Nom, prénom :

TP2 Système d'exploitation temps réel pour Capteurs / Actionneurs

Polytech Nice Sophia

Vous devrez répondre sur l'énoncé aux questions R1, R2, …
Vous devrez écrire, tester puis montrer à l'enseignant les codes C1, C2, …

Ouvrez le logiciel Nios-II SBT for Eclipse, créer une nouvelle « Nios-II application and BSP from template ».
Télécharger le projet fourni sur Jalon et choisissez :

- SOPC information file name : Design_Files\nios1.sopcinfo
- name : un nom de projet sans espace
- template : Hello_uCOS

Cliquez sur Finish, cross-compiler le projet et tester le code (cf. annexe 1).

L'objectif de ce TP est de reprendre le comportement du TP précédent mais en utilisant une couche logicielle supplémentaire, celle de l'OS temps réel uC/OS-II.
La documentation sur cet OS est disponible sur https://doc.micrium.com/display/osiidoc/home

1) Création de tâches

R1. Rappeler les paramètres des deux fonctions de création de tâches uCOS.

OSTaskCreate Paramètre	Type

OsTaskCreateExt Paramètre	Type

2) Services de communication et de synchronisation inter-tâches

R2. Citer les différents services disponibles dans uCOS.

Service	Communication ou Synchronisation ?

3) Premier code avec un RTOS

C1. Adapter le code du template pour créer une communication entre les 2 tâches, la première envoie la valeur d'un compteur puis attend un certain délai D1, la seconde affichant sur les 7-segment la valeur reçue puis attend un autre délai D2. Tester la différence entre le service mailbox et le service MessageQueue en adaptant les exemples fournis dans le chapitre 1 de la documentation.

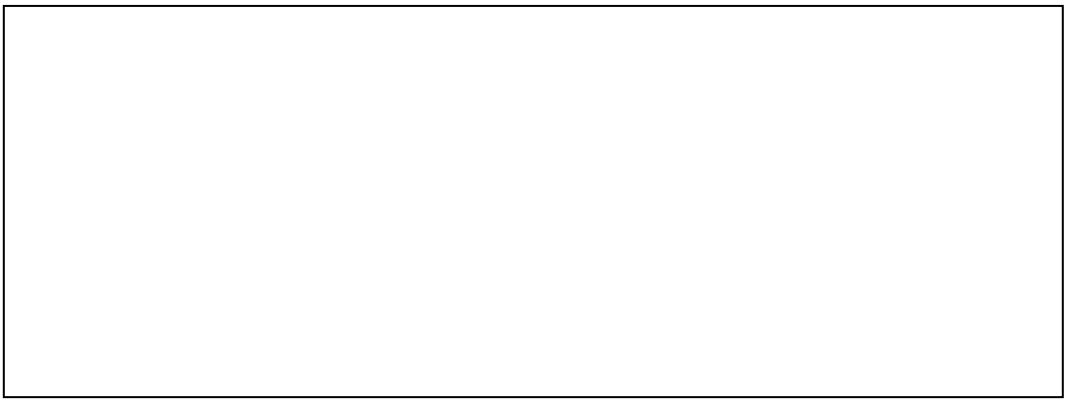
Nom, prénom :

Tester le comportement de la MessageQueue pour différents facteurs multiplicatifs entre D1 et D2**.** D1 = k.D2, pour k in [10, 5, 2, 1, 0.5, 0.01].
R3. Que constatez-vous ?

C2. Reprenez le code du test de réflexe du TP1 et passer le en code multi-tâches uCOS.
Pour cela, vous devez d'abord :

- Identifier les tâches nécessaires (3 au minimum).
- Fixer les priorités
- Définissez les services IPC nécessaires
- Utilisez le service OSTimeGet pour lire le temps à la place de alt_timestamp (vérifier que la case os_tmr_en est cochée dans le menu advanced-> ucosii de l'éditeur de BSP)

R4. Faite le schéma des communications entre tâches et ISR, en nommant les tâches selon leur priorité (T1, T2, T12…).



Tester le code

4) Monitoring de l'exécution

C3. Ajouter le code nécessaire à l'instrumentation du code :

- taille de pile + taux d'utilisation du processeur, cf. annexe 2
- temps d'exécution, cf. annexe 3

Ajouter à votre code une tâche spécifique, de basse priorité, qui se chargera d'afficher les informations de monitoring de toutes les tâches.

R5. Noter les informations mesurées.

