

I - Environnement de travail

I.a La carte Microchip Explorer 8

Vous utiliserez dans cette série de TP la carte **Microchip Explorer 8**. Cette carte dispose actuellement d'un microcontrôleur PIC16F1719, et de diverses entrées-sorties (8 LEDs, 1 Ecran LCD, 2 boutons poussoirs, 1 potentiomètres, et divers connecteurs d'entrée-sortie).

I.a.1 Cavaliers

Avant de commencer, vous devrez vous **assurer que les cavaliers présents sur la carte sont bien positionnés**. Ces cavaliers permettent d'établir ou de rompre des connexions entre les composants de la carte en fonction de l'utilisation qui en est faite. Pour vérifier leur état, référez-vous à documentation de la carte Explorer 8 [4] que vous trouverez sur le groupe Teams :

2223_ISEN_BORDEAUX_AP4_FPGA_MICROCONTROLEUR_FPGA et
Microcontrôleurs/Supports de cours/Microcontrôleurs/TPs/Documentations
techniques/ 2022-2023_4A_S8_AP4_CircuitsProgrammables_TP-uC_Documentation-
Explorer8.pdf

L'état des cavaliers à vérifier y est décrit dans la table 1-1 (p. 14).

I.a.2 Programmeur

Pour programmer le microcontrôleur, vous utiliserez un **programmeur PICKit3**. Il sera branché au connecteur J12 comme décrit dans la documentation de la carte Explorer 8 [4] en Fig. 2-3 (p. 23).

I.a.3 Alimentation

La carte Explorer 8 sera alimentée par le port Micro USB (J18).

I.b Environnement MPLABX

Tous les programmes seront écrits en **langage C**, dans l'IDE **MPLABX**, et seront compilés par le compilateur **XC8**. Vous trouverez dans le groupe Teams un document d'annexe MPLABX donnant des indications pour l'installation de MPLABX et du compilateur XC8 :

2223_ISEN_BORDEAUX_AP4_FPGA_MICROCONTROLEUR_FPGA et
Microcontrôleurs/Supports de cours/Microcontrôleurs/TPs/2022-
2023_4A_S8_AP4_CircuitsProgrammables_TP-uC_Annexe-MPLABX.pdf

Cette annexe décrit également des procédures dont vous aurez besoin lors de la réalisation des exercices des TP :

- La création de projets sous MPLABx
- L'ajout de sources à un projet
- La programmation du microcontrôleur

I.c Répertoires de travail

Pour **organiser vos fichiers de TP**, créez à l'emplacement de votre choix une arborescence de répertoires dédiée :

```
AP4_Circuits_Programmables/  
├── Microcontrôleurs/  
│   ├── TPs/  
│   └── Ressources/
```

Vous placerez chaque nouveau projet créé dans MPLABX dans le répertoire AP4_Circuits_Programmables/Microcontrôleurs/TPs/. Il y sera automatiquement créé un sous-répertoire Nom_du_projet.X/ dans lequel seront rassemblées les sources du projet.

I.d Sources fournies

Vous placerez dans Ressources/ les sources fournies par l'enseignant que l'énoncé vous demandera de télécharger.

Par exemple, **téléchargez dès maintenant les fichiers configbits.h et template.c** et placez-les dans Ressources/.

- Le fichier configbits.h contient les directives #pragma qui paramètreront les bits de configuration de votre microcontrôleur. Vous devrez **ajouter cette source dans chaque nouveau projet** (voir annexe MPLABX).
- Le fichier template.c vous indique le contenu minimal d'un programme fonctionnel. Vous pouvez noter dès à présent les lignes #include "configbits.h" (le fichier précédemment téléchargé) et #include <xc.h> (qui définit les noms de registres spécifiques au microcontrôleur utilisé).

