1 :

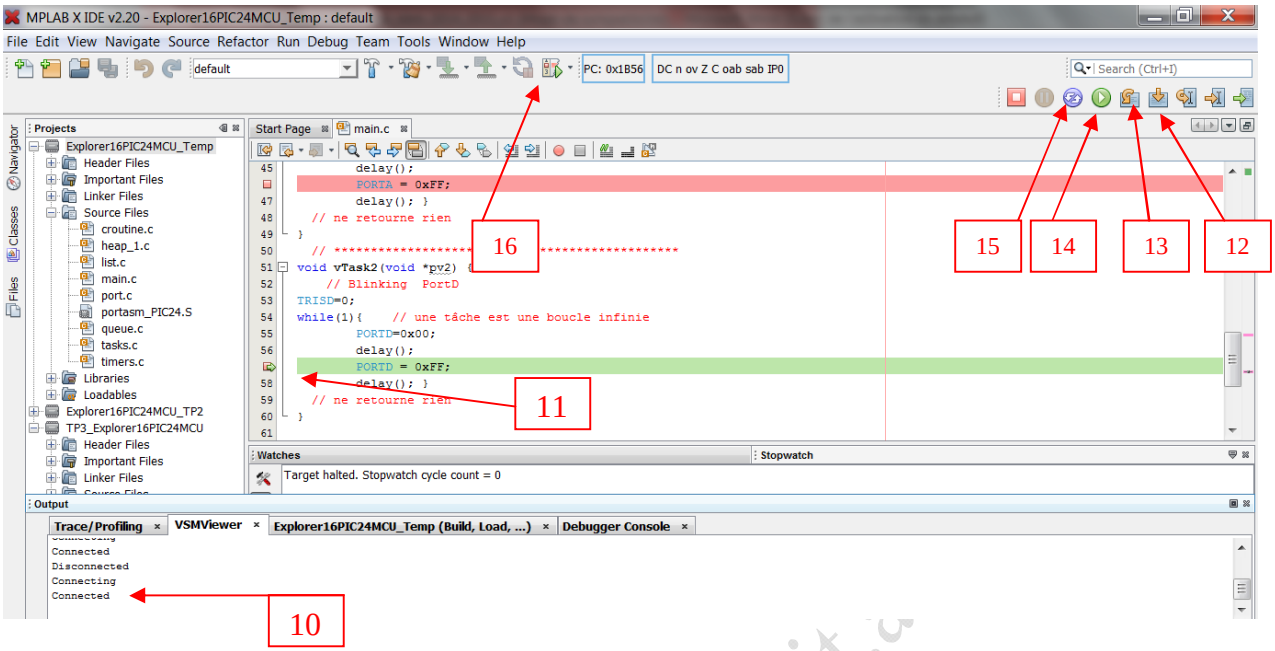
1 : Placez deux points dôarrêts dans le fichier main.c, en cliquant sur le numéro de la ligne où le débugger doit sôarrêter, comme indiqué ci-dessous :



9 : Commencez par démarrez ISIS et ouvrir le fichier design sur lequel vous voulez faire la simulation, Vérifier que ISIS autorise le débogage distant en allant dans son menu Debug et cochez lôoption Use Remote Debug



4. Sur MplabX, lancez le débuggeur (16): après construction, le programme se lance et sôarrête sur le premier point dôarrêt rencontré.



5. Vous devez avoir MPLABx connecté à ISIS (10) (Connected) dans la fenêtre VsmViewer , le curseur Placé sur le premier point dôarrêt (11)

6. Vous pouvez faire évoluer le débuggeur en pas à pas (Step by step), soit pas à pas détaillé(Step In)[12] ou sommaire (Step Over) [13] ou de manière continu jusquôau prochain dôarrêt[14], ou faire un Reset [15] pour relancer lôapplication. A chaque fois que le débuggeur avance et sôarrête sur un point dôarrêt vous pouvez vérifier le résultat sur ISIS.

2.3. Création de plusieurs tâches à partir dôune même fonction tâche :

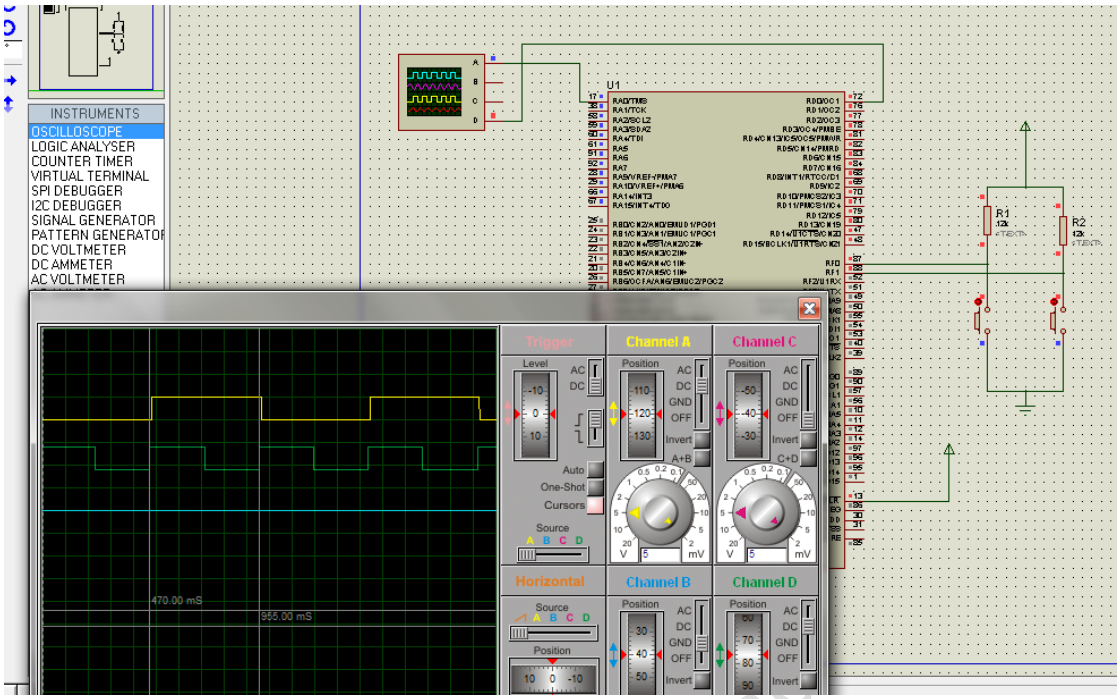
7. Le code suivant utilise une seule fonction tâche vTask( ) pour créer deux tâches nommée Task1 et Task2 différentes utilisant deux paramètres différents. Chaque paramètre passé à une tâche (**par adresse**) est une structure à deux champs (lôadresse du port à faire clignoter et la valeur initiale du décompteur de la temporisation). Ajoutez la définition de la structure **Task_param** dans main.c avant main():

```
typedef struct {//Each parmater is a structure with two fields
   volatile unsigned int *PORT; //PORT's Address To blink
   unsigned long int delay; // frequency=1/2delay
} Task_param;
```

