

# TP de Temps-Réel

*Départements EII & CDTI*

*Année 2019-2020*

*Jean-François NEZAN*

*Olivier DEFORGES*

*Florian LEMARCHANT*

*Glenn HERROU*

*Nicolas SOURBIER*

# TP1 : prise en main CMSIS sur NUCLEO

## 1. PRESENTATION GENERALE

Vous avez déjà programmé une carte Nucleo en utilisant (ou non !) les environnements SMT32CubeMX et Keil. CubeMX permet de générer un projet Keil utilisant les primitives « FreeRTOS ». Lorsque vous générerez le code, vous vous rendrez compte que les fonctions correspondent au RTOS CMSIS plutôt que FreeRTOS. CMSIS est une surcouche à FreeRTOS développée par Keil. Vous trouverez la documentation de CMSIS en ligne ici :

<http://www.keil.com/pack/doc/CMSIS/General/html/index.html>

Le noyau CMSIS supporte les fonctionnalités suivantes (toutes ne seront pas utilisés en TP) :

- les tâches,
- La gestion du temps et synchronisations,
- les signaux (Signal Events), sémaphores et Mutex,
- les messages par boîte aux lettres (Message Queue, Mail Queue)
- les mémoires partagées (Memory Pool)

## 2. Présentation du matériel

Vous devez utiliser la carte Nucleo, un clavier matriciel et un écran Oled connectés de la manière suivante :

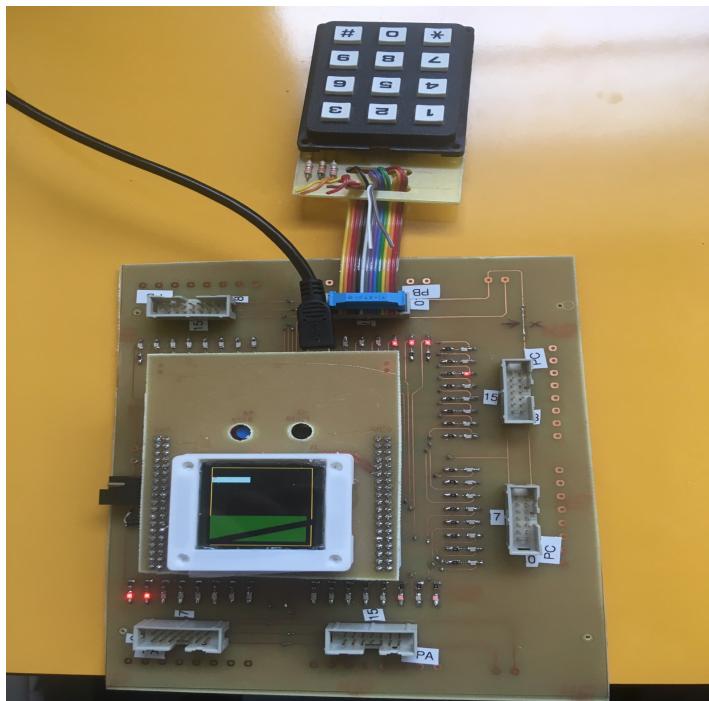


Figure 1: Câblage de la carte Nucleo

### 3. EXEMPLE D'IMPLANTATION

Il s'agit d'implanter l'application décrite par la structure fonctionnelle de la figure suivante :

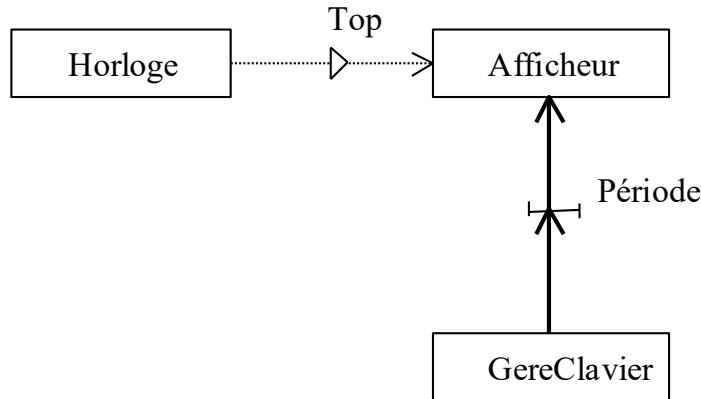


Figure 2: Description fonctionnelle.

