1A GPSE FISA – Année 2025-2026

SYSTÈMES EMBARQUÉS (TP)

Nom	Prénom	Table

Examen pratique

Durée : 1h30

Ressources autorisées :
- Tous documents
- Internet

Consignes

Appelez l'enseignant pour valider ***CHAQUE*** étape.

Ne restez pas bloqué trop longtemps : appeler l'enseignant pour des indices ou corriger une erreur persistante.

Prenez le temps de lire le fichier `main.c` et les autres fichiers du BSP pour vous approprier l'existant.

Toutes les réponses sont à intégrer dans le fichier `main.c` (même si des modifications de périphériques sont demandées).

En fin de séance, vous déposerez votre fichier `main.c` (et uniquement ce fichier) sur l'espace Moodle dédié.

1. Création Projet

10 min

Téléchargez l'archive d'évaluation **« `eval_fisa_squelette.zip` »** depuis la plateforme Moodle.

Renommez-la en **« `eval_nom_prenom.zip` »**, puis l'extraire sur le disque dur.

Créez un projet MPLAB dans le répertoire **`eval_nom_prenom/apps/eval_tp/pjct`** .

Ajoutez au projet le fichier source **`eval_nom_prenom/apps/eval_tp/src/main.c`**.

Assurez-vous de la **bonne compilation du projet**.

Compilez et appelez l'enseignant pour validation.

2. Initialisation de l'UART

10 min

Initialisez l'UART de sorte à pouvoir transmettre la chaîne de caractères « Application starting » au démarrage de l'application. Configuration : 9600 baud + 1 bit de stop + 0 bit de parité (comme en TP).

Préparez un terminal série que vous conserverez pour tous les développements futurs.

Testez votre application et appelez l'enseignant pour validation.

3. Lecture de caractère depuis l'UART

10 min

Déclenchez une lecture de caractère depuis le périphérique UART1.

Si un caractère est reçu, vous devez transmettre un message indiquant le caractère reçu.

Testez votre application et appelez l'enseignant pour validation.

4. Développement de l'application

60 min

Consignes communes

L'algorigramme (sur la dernière page du sujet) montre plusieurs fonctions à implémenter, chacune étant appelée en fonction du dernier caractère reçu par UART (stocké dans la variable **`state`**).

Chaque fonctionnalité doit être implémentée dans sa propre fonction, rédigée dans le fichier **`main.c`**. La fonction **`main()`** ne contient que le test de la variable **`state`** et l'appel aux fonctions développées (cf pages suivantes).

Les fonctionnalités demandées sont indépendantes : traitez-les dans l'ordre que vous voulez. Toutefois notez que l'ordre présenté correspond à une difficulté croissante.

N'oubliez pas de mettre à jour la partie initialisation de votre fonction **`main()`** pour activer les périphériques nécessaires.

Pour chaque fonctionnalité développée, ***testez votre application et appelez l'enseignant pour validation.***

Consignes spécifiques à chaque fonctionnalité

`all_leds_on()`

Allume toutes les LEDs de la carte Microchip Curiosity HPC (D2 à D5).

`all_leds_off()`

Éteind toutes les LEDs de la carte Microchip Curiosity HPC (D2 à D5).

`all_leds_blink_200ms()`

Toutes les LEDs de la carte Microchip Curiosity HPC (D2 à D5) changent d'état, puis une attente de 200 ms par délai logiciel est déclenchée.

La fonction ne fait changer les LEDs d'état qu'une seule fois : ce sont les appels successifs à cette fonction qui font réellement clignoter les LEDs.

`all_leds_blink_20ms()`

Toutes les LEDs de la carte Microchip Curiosity HPC (D2 à D5) changent d'état, puis le Timer0 est lancé pour déclencher une interruption 20 ms plus tard. Il existe plusieurs moyens de répondre à cette question.

La fonction ne fait changer les LEDs d'état qu'une seule fois : ce sont les appels successifs à cette fonction qui font réellement clignoter les LEDs.

`all_leds_blink_50ms()`

Toutes les LEDs de la carte Microchip Curiosity HPC (D2 à D5) changent d'état, puis le Timer0 est lancé pour déclencher une interruption 50 ms plus tard. Cette fonctionnalité **ne** *nécessite* **pas** de changement dans la configuration du périphérique Timer0.

La fonction ne fait changer les LEDs d'état qu'une seule fois : ce sont les appels successifs à cette fonction qui font réellement clignoter les LEDs.

`all_leds_blink_100ms()`

Toutes les LEDs de la carte Microchip Curiosity HPC (D2 à D5) changent d'état, puis le Timer0 est lancé pour déclencher une interruption 100 ms plus tard. . Cette fonctionnalité *nécessite* un changement dans la configuration du périphérique Timer0, changement que vous écrirez dans le `main()` juste après l'appel à `timer0_init()`.

La fonction ne fait changer les LEDs d'état qu'une seule fois : ce sont les appels successifs à cette fonction qui font réellement clignoter les LEDs.

`leds_follow_switches()`

Soient les LEDs D5 à D2 (broches RA7 à RA5).

Version simple :

Déplacez les LEDs allumées vers la gauche (vers le poids fort) à chaque appui sur le bouton S1 ou vers la droite (vers le poids faible) à chaque appui sur le bouton S2. Si le résultat obtenu est « aucune LED allumée », alors modifiez de sorte à avoir au moins une LED allumée.

Version aboutie :

Si les LEDs sont toutes éteintes, alors on impose l'allumage de la LED D2 : ❺❹❸②

Si les LEDs sont toutes allumées, alors on impose l'allumage de la LED D5 : ⑤❹❸❷

Un appui sur le bouton S1 décale les LEDs vers la gauche (vers le poids fort), en imposant D5 comme limite (on ne peut pas aller plus loin) : ⑤❹❸❷

Un appui sur le bouton S2 décale les LEDs vers la droite (vers le poids faible), en imposant D2 comme limite (on ne peut pas aller plus loin) : ❺❹❸②

`leds_follow_adc()`

Le périphérique ADC est opérationnel : les fichiers pilotes **`adc_init()`** et **`adc_read()`** sont complets et fournis dans le répertoire **`bsp/adc/src/`**. La fonction **`adc_read()`** retourne une valeur sur 8 bits, résultant de la conversion analogique numérique sur la broche RA0 (reliée au potentiomètre).

Allumez entre 0 et 4 LEDs en fonction de la valeur analogique imposée sur la broche RA0 via le potentiomètre (comme une jauge ou un Vu-mètre), selon la structure suivante :

- 0 % <= Valeur <= 20 % : ❺❹❸❷
- 20 % < Valeur <= 40 % : ❺❹❸②
- 40 % < Valeur <= 60 % : ❺❹③②
- 60 % < Valeur <= 80 % : ❺④③②
- 80 % < Valeur <= 100 % : ⑤④③②

*Algorigramme de l'application complète (fonction **`main()`**).*