Formation Automatique et Informatique Industrielle

Master 1 S2

Matière : Systèmes Embarqués et Systèmes Temps Réel SE-STR

Par : ATOUI Hamza

Plan du cours et TD-TP

- Système mono-tâche (clignotement d'une LED).
- Positionnement du problème multitâches (Plusieurs tâches de la même nature).
- La solution en pseudo-multitâches.
- Positionnement du problème multitâches (Plusieurs tâches avec différentes natures).
- Problème de la zone STACK de MID-RANGE.
- Présentation de noyau HBM-RTOS (LITE VERSION).
- Implémentation du noyau.
- Exercice.

Système mono-tâche (clignotement d'une LED)

- Dans cette partie de TP, on va réaliser le programme et la maquette de clignotement d'une LED connectée au RB0 par une fréquence de 2Hz
- Solution 1 : La temporisation se fait par programmation.
- Solution 2 : La temporisation se fait par TIMER0.

Positionnement du problème multitâches (Plusieurs tâches de la même nature)

- Imaginez mes chers étudiants que j'ai ajouté une autre LED au RB1 clignote par une fréquence de 1Hz. La solution tout simplement est de réaliser la séquence suivante :
  1. **LED1(ON);**
  2. **LED2(ON);**
  3. **DELAY(250ms);**
  4. **LED1(OFF);**
  5. **DELAY(250ms);**
  6. **LED1(ON);**
  7. **LED2(OFF);**
  8. **DELAY(250ms);**
  9. **LED1(OFF);**
  10. **DELAY(250ms);**
  11. **JUMP TO 1**

Positionnement du problème multitâches (Plusieurs tâches de la même nature)

- Mais, comment faire si les deux fréquences ne sont pas multiple ???
- Imaginez j'ai ajouté d'autres LED !!!

La solution en pseudo-multitâches

- La solution est de faire une **boucle** à **délai constant** et d'associer un **compteur** pour chaque tâche.
- Pseudo-code de la solution :

```
// LED1 clignote en 2Hz (500ms)
// LED2 clignote en 3Hz (334ms)
// LED3 clignote en 1Hz (1000ms)

// Initialisation des compteurs
CNT1 ← 250; // demi-période de 500ms
CNT2 ← 167; // demi-période de 334ms
CNT3 ← 500; // demi-période de 1000ms

// Initialisation des LED
LED1(OFF);
LED2(OFF);
LED3(OFF);
```

```
Tant que (1)
        Si (CNT1 = 0)
                INV(LED1);
                CNT1 ← 250;
        Fin si
        Si (CNT1 = 0)
                INV(LED2);
                CNT1 ← 167;
        Fin si
        Si (CNT1 = 0)
                INV(LED3);
                CNT1 ← 500;
        Fin si
        CNT1 ← CNT1 - 1;
        CNT2 ← CNT2 - 1;
        CNT3 ← CNT3 - 1;
        DELAY(1ms);
Fin tant que
```

La solution en pseudo-multitâches

Réaliser cette solution par le PIC16F84A.

Positionnement du problème multitâches (Plusieurs tâches avec différentes natures)

- Maintenant, on ajoute une autre tâche qui fait le test d'un bouton connecté à RA0; est ce qu'il est appuyé puis relâché, si le cas on va inverser l'état de la LED à RB4.
- Vous remarquez clairement que le test est bloquant (Si le bouton est appuyé la tâche attendra son relâchement infiniment), donc cette dernière ne donne pas la main aux autres tâches pour exécuter.

  Comment faire pour résoudre ce problème ???

Problème de la zone STACK de MID-RANGE

- D’après le cours qu’on a fait sur l’architecture de MID-RANGE, que ce dernier possède une zone STACK de 8 niveaux pour empiler et dépiler les adresses de retour au moment d’appel d’un sous-programme ou le traitement d’une interruption.
- En plus, cette zone reste inaccessible par le programmeur (on ne peut pas lire ni modifier).
- **Cette limite rendre notre µP (MID-RANGE) incapable de prendre un noyau préemptif et ce limite au noyau coopératif**.