La tâche **Horloge** est une tâche périodique qui doit transmettre un événement **Top** à chaque itération. L'activation de la tâche Horloge sera réalisée toutes les secondes. La tâche **Afficheur** doit inscrire dans la fenêtre de l'écran Oled la variable partagée **Période**. **GereClavier** est une tâche permanente qui incrémentera la valeur de la variable partagée **Période** et qui affiche sur l'écran Oled la touche qui a été appuyée par l'utilisateur sur le clavier matriciel.

- récupérer un projet CubeMX FreeRTOS\_TP1.ioc sur le serveur réseau  
[nas2.educ.insa/PUBLIC/EII/TP\\_2019-2020/4EII/TpsReel](nas2.educ.insa/PUBLIC/EII/TP_2019-2020/4EII/TpsReel)
- « ». Dans ce projet, les broches suivantes ont été configurées comme suit :
  - PA0 / PA1 : communication UART afin de communiquer avec l'écran OLED
  - PA2 / PA3 : communication USART2 afin de communiquer et d'afficher via des appels printf dans votre code des messages sur la console TeraTerm de votre PC
  - PB0 → PB6 : GPIO pour acquérir la valeur du clavier matriciel
  - Au niveau de l'onglet Middleware/FreeRTOS, vérifiez que la case « Enabled » générale a bien été activée ainsi que certaines options dans les fenêtres de configuration.
- Reportez les valeurs concernant les tâches de la figure 2 dans le tableau suivant :

TaskName	Priority / Type	Stack Size	Entry Function / CallBack

- Génerez le projet Keil à partir de CubeMX
- Modifiez le projet Keil en ajoutant les fichiers source des périphériques uoled et clavier fournis sur le serveur (répertoire « Peripheriques »)
- Modifiez le fichier principal de votre projet :
  - Ajoutez les #include nécessaires à l'utilisation des périphériques
  - Ajoutez une variable globale « periode » de type caractère (variable partagée Fig.2)
  - Ajoutez le code suivant permettant l'utilisation de printf() via TeraTerm sur votre PC

```
/* USER CODE BEGIN PFP */
#ifndef __GNUC__
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif
/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */
PUTCHAR_PROTOTYPE
{
HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
    return ch;
}
/* USER CODE END 0 */
```

- Initialisez l'écran Oled grâce au code suivant :

```
gfx_Cls();
setbaudWait(115200,BAUD_115200);
```

- Démarrez votre Timer à l'endroit indiqué par le commentaire comme suit :

```
/* start timers, add new ones, ... */
osTimerStart(HorlogeHandle,1000);
```

- Modifiez le code des tâches GereClavier et Afficheur et comme suit :

```
/* USER CODE END Header_StartGereClavier */
void StartGereClavier(void const * argument)
{
    char touche;
    for(;;)
    {
        periode++;
        touche = clavier_scrute();
        printf("clavier : %d\n\r", periode);
        gfx.RectangleFilled(0,10,16,16,WHITE);
        HAL_Delay(50);
        txt_MoveCursor(0,10);
        HAL_Delay(50);
        txt_BGcolour(WHITE) ;
        txt_FGcolour(WHITE) ;
        HAL_Delay(50);
        putCH(touche);
    }
}
```

```
void StartAfficheur(void const * argument)
{
    for(;;)
    {
        osSignalWait (0x0001, osWaitForever);
        printf("Afficheur : %d\n\r", (int)periode);
        gfx_RectangleFilled(0,2,6,3,PALETURQUOISE);
        txt_MoveCursor(0,2);
        txt_BGcolour(PALETURQUOISE) ;
        txt_FGcolour(CHOCOLATE) ;
        putCH(periode);
    }

    void CallbackHorloge(void const * argument)
    {
        printf("horloge : \n\r");
        osSignalSet (AfficheurHandle, 0x0001);
    }
}
```

- Compilez et exécutez le code de votre projet. Visualisez les messages en utilisant TeraTerm et analysez le fonctionnement du système.

Vous avez deux possibilités pour faire évoluer votre projet. Vous pouvez modifier le projet CubeMX ou directement modifier votre projet sous Keil.

Si vous utilisez CubeMX : le fichier main.c sera généré de nouveau en conservant le code spécifique que vous avez intégré entre les balises « USER CODE ». Attention, si vous avez effacé ces balises, votre code sera peut être effacé. Pensez à sauvegarder votre fichier « main.c » avant de le regénérer.