Modifiez le code de la fonction main() comme suit :

```
int main(int argc, char** argv) {

    init_Hard(); //init_Hardware
    Task_param parm1, parm2; //param to pas two task 1 and task2 respectively

    parm1.PORT = &PORTA; //PORTA must be modified (Blinked) --> use address
    parm1.delay = 65535; //value used by counter to create software temporisation
    p1 = &parm1;  //p1 points parm1
    parm2.PORT = &PORTD;
    parm2.delay = 65535 / 2;
    p2 = &parm2;
    // Tasks' creation , notice that param are passed by address
    // and the two Tasks used the same function vTask
    xTaskCreate(vTask, "Task1", 150, p1, 1, &Htask1);
    xTaskCreate(vTask, "Task2", 150, p2, 1, &Htask2);
    vTaskStartScheduler();//Normally never returns
    // we will be here if Scheduler is stopped explicitely or some error occured
    //application are terminated with error
    return (EXIT_SUCCESS);
}
//Functions definition
void init_Hard(void) { //hardware configuration


    TRISA = 0; //PORTA as Outputs
    TRISD = 0; //PORTD as Outputs
}
void delay(unsigned long int delay) {
    unsigned long int i = delay;
    while (i--)
        ;
}


void vTask(void *p) {
    // Blinking  PortA

    while (1) { // a task loops for ever
        *(((Task_param *)p)->PORT) = 0xFFFF; //ON
        delay(((Task_param *)p)->delay); //Notice the casting of p
        *(((Task_param *)p)->PORT) = 0x0000;//OFF
        delay(((Task_param *)p)->delay);
    }
    vTaskDelete(NULL);
}
// *********************************************

void vApplicationIdleHook(void) {
    // si on souhaite que la tâche idle execute un code on le met ici
}
```

8. Construisez et testez le projet sur ISIS en utilisant un design avec un oscilloscope numérique (voir figure ci-dessous) pour montrer que les rapports des fréquences de clignotement des deux ports est de ½.