Problème de la zone STACK de MID-RANGE

- Donc, pour résoudre le problème de multitâches avec différentes natures et la limite de la zone STACK de MID-RANGE, nous sommes devant le développement d'un noyau coopératif par les politiques suivantes:
  - FCFS.
  - PB coopératif.
- Dans ce qui suit on va présenter mon noyau **HBM-RTOS (LITE VERSION)**.

Présentation de noyau HBM-RTOS (LITE VERSION)

- HBM-RTOS est noyau (**KERNEL**) regroupe plusieurs services pour assurer le passage d'une tâche à l'autre sans problème.
- Le service le plus important d'un noyau est le **SCHEDULER** responsable de la cohabitation des différentes tâches pendant leur exécution.
- Notre noyau contrôle les ressources et permet leur utilisation de façon sûre et efficace à travers les services.

Présentation de noyau HBM-RTOS (LITE VERSION)

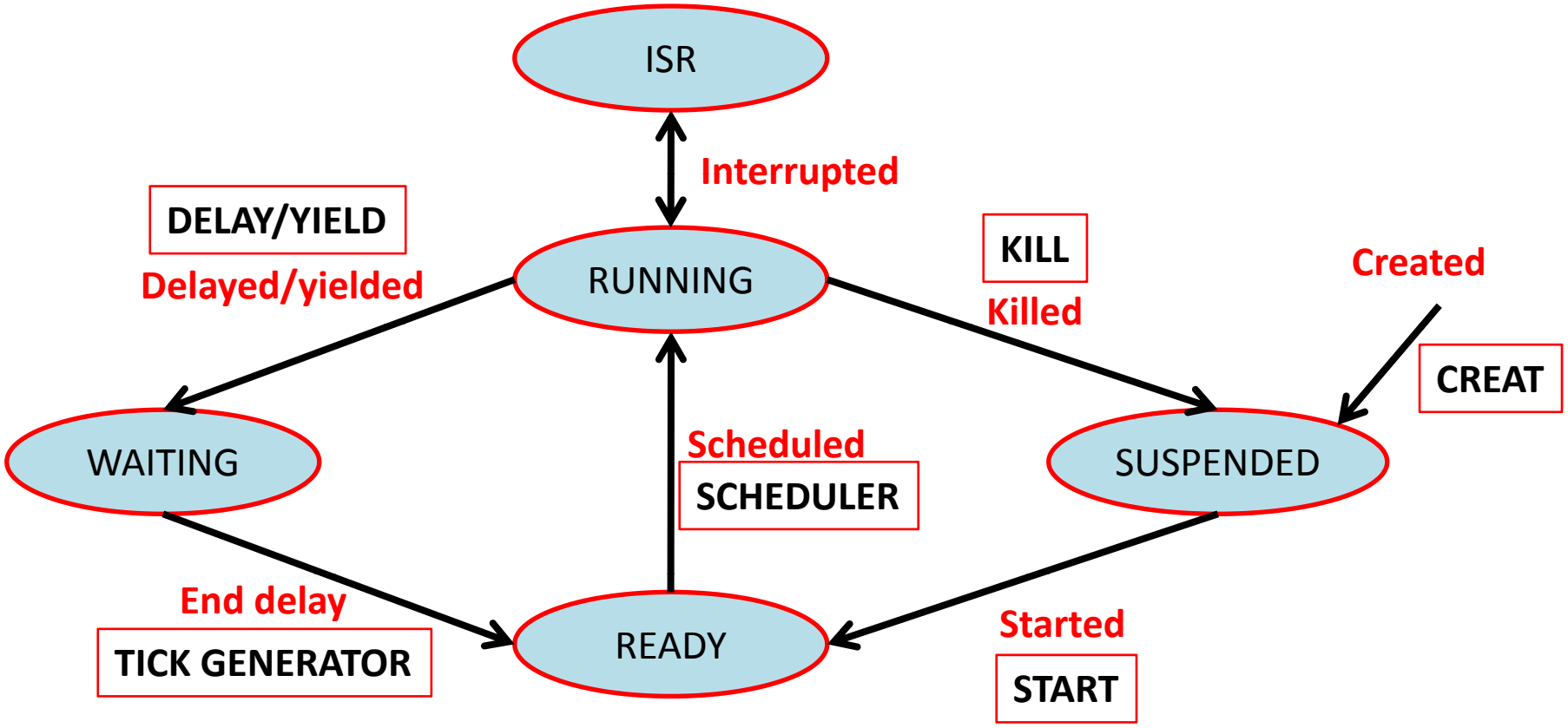
- Cette Multiprogrammation permet aux développeurs de créer des tâches sans ce soucier de savoir s'il existe d'autres tâches dans le µC. Chacun a l'impression que le µC lui est dédié.
- Lorsque le noyau joue parfaitement son rôle de chef d'orchestre (le fonctionnement en parallèle des tâches incombent au noyau seul). On dit que le noyau est PREEMPTIF, si le noyau n'est pas capable de faire une telle chose, alors c'est aux tâches de rendre la main au noyau de temps en temps par ses services. On dit que le noyau est COOPERATIF.