Si vous modifiez votre projet sous Keil : n'oubliez pas d'activer les API que vous utilisez dans le fichier « FreeRTOSConfig.h ».

- Quelle la priorité min du système ?
- Quelle la priorité max du système ?
- Quelle est la priorité de la tâche Horloge ?
- Que se passe t'il si vous mettez la tâche « GereClavier » plus prioritaire ?
- Que se passe t'il si vous mettez une taille de pile à 20 ?
- Envoyez un événement « TOP » toutes les 2 secondes au lieu de toutes les secondes
- Remplacez l'utilisation d'un signal par un sémaphore pour l'événement « TOP »

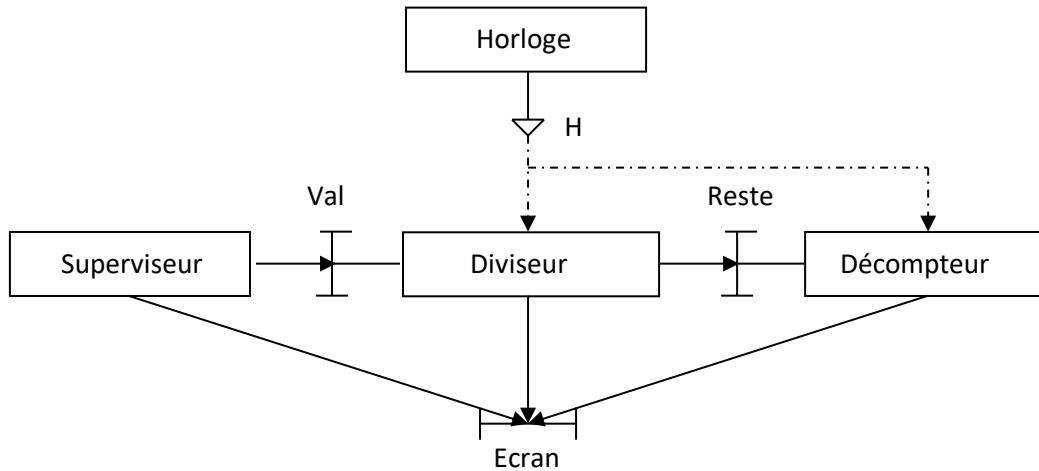
## TP 2 : Exclusion mutuelle, synchronisation

Cette manipulation a pour objectif :

- d'analyser la synchronisation par sémaphore,
- de mettre en œuvre l'exclusion mutuelle,
- d'analyser l'influence des priorités respectives des tâches.

### 1. SITUATION

On désire implanter la structure fonctionnelle ci-dessous à l'aide d'un exécutif temps-réel :

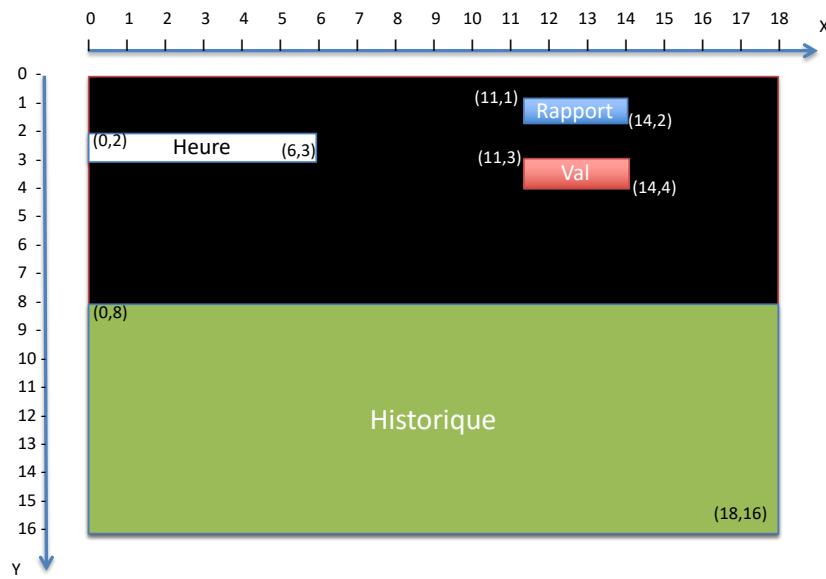


**Figure 3: Structure fonctionnelle**

La description comportementale des fonctions est présentée ci-dessous. Afin de suivre l'évolution de l'application (commutation des tâches, ...), utilisez la fonction « `printf` » qui permettra d'afficher des messages dans la console TeraTerm de votre PC.