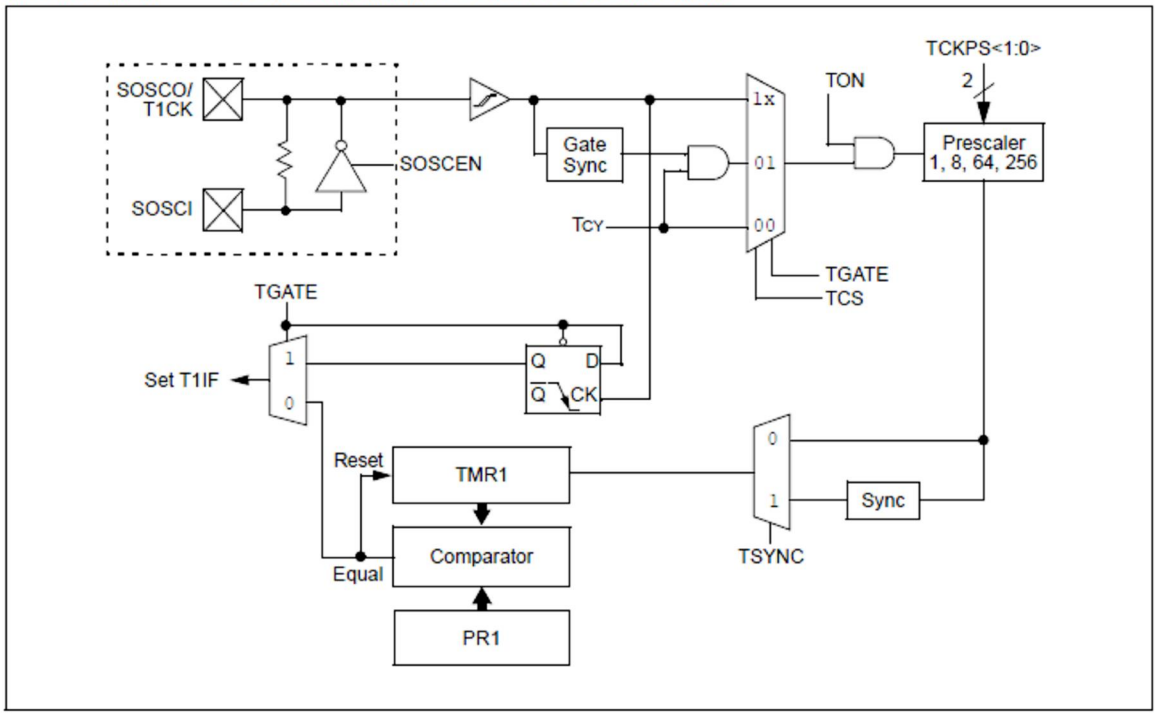


9. Dans freeRTOSconfig.h modifiez la fréquence des ticks dôhorloge pour lui donner la valeur 8Hz : cette valeur est la plus petite valeur entière quôon peut atteindre avec un prescaler du Timer1 réglé à 8 et 8Mhz comme fréquence du quartz: en effet, ce Timer compte jusquôà atteindre la valeur mise dans le registre 16bits PR1 puis retourne à 0 et génère une interruption (un tick dôhorloge) : la valeur à mettre dans ce registre dépend de la fréquence CPU et la fréquence désirée pour le TickRate _Hz (comme le montre le code source extrait du fichier port.c).

```
311
312 /*
313  * Setup a timer for a regular tick.
314  */
315 __attribute__(( weak )) void vApplicationSetupTickTimerInterrupt( void )
316 {
317 const uint32_t ulCompareMatch = ( ( configCPU_CLOCK_HZ / portTIMER_PRESCALE ) / configTICK_RATE_HZ ) - 1;
318
319         /* Prescale of 8. */
320         T1CON = 0;
321         TMR1 = 0;
322
323         PR1 = ( uint16_t ) ulCompareMatch;
324
325         /* Setup timer 1 interrupt priority. */
326         IPC0bits.T1IP = configKERNEL_INTERRUPT_PRIORITY;
327
```

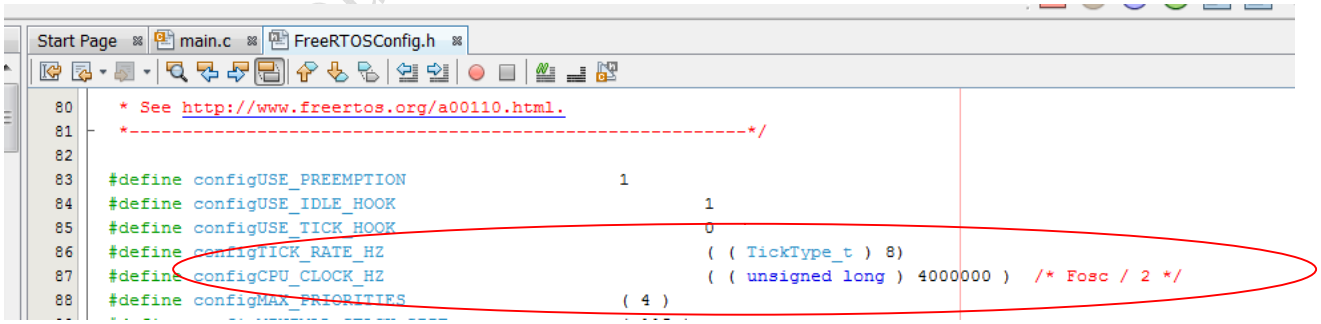
Le diagramme fonctionnel de ce Timer 1 extrait du datasheet du PIC24FJ128GA010 est rappelé ci-dessous

FIGURE 11-1: 16-BIT TIMER1 MODULE BLOCK DIAGRAM



La valeur maximal ulCompareMatch sur 16 bits à charger dans PR1 est de 65535 (ce qui correspond à la plus petite fréquence de TickRate _Hz : cette valeur minimale sera pour une fréquenceCPU=4Mhz=fosc/2 : TickRate_Hz_min=freqCPU/prescaler/(ulCompareMatch_max +1)=4Mhz/8/65536=500kHz/65536=7.26Hz, quôon arrondit à la valeur entière la plus proche soit 8 (car dans port.c le calcule de ulCompareMatch_max se fait par division entière).

Avec un TickRate _Hz réglé à 8Hz les ticks horloge système seront générées une fois tous les 1/8=125ms.

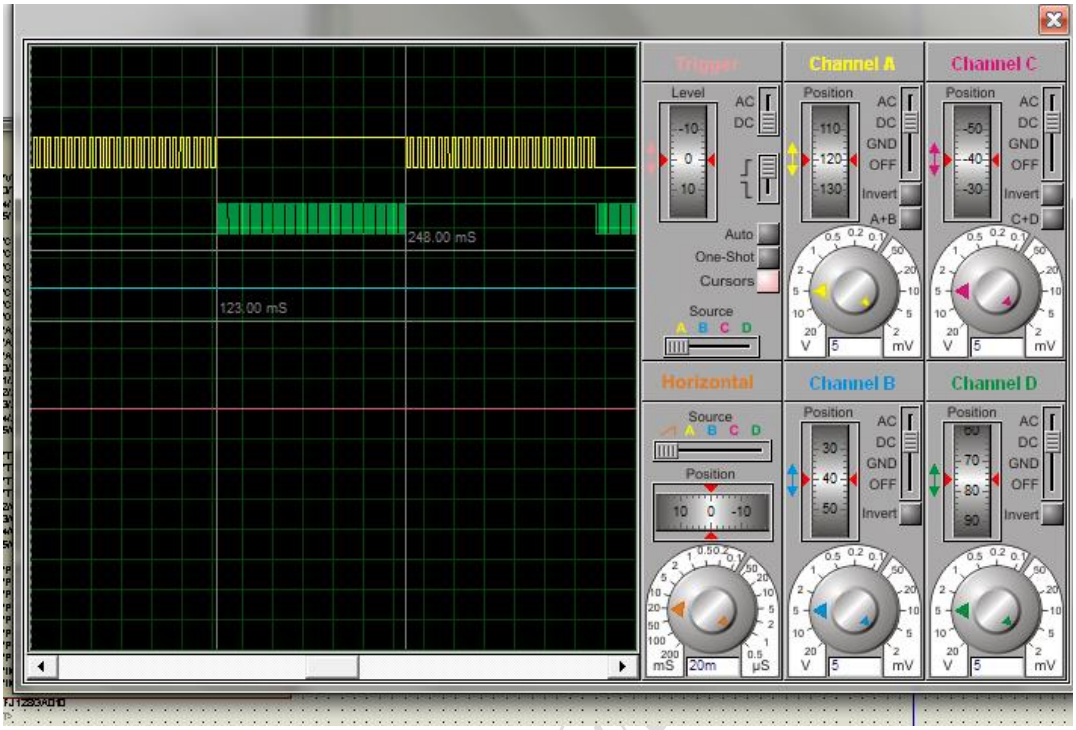


et réduisez les temporisations comme suit:

```
parm1.PORT = &PORTA; //PORTA must be modified (Blinked) --> use address
parm1.delay = 655; //value used by counter to create software temporization
p1 = &parm1;  //p1 points parm1
parm2.PORT = &PORTD;
```

```
parm2.delay = 655 / 2;
```