Tâche	Nombre d'exécutions	Temps d'exécution moyen	Pile libre	Pile utilisée

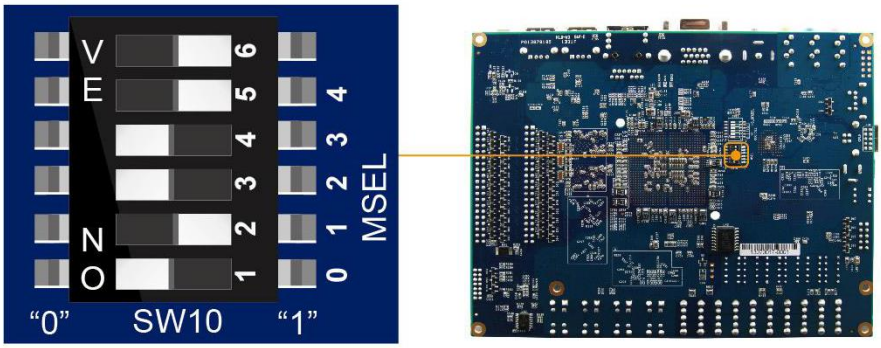
Nom, prénom :

Annexe 1 – Programmation de la carte DE1 SoC

Pour Configurer le circuit FPGA sur la cible (une seule fois) :

- brancher la carte au port USB Blaster
- lancer Nios II > Quartus II Programmer,
- ajouter le device SoC Series V-> SoCVHPS et ajouter le fichier suivant
- SDRAM_DE1_SoC\output_files\Test_Quartus_161_time_limited.sof
- Cliquer sur start pour configurer le FPGA

Les micro-switches sur la face arrière de la carte doivent être dans la position suivante :



Pour télécharger l'exécutable (fichier .elf) dans la mémoire du micro-contrôleur :

- compiler le projet
- lancer Run > Run Configuration …
- New NiosII Hardware
- Run

Annexe 2 – Mesure de taille de pile

La tâche statistique de uC/OS-II peut être utilisée pour monitorer l'exécution du code :

- Taux d'utilisation du CPU
- Mesure de taille de pile

Pour autoriser l'exécution de cette tâche OS_TaskStat(), la constante OS_TASK_STAT_EN doit être à 1 dans OS_CFG.H (à définir via le BSP Editor d'eclipse dans le menu menu advanced->ucosii).
Pour que cette tâche puisse mesurer la pile effectivement utilisée par les autres tâches, celle-ci doivent être créées avec la fonction OSTaskCreateExt, en passant comme dernier paramètre les flags de mesure de pile : OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR.

La mesure de taux d'utilisation est fournie dans la variable OSCPUUsage.

La mesure de taille de pile est réalisée par la fonction OSTaskStkChk(int prio, OS_STK_DATA *p), où prio est la priorité de la tâche à mesurer, et p le pointeur sur une structure OS_STK_DATA définie dans uCOS_II.h qui contient deux champs : OSFree et OSused, représentant le nombre d'octets libres et utilisés par la tâche prio.

Annexe 3 – Instrumentation personnelle du code

On peut se servir des fonctions « Hook » de uC/OS-II pour étendre son code à des instants particuliers de l'exécution. Les prototypes sont définis dans BSP/HAL/src/os_cpu_c.c :

- **OSInitHookBegin**, Démarrage de l'OS,
- OSInitHookEnd, Fin du démarrage de l'OS,

Nom, prénom :

- OSTaskCreateHoo, création d'une tâche,
- OSTaskDelHook, Suppression d'une tâche,
- OSTaskIdleHook, Exécution de la tâche Idle,
- OSTCBInitHook, Initialisation des TCB,
- **OSTimeTickHook**, Réveil de l'OS par le timer,
- **OSTaskSwHook**, Changement de contexte

On utilisera la fonction liée au changement de contexte pour comptabiliser des informations sur l'exécution de chaque tâche. Ces informations seront stockées dans une structure :

```
Typedef struct {
        Char TaskName[30] ;
        INT16U      TaskCtr ;
        INT16U      TaskExecTime ;
        INT16U      TaskTotExecTime;
} TASK_USER_DATA;
```

Il y aura autant de structures que de tâches :

```
TASK_USER_DATA          TaskUserData[NB_TASKS] ;
```

Le code de la fonction Hook est à ajouter à votre code :

```
void OSTaskSwHook(void)
{
        INT16U taskStopTimestamp, time;
        TASK_USER_DATA *puser;

        taskStopTimestamp = OSTimeGet();

        time =(taskStopTimestamp - taskStartTimestamp) / (OS_TICKS_PER_SEC / 1000); // in ms
        puser = OSTCBCur->OSTCBExtPtr;
        if (puser != (TASK_USER_DATA *)0) {
                puser->TaskCtr++;
                puser->TaskExecTime = time;
                puser->TaskTotExecTime += time;
        }

        taskStartTimestamp = OSTimeGet();
}

void OSInitHookBegin(void)
{
   OSTmrCtr = 0;
   taskStartTimestamp = OSTimeGet();
}

void OSTimeTickHook (void)
{
   OSTmrCtr++;
   if (OSTmrCtr >= (OS_TICKS_PER_SEC / OS_TMR_CFG_TICKS_PER_SEC)) {
      OSTmrCtr = 0;
      OSTmrSignal();
   }
}
```