## 2. CONTRAINTES D'IMPLANTATION

La présentation des résultats à l'écran doit respecter la décomposition en fenêtres suivante :



**Figure 4:** présentation écran

Dans un premier temps, vous n'afficherez pas la fenêtre « historique ».

### 2.1 Fonction Horloge

La tâche **Horloge** est une tâche périodique qui doit transmettre un événement **H** à chaque itération. L'activation de la tâche Horloge sera réalisée toutes les secondes.

### 2.2 Fonction Diviseur

Action temporaire activée par l'événement H. Une activation sur 3, la fonction recopie la valeur de la variable Val dans la variable Reste, et affiche le temps écoulé depuis le lancement de l'application dans la fenêtre qui lui est réservée (format « minutes : secondes »).

### 2.3 Fonction Superviseur

Il s'agit d'une fonction permanente (tâche de fond). Elle gère les touches du clavier matriciel : 0 pour incrémenter et 1 pour décrémenter la variable Val. La valeur de cette variable doit être initialisée à 5 et toujours être comprise entre 0 et 9 inclus. Superviseur doit afficher la valeur de Val, après chaque modification, dans la fenêtre destinée à cet effet.

## 2.4 Fonction Décompteur

Son comportement fonctionnel est le suivant :

```

action Décompteur    sur événement H
  avec ( entrée Var   Reste :      integer,
         sortie Var    Ecran :     screen);

  var div_raz, rapport : integer
  begin
    rapport:=Reste;
    div_raz:=10;
    cycle H:
    begin
      div_raz:=div_raz-1;
      if (div_raz = 0)
        then begin
          rapport:=Reste;
          div_raz:=10;
        end.
      else
        if (rapport > 0)
          then rapport:=rapport-1;
        édition(rapport, Ecran);
    end;
  end_cycle;
end_Décompteur;

```

## 3. TRAVAIL A EFFECTUER

- A partir du fichier de configuration CubeMX du TP1, faites évoluer la configuration FreeRTOS afin d'implanter la structure fonctionnelle demandée. Pour cela, identifiez les éléments (tâches, sémaphores, ...) strictement nécessaires à l'implantation et déterminez en conséquence les paramètres d'initialisation du noyau temps-réel. Plus précisément :
  - Comment ne nomme la tâche de fond ?
  - Combien de tâches sont créées et à l'aide de quelle fonction ?
  - Combien de sémaphores sont créés et à l'aide de quelle fonction ?
- Générez le projet Keil à partir de votre nouvelle configuration et complétez le code du fichier principal (configuration + code associé à chacune des tâches).
- Analyser via la console du PC le comportement de l'application en fonction de la priorité respective des tâches, de leur ordre de création et du principe de chaînage des tâches en attente des sémaphores. Modifier les niveaux de priorité pour analyser les différents cas.



#### 4. Gestion de l'historique

Dans votre projet, ajoutez les éléments suivants :

```
/* USER CODE BEGIN PV */  
/* Private variables */  
char histo[67] = {0,};  
char hist_wr = 0;
```

```
/* Private function prototypes -----*/  
void trace(char car, char num);
```

```
/* USER CODE BEGIN 5 */  
void trace(char car, char num) {  
    if (hist_wr < 64) {  
        histo[hist_wr++] = car;  
        histo[hist_wr++] = num;  
        histo[hist_wr++] = ' ';  
        histo[hist_wr] = 0;  
    }  
}
```

Dans vos tâches, vous pourrez appeler la fonction trace en utilisant un caractère « car » différent pour chacune des tâches et des caractères « num » différents si vous utilisez plusieurs fois la fonction trace dans une même tâche. Cela aura pour effet de remplir le tableau « histo ». Dans votre tâche de fond, il est alors possible d'afficher le contenu du tableau « histo » afin de voir dans quelles tâches votre programme est passé et à quel moment.