Reconstruisez le projet et le testez et commentez ce que lôon observe sur lôoscilloscope numérique :



10. Annulez les changements précédents et retournez à la situation initiale (TickRate_Hz=1000Hz). Ajoutez au projet deux autres tâches qui font clignoter le PORTB et PORTC avec des fréquences respectivement 2 fois et 4 fois la fréquence de clignotement du PORTA. Testez le bon fonctionnement de votre application.

NB : pour que le port B fonctionne en mode digital et non pas analogique il faut ajouter lôinstruction suivante lors de sa configuration :

```
AD1PCFG = 0xFFFF;  //voir Datasheet pour plus dôexplication
```

2.4.Blocage des tâches :

11. Supprimez les deux tâches que vous avez ajoutées au point 8. Au lieu dôutiliser une temporisation logicielle (fonction delay( )) on va maintenant utiliser une **temporisation matérielle basée sur les ticks dôhorloge** (1ms par tick puisque le TickRate_Hz=1000Hz), à lôaide de la fonction **bloquante**

void vTaskDelay(portTickType xTicksToDelay);

Pour cela modifier le code de la fonction vTask comme suit :

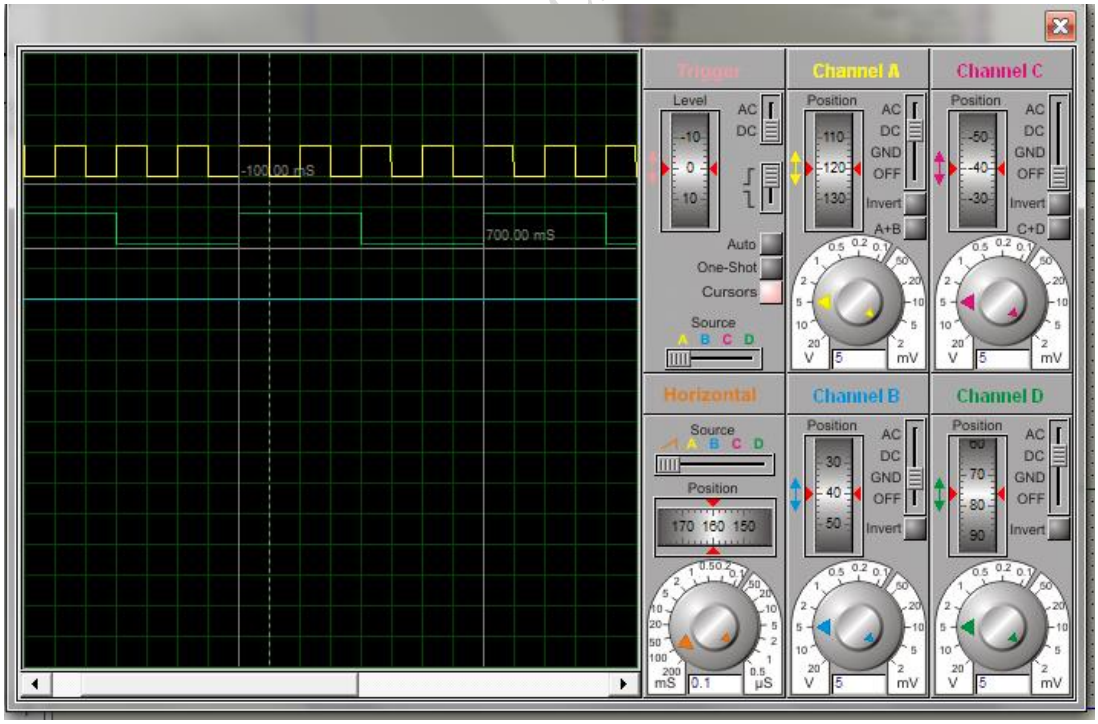
```
67  void vTask(void *p) {
68      // Blinking  PortA
69
70      while (1) { // a task loops for ever
71          *(((Task_param *)p)->PORT) = 0xFFFF; //ON
72          vTaskDelay(((Task_param *)p)->delay); //Notice the casting of p
73          *(((Task_param *)p)->PORT) = 0x0000;//OFF
74          vTaskDelay(((Task_param *)p)->delay);
75      }
76      vTaskDelete(NULL);
77  }
```

Ce ci impose le changement du type du deuxième champ de la structure paramètre à passer aux tâches comme suit (**il désigne maintenant le nombre de ticks dôhorloge à attendre** : voir documentation freertos API sur le fcichier **APIfreertos.chm** fourni parmi les outils sur DVD) :