I.e Rendus

Vous consignerez vos notes et observations **directement dans vos sources** :

- S'il vous est demandé de commenter des résultats, **consignez-vos observations dans un commentaire en en-tête** du fichier contenant votre fonction main.
- Commentez également vos **lignes de configuration de registres**, de manière à justifier vos choix

Il n'est pas demandé de compte-rendu séparé. Les modalités de rendu des TP seront discutées lors de la première séance.

II - Exercices

II.a Hello, World ! (blink)

Dans ce premier exercice, il s'agit de réaliser le « Hello, World ! » du microcontrôleur : faire clignoter des LEDs, ou réaliser le programme « blink ». Ceci vous familiarisera avec la carte Explorer 8, la connexion de ses composants, et la syntaxe de configuration de périphériques simples (GPIO) en C.

- 1) Créer un nouveau projet, nommé « TP1a_helloworld »
- 2) Ecrire un programme permettant d'allumer alternativement les LEDs D1-D4, puis D5-D8, avec une période d'environ une seconde (comme représenté en Fig. 1).
Le délai entre l'allumage des LEDs se fera à l'aide d'une fonction `void delai_approx(void)` que vous écrirez pour réaliser une temporisation logicielle approximative. *Il n'est pas important d'approcher précisément 1 seconde.*
A titre indicatif, le temps d'exécution d'une instruction est de $T_{instr} = \frac{1}{F_{osc}/4} = 0.5 \mu s$
- 3) Compiler et tester votre programme sur la carte Explorer 8.
- 4) Commenter les résultats obtenus.

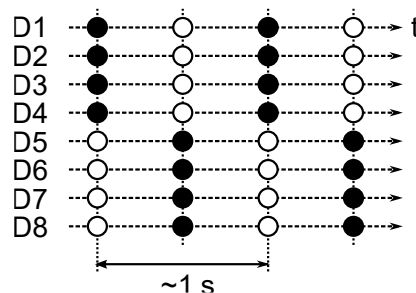


Figure 1: Motif à programmer sur les LEDs dans TP1a_helloworld

II.b Blink - Utilisation du Timer 0 (blink)

Il s'agit maintenant de recréer le motif de l'exercice précédent, cette fois-ci avec une temporisation précise : vous utiliserez des routines de temporisation basées sur un timer pour imposer une période d'exactement une seconde. Vous vous familiariserez cette-fois ci avec la méthode de dimensionnement et de configuration des timers.

Vous utiliserez le Timer 0, dont la documentation est présentée en [1] p. 271. Il se configure à l'aide du registre **OPTION_REG**, explicité en [1] p. 273.

Celui-ci n'a pas de registre dédié à une valeur « seuil » déterminant la période du timer ; vous devrez donc jouer sur la valeur de réinitialisation du timer à chaque overflow. Vous pourrez tester l'expiration du Timer 0 par **scrutation** du bit **TMR0IF**.

- 1) Créer un nouveau projet nommé « TP1b_timer0 »
- 2) Ecrire un programme permettant d'allumer alternativement les LEDs D1-D4, puis D5-D8, avec une période d'exactement une seconde (comme représenté en Fig. 2).
- 3) Compiler et tester votre programme sur la carte Explorer 8. Le motif est conçu pour pouvoir vérifier facilement qu'il clignote à la bonne fréquence.
- 4) Commenter les résultats obtenus.

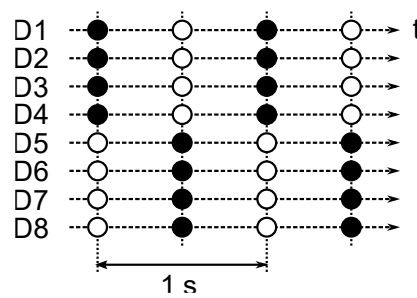


Figure 2: Motif à programmer sur les LEDs dans TP1b_timer2

II.c Utilisation du Timer 2 (chenillard)

Dans cet exercice, vous changerez le motif affiché par les LEDs (de manière à les piloter une par une). En plus de cela, vous utiliserez une source différente de temporisation.