Cette gestion des traces peut remplacer ou compléter l'utilisation des « printf », utilisez les primitives de trace et affichez l'historique dans la fenêtre dédiée à partir de la tâche de fond. Analysez le comportement. Corrigez le problème d'affichage en utilisant la technique de l'exclusion mutuelle par séaphore, puis par modification de priorité et par l'utilisation d'un mutex. Assurez-vous d'avoir bien compris les différences entre ces 3 techniques.



## TP 3 : Attente passive versus Attente active

Cette manipulation a pour objectif :

- de montrer la différence entre attente active et attente passive,
- de mettre en évidence l'importance de l'ordre d'exécution des instructions,
- d'analyser la nécessité d'une exclusion mutuelle.

### 5. CAHIER DES CHARGES

Il s'agit d'implanter une procédure, My\_Sleep(i), permettant de mettre en sommeil (attente passive) une tâche pendant une durée déterminée. Cette procédure sera comparée à une attente active. Dans un premier temps, une seule tâche peut faire appel à My\_Sleep(i).

La structure fonctionnelle générale est donnée figure suivante :

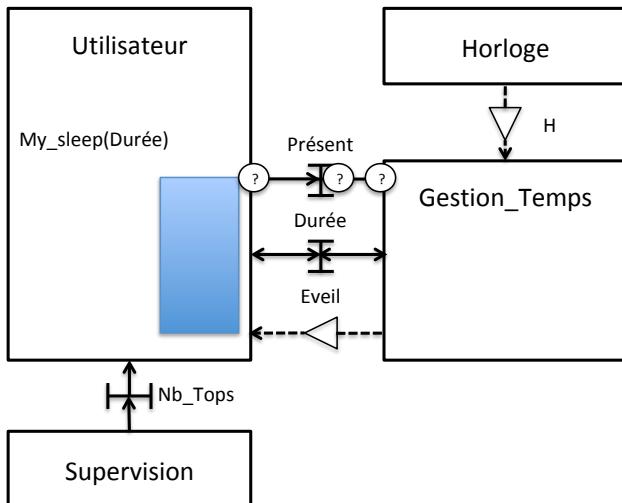


Figure 5: Structure fonctionnelle globale avec gestion du mécanisme My\_Sleep.

Les différentes spécifications sont données ci-dessous :

#### 5.1 Procédure My\_sleep

Le principe est de réaliser une temporisation sous la forme d'une attente passive. La procédure spécifie la présence d'une demande (Présent=Vrai) ainsi que la durée d'attente en termes de nombre coup d'horloge, puis vient se bloquer sur la réception du sémaforo Eveil.

#### 5.2 Tâche Gestion\_temps

Activée sur Div\_H, elle vérifie la présence d'une demande, puis décrémente Durée jusqu'à la valeur 0. La tâche libère ensuite la demande à travers le sémaforo Eveil.

#### 5.3 Tâche Superviseur

La tâche superviseur doit à la fois récupérer Nb\_Tops à partir du clavier (valeur entre 0 et 9), et afficher l'historique. Elle constitue la tâche de fond.

## 5.4 Tâche Utilisateur

La tâche utilisateur est la seule tâche à appeler les 2 types d'attente. Son algorithme est donné par :

```
action Utilisateur()
begin
    cycle
    begin
        affiche(date_sommeil);
        My_Sleep(Durée);
        affiche(date_eveil);
        Delay(4s);
    end.
end_cycle;
end_utilisateur;
```

Durée est calculée à partir de Nb\_Tops pour que l'attente dure de 1 à 9 secondes.

Delay sera implanté comme une procédure d'attente active :

```
***** définition de la fonction Delay *****
void delay (long secondes)
{
    volatile unsigned int i, j;
    for (i = secondes*60; i > 0; i--)
        for (j=65535; j > 0; j--);
}
```

Une autre solution consiste à utiliser la fonction de HAL (Hardware Abstraction Layer) « HAL\_Delay »