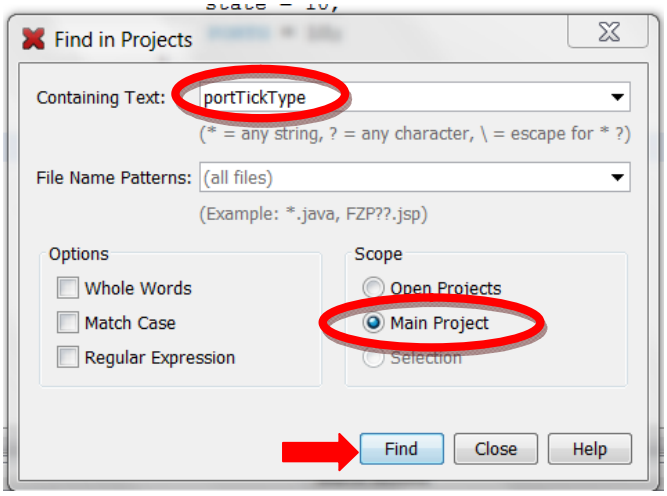
```
26
27  typedef struct {//Each parmater is a structure with two fields
28      volatile unsigned int *PORT; //PORT's Address To blink
29      portTickType delay;  // Nbr of ticks to wait
30  } Task_param;
31
```

```
38      parm1.PORT = &PORTA; //PORTA must be modified (Blinked) --> use address
39     // parm1.delay = 65535; //value used by counter to create software temporisation
40      parm1.delay = 100; //Nbr of ticks to wait
41      p1 = &parm1;  //p1 points parm1
42      parm2.PORT = &PORTD;
43      // parm2.delay = 65535/2; //value used by counter to create software temporisation
44      parm2.delay = 400;//Nbr of ticks to wait
45      p2 = &parm2;
```

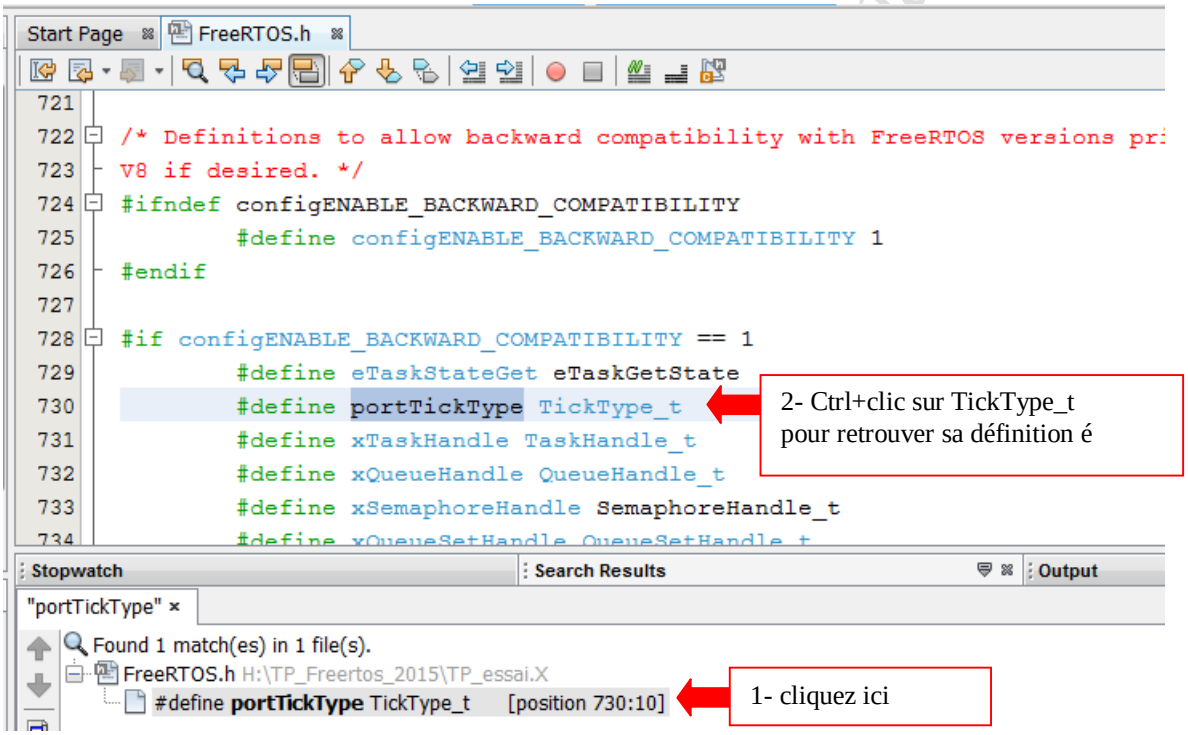
Testez votre projet et vérifiez les fréquences obtenues sur oscilloscope numérique.



12. Dans le menu **Edit** de MPLABX, choisissez lôoption **Find in projects,** cherchez la définition du type portTickType (voir figures suivantes),



puis, on voit que portTickType est définie dans le fichier freertos.h, cliquez sur ce résultat pour ouvrir ce fichier