Vous utiliserez le Timer 2, dont la documentation est présentée en [1] p. 285. Il se configure à l'aide du registre **T2CON**, explicité en [1] p. 287.

- 1) Créer un nouveau projet nommé « TP1c_timer2 »
- 2) Ecrire un programme permettant d'allumer les LEDs suivant le motif de chenillard, avec une période d'exactly une seconde (comme représenté en Fig. 3).
- 3) Compiler et tester votre programme sur la carte Explorer 8. Le motif est conçu pour pouvoir vérifier facilement qu'il défile à la bonne fréquence.
- 4) Commenter les résultats obtenus.

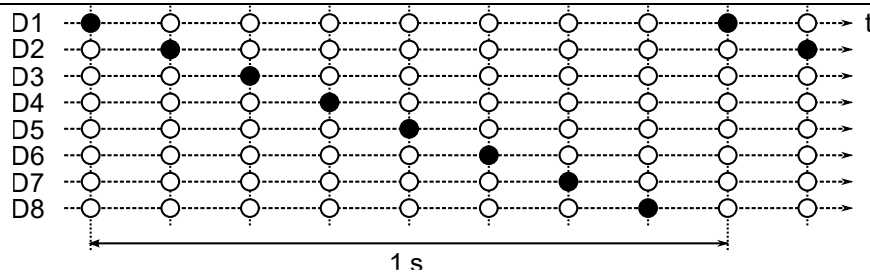


Figure 3: Motif de chenillard à réaliser dans T1Pd_timer2

II.d Utilisation des interruptions (chenillard)

Dans cet exercice, vous réaliserez le même chenillard que précédemment, mais plutôt que d'exploiter le Timer 2 en scrutation, vous utiliserez une **interruption** : tout test de l'interrupt flag du Timer 2 est alors interdit dans la fonction main, et il est possible de faire fonctionner ce programme avec un main ne contenant que :

- L'initialisation des LEDs
- L'initialisation du Timer 2
- L'initialisation des interruptions
- Une boucle `while(1) { /* vide */ }`

Le fonctionnement des interruptions est documenté en [1] p. 87. Elles sont paramétrées par le registre **INTCON**. Vous devrez également activer la sensibilité des interruptions au Timer 2 dans l'un des

registres **PIEx**. La syntaxe de déclaration d'une routine d'interruption (isr) est décrite dans [3] p. 200 (plus spécifiquement, un exemple de la déclaration d'une isr utilisant le prototype `void __interrupt() isr (void)` est donné en [3] p.203 ; le PIC16F1719 n'ayant qu'un vecteur d'interruption, l'appel à `__interrupt()` sera laissé vide).

- 1) Créer un nouveau projet nommé « TP1d_interrupt »
- 2) Ecrire un programme permettant d'allumer les LEDs suivant le motif de chenillard, avec une période d'exactly une seconde (comme représenté en Fig. 3) utilisant une interruption sur TMR2IF.
- 3) Compiler et tester votre programme sur la carte Explorer 8. Le motif est conçu pour pouvoir vérifier facilement qu'il défile à la bonne fréquence.
- 4) Commenter les résultats obtenus.

II.e Bonus (1/2) : Cohabitation d'interruptions et de routines lentes

On essaiera dans ce premier exercice supplémentaire de mettre en évidence les avantages des interruptions en introduisant un programme « lent » en plus des routines d'interruption.

- 1) Créer un nouveau projet nommé « TP1e_bonus1 », dans lequel vous dupliquerez le programme précédent.
- 2) Modifiez le programme pour que :
 - Le chenillard utilisant les interruptions sur le Timer 2 défile sur les LEDs D1 à D4.
 - Un second chenillard défile en sens inverse sur les LEDs D8 à D5, celui-ci temporisé comme dans le premier exercice (c'est-à-dire sans timer ni interruption). Ce second chenillard sera volontairement programmé pour être lent, afin de représenter un programme exécutant beaucoup d'instructions dans la boucle main.
- 3) Compiler et tester votre programme sur la carte Explorer 8.
- 4) Commenter les avantages des interruptions sur le développement d'un tel programme.