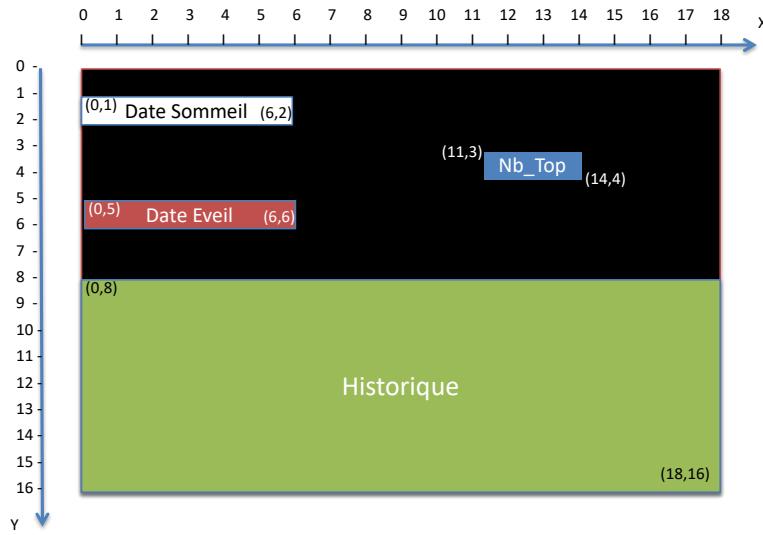
## 5.5 Tâche Horloge

La tâche Horloge envoie un événement périodique H. Vous pouvez utiliser soit un signal, soit un séaphore dans votre implémentation CMSIS. Pour gérer la priorité de la tâche Horloge sous CubeMX, vous pouvez donner une priorité TIMER\_TASK\_PRIORITY comprise en 0 et MAX\_PRIORITIES. En conservant MAX\_PRIORITIES=7, vous avez la correspondance suivante :

osPriorityIdle	0
osPriorityLow	1
osPriorityBelowNormal	2
osPriorityNormal	3
osPriorityAboveNormal	4
osPriorityHigh	5
osPriorityRealtime	6

## 6. CONTRAINTES D'IMPLANTATION

La gestion de l'affichage doit suivre le schéma suivant :



**Figure 6:** Affichage écran.

## 7. TRAVAIL A EFFECTUER

### 7.1 Préciser et implanter la solution à 1 utilisateur

- Indiquer la nature des accès à la variable Présent par Gestion\_Temps (lecture, écriture ou lecture/écriture).
- Spécifier le comportement de My\_Sleep et Gestion\_Temps par des réseaux de Pétri, puis des descriptions algorithmiques.
- Identifier les éléments nécessaires à l'implantation de l'application et justifier le besoin éventuel d'exclusion mutuelle. Déterminer les paramètres du noyau temps-réel.
- Décrire le programme C utilisant CMSIS pour réaliser l'application.
- Quelle est la principale différence entre la procédure Delay et la procédure My\_Sleep ? Quels sont les temps d'attente de ces deux primitives ?
- Que se passe t'il si la variable NB\_Tops est initialisée à 0 ?
- Réaliser une première implantation en respectant les priorités :
  - Prio(Horloge) > Prio(Gestion\_temps) > Prio(Utilisateur).
- Tester les autres possibilités de priorités relatives.

### 7.2 Etendre la solution à un nombre n d'utilisateurs

Il s'agit de multiplier le nombre d'utilisateurs de la primitive My\_Sleep en ne gardant qu'une seule fonction Gestion\_temps. Plusieurs tâches Utilisateur de fonctionnement identique sont implantées, chacune réalisant un appel à My\_Sleep,

- Déterminer les éléments de la structure fonctionnelle qui doivent être dupliqués. Pour l'implantation, il est conseillé de regrouper ces éléments dans un tableau.
- Définir la nouvelle solution en affectant dynamiquement les éléments libres du tableau aux utilisateurs.

## TP 4 : Transfert de messages

Cette manipulation a pour objectif d'implanter le mécanisme de boîte aux lettres,

### 1. CAHIER DES CHARGES

On considère la structure fonctionnelle suivante :