Présentation de noyau HBM-RTOS (LITE VERSION)

- Mon noyau se base sur les travaux de «**GEOFF DRAKE** » par son noyau « **PIC'XE** » réalisé le 16/05/2005 sous la licence GPL (version 1.0).
- A cause du problème de l'impossibilité de gestion de la pile (STACK ZONE) dans le µP MID-RANGE chez Microchip, on est devant la réalisation d'un noyau **COOPERATIF** suit la politique **PRIORIRY BASED**.

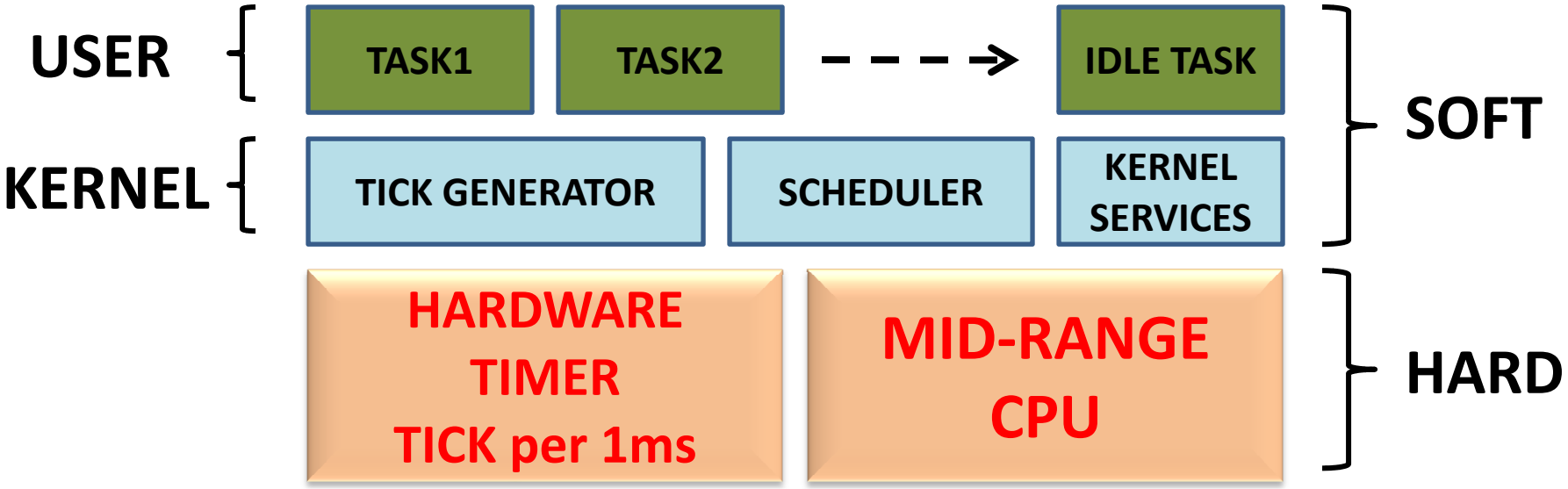
Présentation de noyau HBM-RTOS (LITE VERSION)

