C :Boucles while, for- hard

Documents fournis	Apports de cours
- Schéma de la maquette - Docs intégrées au logiciel - Programme non	- Structure prog C (sur exemple fourni) - format des variables (type int signé ou non) - incrémentations en C (les 3 syntaxes) - boucles while/for (détails sur exemple fourni) - Commentaires

Important : tout ce que vous faites doit être consigné sur le compte-rendu de TP. C'est le compte-rendu de TP qui sera noté. Coller le texte de TP sur le compte-rendu.

1 Création du projet

1.1 A l'aide du logiciel PSOC CREATOR créer un nouveau projet qui s'appelle **led**. Un fichier source **main.c** s'est automatiquement créé dans la fenêtre **Workspace Explorer** (votre espace de travail ou projet ds d'autres logiciels).

1.2 Ouvrir la feuille de schéma du composant (workspace double click sur TopDesign.cysch)

1.3 Dans la fenêtre **Component Catalog**, dans Ports & Pins, cliquer sur digital output et la faire glisser sur le schéma.

1.4 Dé-valider l'**option HW** (hardware) de la broche, pour le moment on ne la pilote que par logiciel (software).

1.5 Ouvrir **led.cydrw** depuis WorkSpace et à l'aide de la doc du kit, affecter la sortie à une broche reliée à une LED.

1.6 Cliquer sur **generate application** (icône direct ou menu build) pour câbler la sortie
un fichier **pin_1.h** (header) est apparu il fournit le prototype de la fonction à utiliser pour écrire sur la broche : `void Pin_1_Write(uint8 value) ;`
et pour lire ce qu'on y a écrit : `uint8 Pin_1_ReadDataReg(void) ;`

1.7 *Rappels* : void signifie vide ou nul, uint8 : unsigned integer 8 bits (donc un unsigned char) .
Expliquer ce que signifient ces définitions.

1.8 En faire un copié/collé dans main(), ajouter une tempo, remplacer les void par rien et les uint8 par des variables, votre code devrait ressembler à ceci :

```
#include <device.h>
void tempo(uint16 i)
{
      while(i!=0)
      {     i--;
      }
}
void main()
{     uint8 i;
      for(;;)
      {    i=   Pin_1_ReadDataReg() ;
            Pin_1_Write(~i) ;
            tempo(40000);
      }
}
```

1.9 Expliquer chaque ligne de ce programme.

1.10 Vérifier l'absence de pb de syntaxe (compile file) puis appeler le debugger (bug (cafard)ou F5).

2 Tester le projet

2.1 *Remarque* : Si on n'a pas modifié le programme, on peut retourner plus vite ds le debugger (sans reprogrammer) avec alt F5.

2.2 Ce programme en principe ne contient pas d'erreurs => lancer le programme et expliquer en observant la LED le rôle de ce programme.

2.3 Mettre un point d'arrêt dans la boucle while puis continuer en pas à pas en observant l'onglet variables locales. Expliquer le fonctionnement.

2.4 Proposer une méthode pour vérifier le test de la sortie de la boucle while().
2.5 Comment sortir rapidement si on ne souhaite pas vérifier le test (il peut y avoir plusieurs solutions)
2.6 Rajouter des **commentaires utiles** au programme principal
2.7 Si vous êtes en avance tester la tempo fournie par Cypress et l'empilement des fonctions pour raccourcir le code :

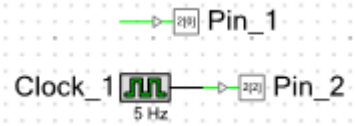
```
void main()
{  for(;;)
   {  Pin_1_Write(~Pin_1_ReadDataReg()) ;
            CyDelay(100);
   }
}
```

Expliquer ces 2 lignes

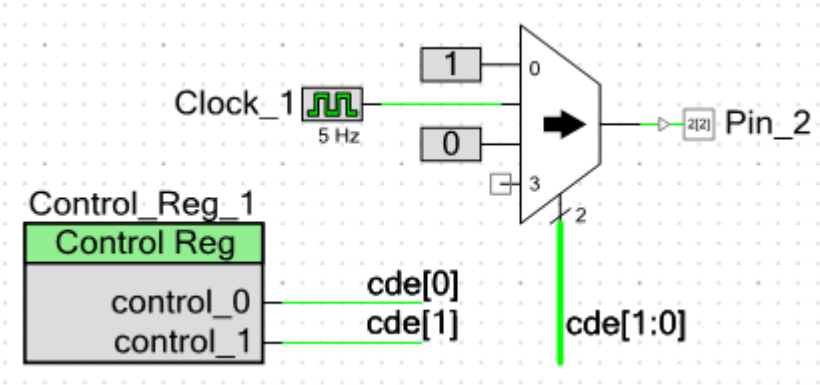
3 Modification Projet

Attention, il s'agit d'un exercice, *le µP ne peut rien faire d'autre pendant la temporisation.* un vrai projet utiliserait plutôt la lecture périodique d'un timer pour pouvoir faire aussi autre chose en attendant.

3.1 Sur le PsoC on aussi la possibilité d'utiliser de la logique câblée
Faire glisser une autre sortie, choisir sa broche de sortie sur une autre led sur le dessin et la câbler avec une horloge, régler l'horloge à 5 Hz .
Observer les leds programme en marche ou arrête et conclure.



3.2 Le programme n'a rien à faire pour que la led pin_2 clignote, mais il ne peut la commander.
On souhaite maintenant pouvoir piloter depuis le programme : Led clignotante, led éteinte, led allumée



3.2.1 Quel état de pin_2 allume la LED justifier
3.2.2 Ajouter au dessin un registre de contrôle, le réduire à 2 bits (double click pour éditer les propriétés d'un objet), ajouter un multiplexeur, les états logiques 0 et 1, ajouter les connections, et les nommer comme sur le dessin. Faire generate application.
3.2.3 Un contol_reg_1.h a été créé, y récupérer : `Control_Reg_1_Write(uint8 control) ;` et l'adapter à votre code.
3.2.4 Quelles sont les valeurs à écrire dans Control_Reg_1 et donner pour chacune l'état correspondant de la led.
3.2.5 Réaliser un programme qui teste ces différentes possibilités. On peut mettre des points d'arrêt pour changer une variable avec le débugger. Expliquer votre test en détail

3.3 S'il vous reste du temps, essayer de faire la même chose avec le registre, une porte, et la sortie en logique 3 états (output enable coché dans les propriétés).