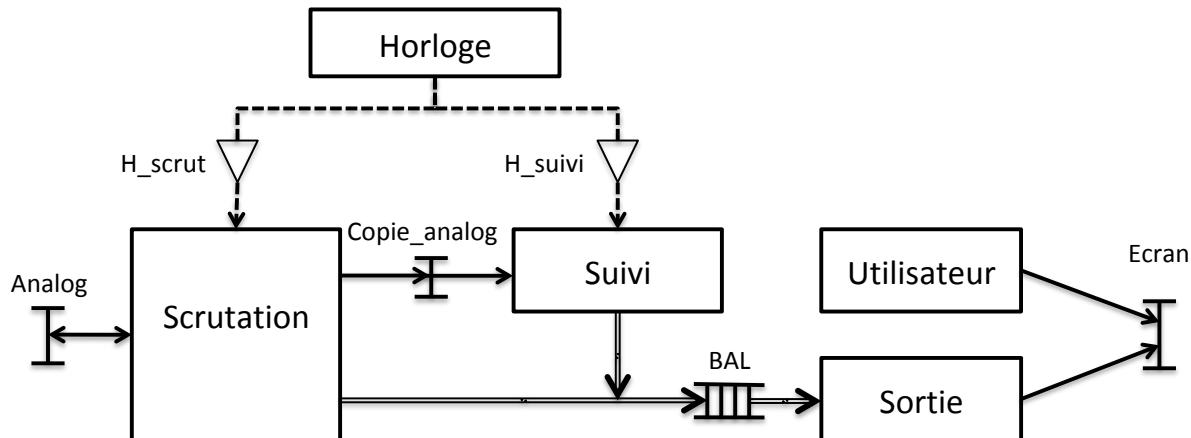


Figure 7: Structure fonctionnelle globale.

#### 1.1 Tâche Scrutation

Action activée sur **H\_scrut**. Elle fait l'acquisition de la valeur de la variable **Analog**, l'incrémenté puis la copie dans la variable **Copie\_analog**. Si la valeur de **Analog** dépasse le seuil de 10, un message de type "alarme" est déposé dans la boîte aux lettres et la variable **Analog** est remise à 0.

#### 1.2 Tâche Suivi

Activée par **H\_suivi**. Elle met des messages de type "suivi" dans la boîte aux lettres, ces messages contenant la valeur de la variable **Copie\_analog**.

#### 1.3 Tâche Horloge

La tâche **Horloge** est une tâche périodique activée toutes les secondes. Elle envoie un événement **H\_scrut** toutes les secondes et **H\_suivi** toutes les 3 secondes. Vous pouvez utiliser soit des signaux, soit des sémaphores dans votre implémentation CMSIS.

#### 1.4 Tâche Sortie

Cette tâche récupère le contenu de la boîte à lettre et l'affiche à l'écran.

#### 1.5 Tâche Utilisateur

Action permanente permettant à l'utilisateur d'éditer des lignes de texte dans la partie inférieure de l'écran. Le comportement souhaité est d'interdire l'affichage des messages tant que l'utilisateur n'a pas terminé la saisie de la ligne en cours (caractère \*).