II.f Bonus (2/2) : Interruptions multiples

Dans ce deuxième exercice supplémentaire, vous apprendrez à gérer des interruptions multiples. En plus du fonctionnement des chenillards précédents, on souhaite ajouter une routine d'interruption qui stoppe le Timer 2 s'il est en marche (et le relance s'il est arrêté) à l'appui d'un bouton.

Le PIC16F1719 a la particularité de n'avoir qu'un vecteur d'interruption, donc vous devrez implémenter vos routines d'interruption dans une seule ISR et différencier les cas.

Les interruptions déclenchées par les GPIO sont décrites en [1] p. 157. Vous devrez activer les « interrupt on change » par le bit **IOCI**, ainsi que la sensibilité à la broche voulue dans l'un des registres **IOCxP** ou **IOCxN** (selon la détection sur front positif ou négatif, voir [1] p. 159).

- 1) Créer un nouveau projet nommé « TP1e_bonus2 », dans lequel vous dupliquerez le programme précédent.
- 2) Modifiez le programme pour que :
 - L'appui sur le bouton S2 stoppe le Timer 2 s'il est en marche (et le relance s'il est arrêté) dès l'appui sur le bouton
- 3) Compiler et tester votre programme sur la carte Explorer 8.
- 4) Commenter les résultats.

III - Pour la suite ... Nettoyage de code

Maintenant que vous avez une bonne compréhension de la configuration du microcontrôleur et de ses périphériques de base, vous pouvez réunir votre code utile dans des fichiers que vous réutiliserez par la suite par des `#include`. Vous placerez ces fichiers dans votre répertoire `AP4_Circuits_Programmables/Microcontrôleurs/TPs/Ressources/`.

III.a A l'aide de fonctions

Vous pourrez créer des couples de fichiers `.c` et `.h` contenant des fonctions d'initialisation de vos périphériques. Par exemple :

```
// leds.h

#ifndef _LEDS_H
#define _LEDS_H

#include <xc.h>

void init_leds(void);

/* ... Autres fonctions utiles ... */

#endif // _LEDS_H
```

```
// leds.c

#include <xc.h>

void init_leds(void){
    /* ... Code utile ... */
}

/* ... Autres fonctions utiles ... */
```

III.b A l'aide de définitions dans des fichiers d'en-tête

Vous pourrez également définir des « macros » dans des fichiers d'en-tête pour simplifier l'écriture d'instructions répétitives, améliorer la lisibilité du code, et améliorer la facilité de maintenance du code. Par exemple :

```
// leds.h

#ifndef _LEDS_H
#define _LEDS_H

#include <xc.h>

/* Direction des LEDs (eg. LEDS_Dx_DIR = 0) */
#define LEDS_D1_DIR TRISDbits.TRISD0
#define LEDS_D2_DIR TRISDbits.TRISD1
// etc.

/* Etat des LEDs (eg. LEDS_Dx_STATE = 1) */
#define LEDS_D1_STATE LATDbits.LATD0
#define LEDS_D2_STATE LATDbits.LATD1
// etc.

#endif // _LEDS_H
```

Références

[1] Fiche technique de la famille de microcontrôleurs PIC16(L)F1717/8/9

http://ww1.microchip.com/downloads/en/DeviceDoc/PIC16F1717_8_9-data-sheet-40001740C.pdf

[2] Microchip Developer Help

<http://microchipdeveloper.com/8bit:peripherals>

[3] MPLABX XC8 User guide

<https://ww1.microchip.com/downloads/en/devicedoc/50002053g.pdf>

[4] Fiche technique de la carte Microchip Explorer 8

<https://ww1.microchip.com/downloads/en/DeviceDoc/40001812B.pdf>