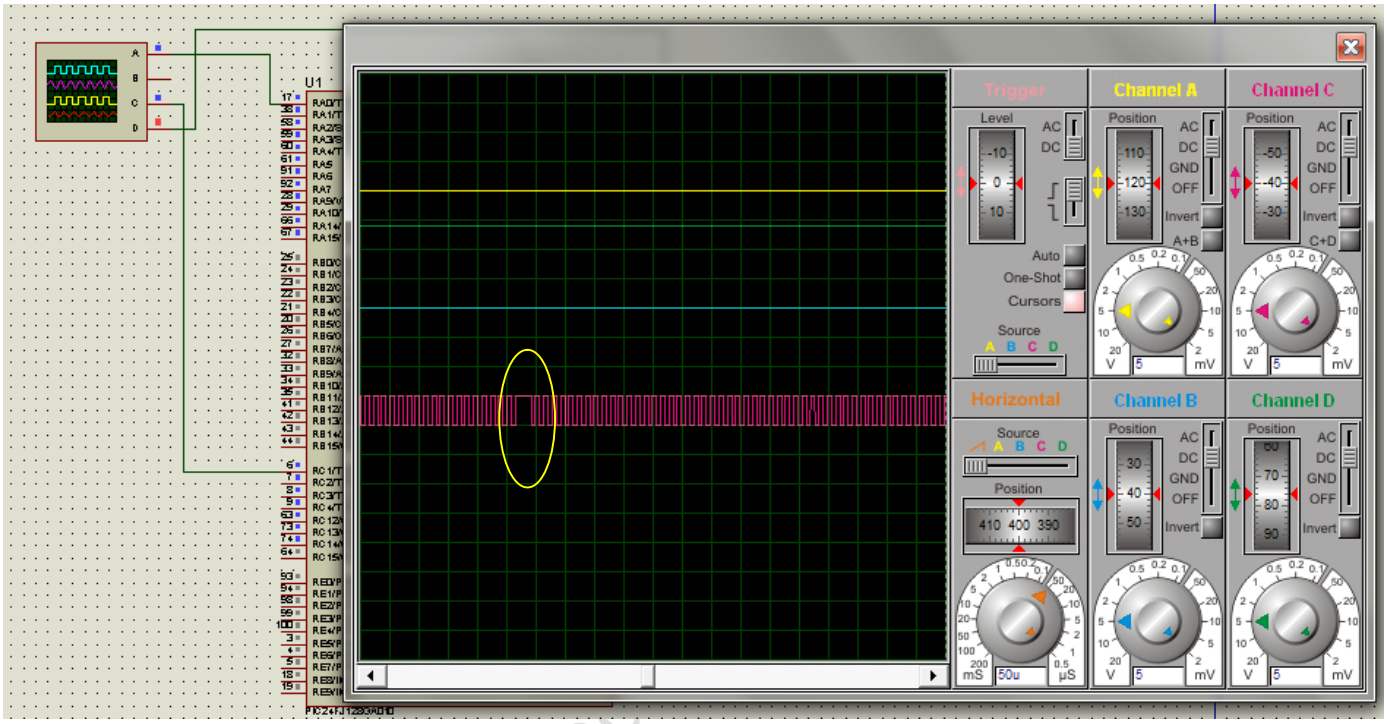


En suivant les macros définitions on trouve que ce type est équivalent à unsigned int (soit un entier signé sur 16 bits car le PIC choisi est dôarchitecture 16bits) : ceci implique alors que la plus grande délai possible avec vTaskDelay() sera 65535 Ticks dôhorloge, soit 65s si un la fréquence des Ticks est de 1000Hz (1 Tick chaque 1 ms), comme choisi dans freertosconfig.h via la macro définition **TickRate_Hz.**

13. La plus part de temps les deux tâches se retrouvent toutes les deux dans un état bloquées : et la seule tâche qui sera exécuter sera la tache idle ; pour le prouver mettez un code dans la fonction vApplicationIdleHook() qui fait basculer RC1 (Nôoubliez pas de configurer le PORTC en sortie dans init_Hard( )):

```
85
86  void vApplicationIdleHook(void) {
87      if (_RC1==0) _RC1=1;
88      else _RC1=0;
89  }
90
```

NB : on nôappelle jamais vTaskDelay() à lôintérieur de la tâche idle.



Notez le clignotement des ports mais aussi le clignotement de RC1 (réglez la base de temps sur 50us sur lôoscilloscope).

14. Effacer le code dans vApplicationIdleHook(), ajoutez une tâche basée sur une fonction vTask3 ( ) qui teste lôétat du bouton BP0 et BP1 reliés respectivement à RF0 à RF1 : si BP0 bouton est appuyé Task3 suspend Task1(qui fait clignoter le PORTA), et si BP1 est appuyé Task3 reprend Task1 : utilisez pour cela les fonctions de lôAPI freertos : void vTaskSuspend(xTaskHandle pxTaskToSuspend);

void xTaskResume(xTaskHandle pxTaskToResume);

Une solution possible pourrait être impléentée comme suit:

```
void vTask3(void *pv2) {
    short Task1suspended = 0; //Task1 non suspendue
    while (1) {
        if ((_RF0 == 0)&&(Task1suspended == 0)) {
            vTaskSuspend(H1);
            Task1suspended = 1;//Task1 est suspendue
        } else if ((_RF1 == 0)&&(Task1suspended == 1)) {
            vTaskResume(H1);
            Task1suspended = 0;
        }

    }
    vTaskDelete(NULL);
}
```

15. Reprendre la question 14 pour suspendre les deux tâches Task1 et Task2 en utilisant les fonctions de lôAPI freertos:

```
void vTaskSuspendAll();
void xTaskResumeAll();
```

Exercice de synthèse:

On souhaite développer une application qui génère des signaux périodiques sur les pins RB0 à RB3 de manière indépendante et avec des fréquences différentes et des rapports cycliques, comme suit :

RB0 : une fréquence de 100Hz, rapport cyclique=0.5 : soit Ton=5ms et Toff=5ms.

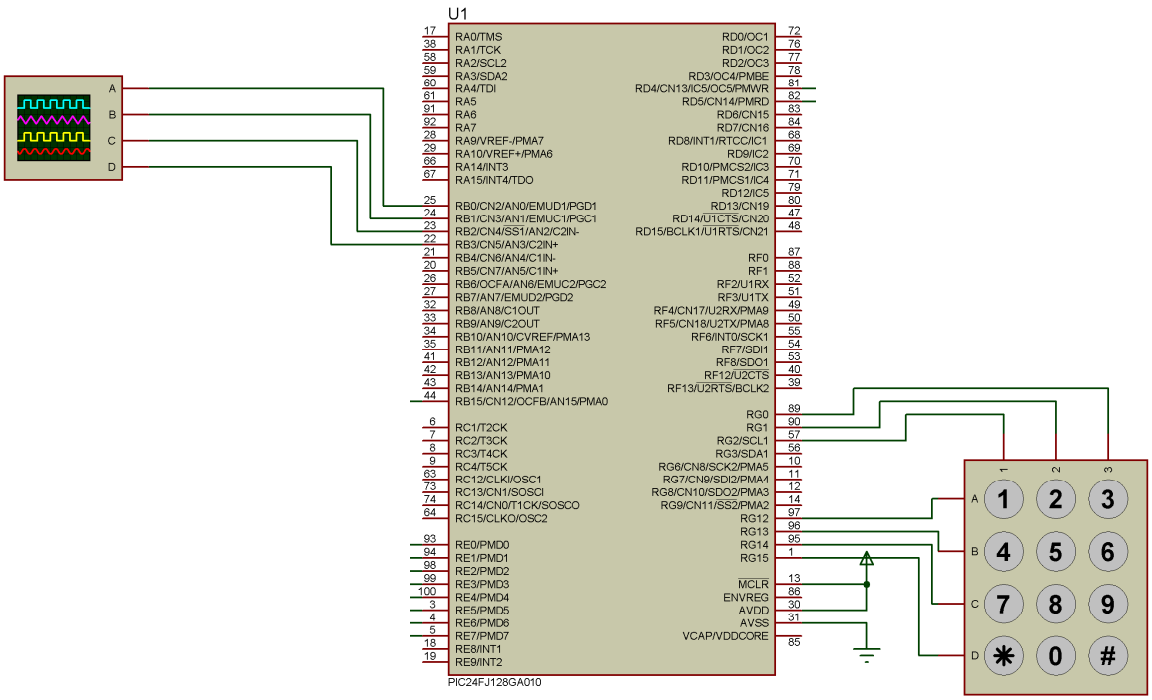
RB1 : une fréquence de 10Hz, rapport cyclique=0.4 : soit Ton=40ms et Toff=60ms.