- La version LITE de mon noyau, ses tâches utilisent la représentation d'état suivante :
- **En rouge présente l'événement.**
- **En noir encadré présente le service responsable sur le déclanchement de l'événement.**



Présentation de noyau HBM-RTOS (LITE VERSION)

- La synoptique générale de la structure de système :



Présentation de noyau HBM-RTOS (LITE VERSION)

- Services de noyau :
- **SCHEDULER** : responsable sur le chargement de la tâche de la plus haute priorité prête à exécuter.
- **TICK GENERATOR** : responsable sur la décrémentation de délai des tâches à l'état WAITING, si le délai est écoulé la tâche passe vers l'état READY.
- **START** : responsable sur le passage de la tâche vers l'état READY.
- **KILL** : responsable sur le passage de la tâche vers l'état SUSPENDED puis l'appel de SCHEDULER.

Présentation de noyau HBM-RTOS (LITE VERSION)

- **CREAT** : responsable sur la création de la tâche (chargement de **TCB -Task Control Block-** par les paramètres de la tâche) avec la tâche passe à l'état SUSPENDED.
- **DELAY** : responsable sur le passage de la tâche vers l'état WAITING pendant un délai puis l'appel de SCHEDULER.
- **YIELD** : responsable sur le passage de la tâche vers l'état WAITING volontairement pour laisser la main au autre tâches par l'appel de SCHEDULER.

Implémentation du noyau

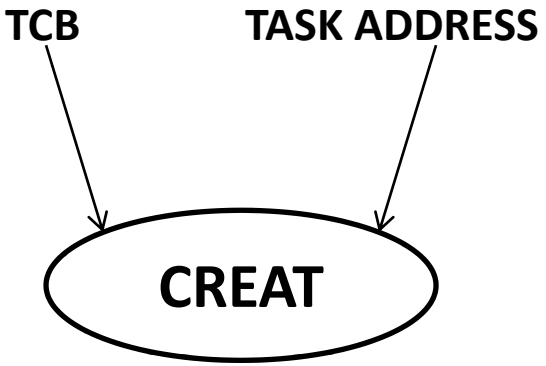
- Avant de passer vers l'implantation de noyau, il faut définir la structure de contrôle TCB.
- Chaque tache dans le système est définie par :
  - Son **état (byte)**.
  - Son **PC (word)**.
  - Son **compteur de délai (word)**.
- En total 5 bytes pour la structure TCB.
- **TCB.STATE, TCB.PC et TCB.DELAY**

Implémentation du noyau

- Le service **CREAT** : responsable sur la création de la tâche (chargement de **TCB -Task Control Block-** par les paramètres de la tâche) avec la tâche passe à l'état SUSPENDED.

Pseudo-code :

1- Désactiver les interruptions (GIE ← 0)
2- TCB.STATE ← 'S'
3- TCB.PC ← TASK ADDRESS
4- TCB.DELAY ← 0
5- Activer les interruptions (GIE ← 1)

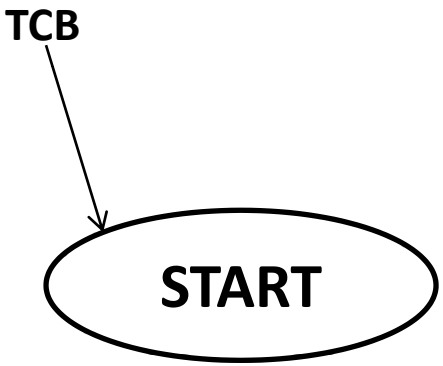


Implémentation du noyau

- Le service **START** : responsable sur le passage de la tâche vers l'état READY.

Pseudo-code :

1- Désactiver les interruptions (GIE ← 0)
2- if (TCB.DELAY = 0) then
    TCB.STATE ← 'R'
  end if
3- Activer les interruptions (GIE ← 1)

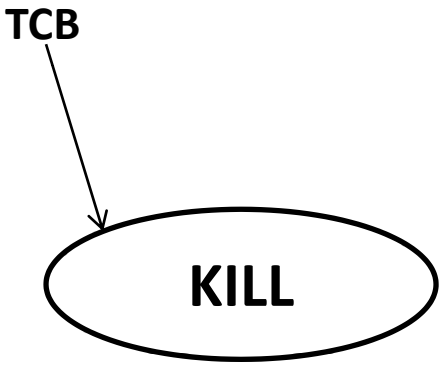


Implémentation du noyau

- Le service **KILL** : responsable sur le passage de la tâche vers l'état SUSPENDED puis l'appel de SCHEDULER.