Voici un exemple de code pour la tâche Utilisateur :

```

void StartUtilisateur(void const * argument)
{
    /* USER CODE BEGIN StartUtilisateur */
    char caract, caract_memo;
    int caract_pos,i;
    char chaine_caract[10];

    printf("utilisateur : %c\n\r", '0');

    for(;;) {
        printf("utilisateur : %c\n\r", '1');
        caract = clavier_scrute();
        caract_pos=0;
        caract_memo = 'a';

        osSemaphoreWait(Mut_ecranHandle, osWaitForever);
        while (caract != '*') {
            switch(caract){
                case '1' : caract_memo ++ ; break;
                case '2' : caract_memo -- ; break;
                case '#' : caract_pos= (caract_pos+1)%10;
                            caract_memo = 'a'; break;
                default : break;

                chaine_caract[caract_pos]=caract_memo;
                for (i=caract_pos+1;i<10;i++) chaine_caract[i]=' ';

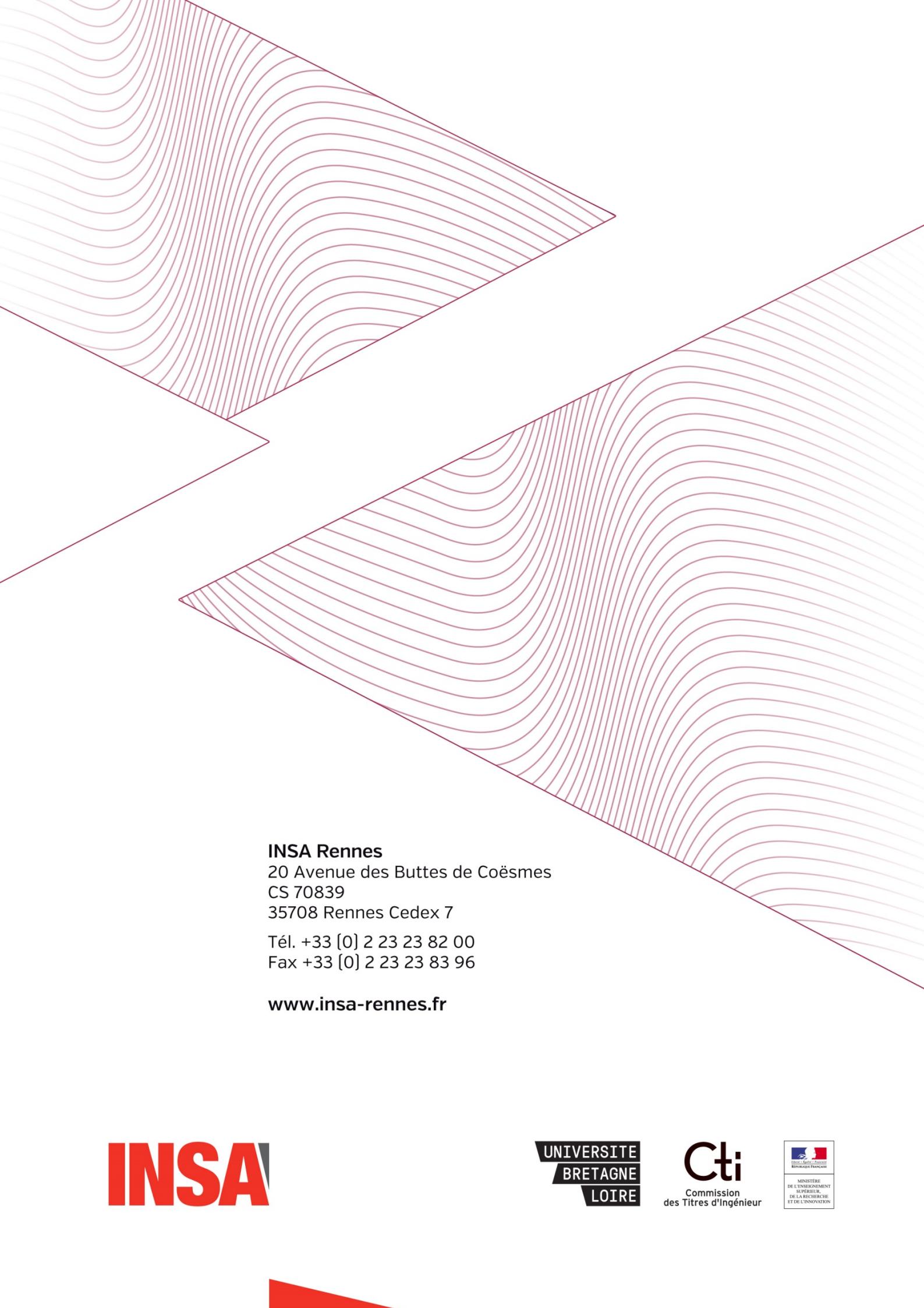
                gfx_RectangleFilled(0,6,10,10,WHITE);
                txt_MoveCursor(0,6);
                txt_BGcolour(WHITE) ;
                txt_FGcolour(BLACK) ;
                for (i=0;i<10;i++) putCH(chaine_caract[i]);

                caract = clavier_scrute();    }
            osSemaphoreRelease(Mut_ecranHandle);
        /* USER CODE END StartUtilisateur */
    }
}

```

## 2. TRAVAIL A EFFECTUER

- Spécifier sous forme de réseau de Petri le comportement des fonctions Utilisateur et Sortie.
- Définir une solution pour l'implantation de la boîte aux lettres BAL (taille finie) en n'utilisant que le partage de variables et la synchronisation par sémaphore (envisager l'exclusion mutuelle si nécessaire).
- Définir les primitives SEND et RECEIVE correspondant à cette solution.
- Décrire le programme C pour réaliser l'application.
- Implanter l'application définie par la structure fonctionnelle de la figure 3 en utilisant le mécanisme « Message Queue » disponible dans l'OS CMSIS.



**INSA Rennes**  
20 Avenue des Buttes de Coësmes  
CS 70839  
35708 Rennes Cedex 7  
Tél. +33 [0] 2 23 23 82 00  
Fax +33 [0] 2 23 23 83 96

[www.insa-rennes.fr](http://www.insa-rennes.fr)

**INSA**

UNIVERSITE  
BRETAGNE  
LOIRE

**Cti**  
Commission  
des Titres d'Ingénieur