RB2 : une fréquence de 1Hz, rapport cyclique=0.2 : soit Ton=200ms et Toff=800ms.

RB3 : une fréquence de 5Hz, rapport cyclique=0.1 : soit é

En plus, chaque signal peut être activé (généré) ou désactivé (sortie reste à 0), à lôaide dôune touche du clavier 0 pour RB0, 1 pour RB1 ..etc.

a- Réalisez le Design suivant sur ISIS:



b- Utilisez la programmation multitâche basée sur freertos pour développer cette application. Les tâches seront créées en se basant sur la même fonction tâche qui fera clignoter une sortie avec une certaine fréquence et un certain rapport cyclique fournis en paramètre. Les tâches seront donc paramétrée avec des paramètres de

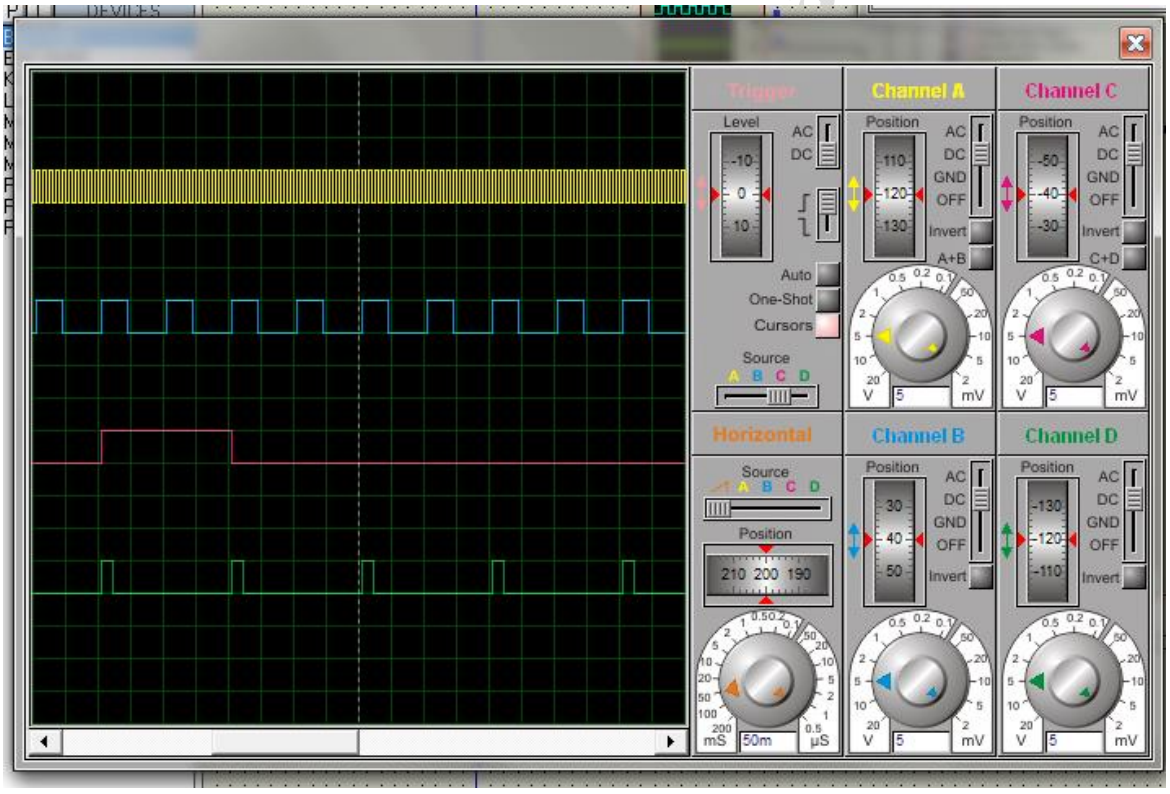
type structure contenant trois informations : Numéro de pin (0 pour RB0, 1 pour RB1, ..etc), Ton (nombre de ticks horloge pour lesquels la pin sera mise à 1) Toff(nombre de ticks pour lesquels la pin sera mise à 0) ; cette structure sera définie comme suit :

```
typedef struct {//Each parmater is a structure with tree fields
    short PinNum; //PinNum
    portTickType Ton; // Number of ticks during which the pin is maintained at 1
    portTickType Toff; // Number of ticks  during which the pin is maintained at 0
} Task_param;
```

La fonction tâche à créer doit tester ((Task_parm *)p)->PinNum et en fonction de sa valeur (0, 1, 2 ou 3) elle doit agir sur _RB0,_RB1, _RB2, _RB3.

Lôapplication utilisera donc 4 tâches basées sur la même fonction tâche, avec le même niveau de priorité.