Pseudo-code :

1- Désactiver les interruptions (GIE ← 0)
2- TCB.STATE ← 'S'
3- TCB.DELAY ← 0
4- Activer les interruptions (GIE ← 1)
5- Appel de service SCHEDULER

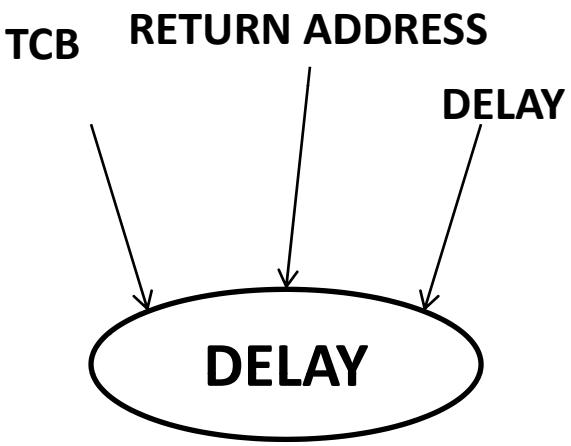


Implémentation du noyau

- Le service **DELAY** : responsable sur le passage de la tâche vers l'état WAITING pendant un délai puis l'appel de SCHEDULER.

Pseudo-code :

1- Désactiver les interruptions (GIE ← 0)
2- TCB.STATE ← 'W'
3- TCB.PC ← RETURN ADDRESS
4- TCB.DELAY ← DELAY
5- Activer les interruptions (GIE ← 1)
6- Appel de service SCHEDULER



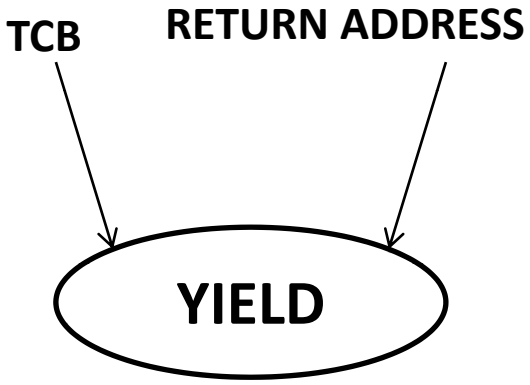
RETURN ADDRESS : est l'adresse de l'instruction après l'appel de service

Implémentation du noyau

- Le service **YIELD** : responsable sur le passage de la tâche vers l'état WAITING volontairement pour laisser la main au autre tâches par l'appel de SCHEDULER.

Pseudo-code :

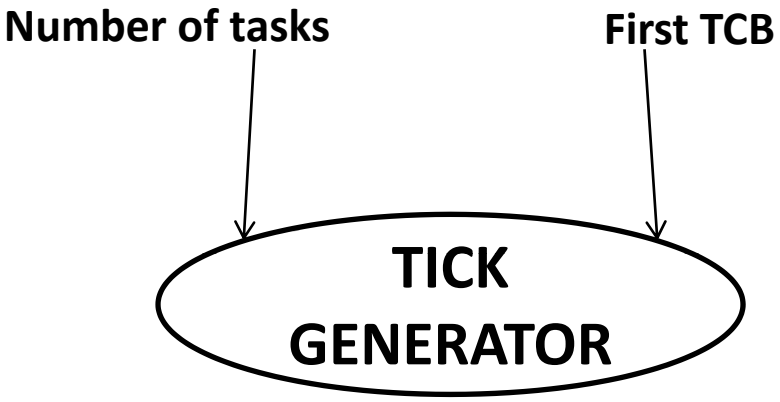
1- Désactiver les interruptions (GIE ← 0)
2- If (NT < Number of tasks-1) then
    TCB.STATE ← 'W'
  Else
    TCB.STATE ← 'R'
  End if
3- TCB.PC ← RETURN ADDRESS
4- Activer les interruptions (GIE ← 1)
5- Appel de service SCHEDULER



RETURN ADDRESS : est l'adresse de l'instruction après l'appel de service.
NT : numéro de la tâche ou son priorité.

Implémentation du noyau

- Le service **TICK GENERATOR** : responsable sur la décrémentation de délai des tâches à l'état WAITING, si le délai est écoulé la tâche passe vers l'état READY (ce service est invoqué par le traitement de l'interruption du TIMER).



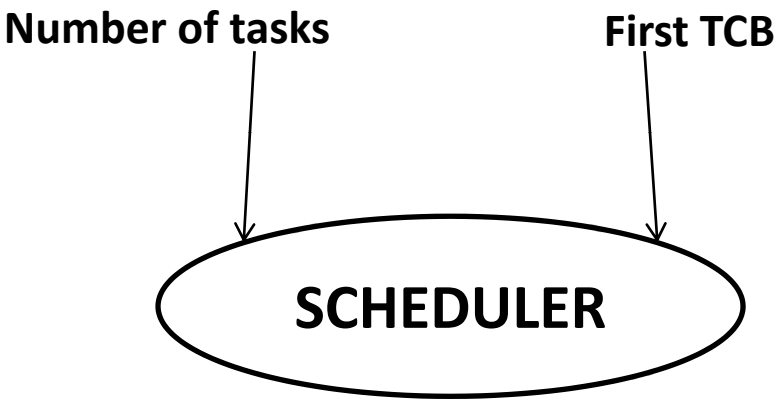
Implémentation du noyau

Pseudo-code :

```
For k=0:Number of tasks-1
        if (TCB[k].STATE = 'W') then
                if (TCB[k].DELAY > 0) then
                        TCB[k].DELAY ← TCB[k].DELAY - 1
                else
                        TCB[k].STATE ← 'R'
                end if
        end if
End for
```

Implémentation du noyau

- Le service **SCHEDULER** : responsable sur la chargement de la tâche de la plus haute priorité prête à exécuter.



Implémentation du noyau

Pseudo-code :

Désactiver les interruptions (GIE ← 0)
For NT=0:Number of tasks-1
    if (TCB[NT].STATE = 'R') then
        TCB[NT].STATE ← 'E'
        Activer les interruptions (GIE ← 1)
        PC ← TCB[NT].PC
    end if
End for

N'oublier pas que au moment de chargement de registre PC la boucle For se casse, par ce que l'écriture dans le registre PC se traduit par un saut (GOTO).

Implémentation du noyau

- Dans un premier temps, on demande de programmer tous les services en assembleur de MID-RANGE.
- Le TICK_SYSTEM = 1ms (assurer par le TIMER0 sous interruption).

Implémentation du noyau

- Dans un deuxième temps, d'exploiter les services du noyau pour réaliser le système multitâches suivant:
- **La tâche 1** : clignote la LED1 (RB0) chaque 700 ms.
- **La tâche 2** : clignote la LED2 (RB1) chaque 200 ms.
- **La tâche 3** : clignote la LED3 (RB2) chaque 1000 ms.
- Et sans oublier **la tâche IDLE** qui boucle infiniment sur l'incrémentation d'un compteur et passe volontairement le CPU au autre tâches par le biais de service **YIELD**.
- Avec Task1 > Task2 > Task3 > IDLE.

Exercice

- Exercice : implanter le système multitâches suivant par le biais de noyau HBM-RTOS :
- **La tâche 1** : clignote la LED1 (RB0) chaque 300 ms, 10 fois et se termine.
- **La tâche 2** : clignote la LED2 (RB1) chaque 400 ms.
- **La tâche 3** : fait le test de bouton poussoir (RA0) est ce qu'il est appuyé puis relâché pour inverser l'état de la LED3 (RB2).
- Et sans oublier **la tâche IDLE** qui boucle infiniment sur l'incrémentation d'un compteur et passe volontairement le CPU au autre taches par le biais de service **YIELD**.
- Avec Task1 > Task2 > Task3 > IDLE.