Testez votre projet sur ISIS pour voir le résultat. Les signaux générés observés sur lôoscilloscope numérique sur ISIS doivent en principe respecter les fréquences et rapports cycliques exigés, comme le montre la figure ci-dessous :



c- Il faut utiliser maintenant le clavier pour suspendre(Suspend) ou reprendre (Resume) lôune ou plusieurs des 4 tâches créés précédemment. Ajoutez une fonction tâche vControlTask( ) qui gère le clavier pour détecter la touche appuyée et si cette touche est la touche 0 et la tâche qui fait clignoter RB0 nôest pas suspendue elle sera lors suspendue , et sinon elle sera reprise, de même pour les autres tâches avec les touches 1, 2 et 3. Cette fonction tâche fera appel à la fonction ucPollKeyBoard( ) qui scrute le clavier et retourne un

caractère correspondant à la touche appuyée ou óEô si aucune touche nôest appuyée ou plusieurs touche ont été appuyée au même temps :

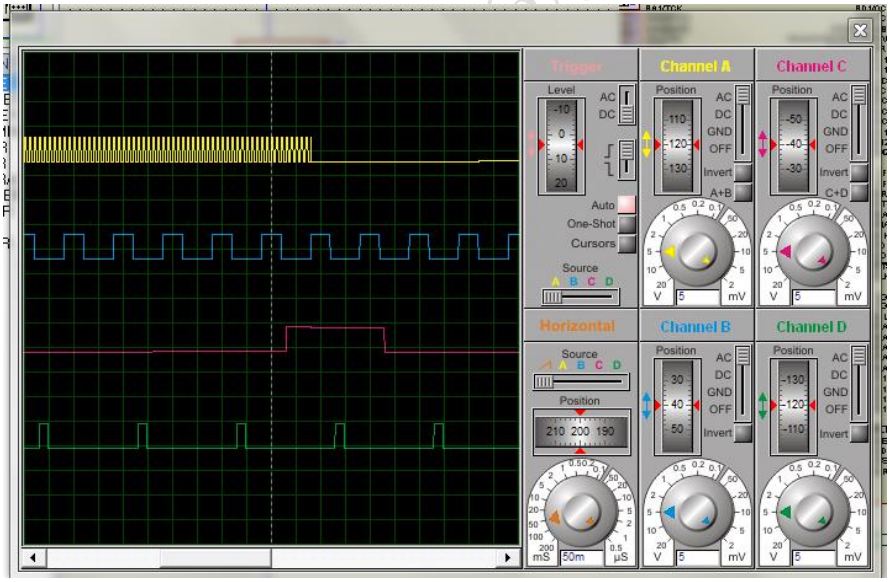
```
unsigned char ucPollKeyboard(void) {
   PORTG = 0x0001; //KeyBoard's column 3 is on
   switch (PORTG & 0xF000) { //masks all bits except RG12 to RG15
      case 0x1000: //RG12=1 Key 3 is pressed
      { while ((PORTG & 0x1000)); //wait until key released
         return '3'; //return the character ó3 ó
      }
      case 0x2000: //RG13=1 Key 6 is pressed
      {  while ((PORTG &0x2000));
         return '6';   }
      case 0x4000: //RG14=1 Key 9 is pressed
      {  while ((PORTG & 0x4000));
         return '9';  }
      case 0x8000: //RG9=1 Key # is pressed
      {  while ((PORTG & 0x8000));
         return '#';   }
   }
   PORTG = 0x0002; //KeyBoard's column 2 is on
   switch (PORTG & 0xF000) {
      case 0x1000:
      {  while ((PORTG & 0x1000));   return '2';  }
      case 0x2000:
      {   while ((PORTG & 0x2000));   return '5';  }
      case 0x4000:
      {  while ((PORTG & 0x4000));   return '8'; }
      case 0x8000:
      {  while ((PORTG & 0x8000));  return '0';  }
   }
     PORTG = 0x0004; //KeyBoard's column 1 is on
   switch (PORTG & 0xF000) {
      case 0x1000:
      {  while ((PORTG & 0x1000));   return '1';  }
      case 0x2000:
      {   while ((PORTG & 0x2000));   return '4';  }
      case 0x4000:
      {  while ((PORTG & 0x4000));   return '7'; }
      case 0x8000:
      {  while ((PORTG & 0x8000));  return '*';  }
```

```
    }
return 'E'; //error more than one key are pressed or any key pressed
}
```

Complétez la fonction tâche de contrôle suivante, et testez votre projet pour vérifier son bon fonctionnement :

```
/*********************************************************/
/*Control function : suspends or resumes blinking of RB0 to RB3 */
void vControlTask(void *pvParameters) {
unsigned char ucTask0_suspended=0, ucTask1_suspended=0,ucTask2_suspended=0,ucTask3_suspended=0;
    while (1) {
        switch (ucPollKeyboard()) {
            case '4': case '5': case '6': case '7': case '8': case '9': case 'E':    break;
            case '0':
                if (ucTask0_suspended) { //if task0 is suppended
                    vTaskResume(Htask0); //resume it
                    ucTask0_suspended = 0;//Update its status
                } else { //if task0 is not suspended
                    vTaskSuspend(Htask0); //suspend it
                    ucTask0_suspended = 1; //Update its status
                    _RB0=0; //hold RB0 to 0 while the task is suspended
                }
                break;
            case '1': if (ucTask1_suspended) {    vTaskResume(Htask1);  ucTask1_suspended = 0;
```

Ci-dessous une capture dôécran qui montre que la tâche 0 a été suspendue après avoir appuyé sur la touche 0.



d- Complétez votre projet de telle sorte que si on appuie sur la touche # toutes les tâches sont suspendue et si on appuie sur la touche * toutes les tâches seront reprises. Testez le bon fonctionnement de votre projet.

NB : après avoir suspendu toutes les tâches par un appel à vSuspendAll() remarquez quôon ne peut pas reprendre une seule tâche en appuyant sur la touche correspondante : justifiez ce comportement en étudiant les indications concernant les fonctions de lôAPI freertos (fichier freertos API reference.chm fourni).