Reprise en main de l'environnement de développement et découverte des bibliothèques.

Le but de ce tutorial est de revoir une application programmée au semestre 1 et de la reprogrammer en utilisant des fonctions de la bibliothèque qui s'appuient sur des structures C. Les programmes vous sont fournis et vous devez les comprendre et mettre des commentaires sur chaque ligne pour expliquer son rôle. Ce fichier commenté sera une partie du travail que vous aurez à rendre.
L'application aura pour but de faire clignoter une led toutes les secondes.

Récupération du projet et des fichiers.

Vous devez tout d'abord avoir sur votre espace personnel un projet uvision pour développer votre code. Les fichiers sont à récupérer sur ogam\ !pedagogie. Vous copierez chez vous le dossier Projet_Simon2018INFO qui est correctement configuré pour permettre le développement pour la carte ciblée. Vous enlèverez les fichiers actuellement associés à ce projet et intégrerez à la place les fichiers pour ce tutorial, disponibles sur moodle et sur ogam. Compilez le projet (build). Vous êtes prêt à étudier maintenant le tutorial.

1 Rappel du principe d'utilisation du timer pour faire clignoter la led .

Revoyez comment fonctionne un timer et les registres qui permettent de le gérer.
On utilisera ici le timer 0 et sa sortie MAT0. La fréquence d'horloge choisie pour le timer 0 est PCLK/4= 25 MHZ.
Si on souhaite que la led reste allumée 0.5 seconde, puis éteinte 0.5 seconde, avec une précision de 1ms sur ce temps, quelles valeurs faut-il mettre dans le prescale register et dans le match value register ?
Comme la led est reliée au port P0.0 et que ce port ne peut pas être piloté par la sortie MAT0 du timer 0, on reliera par un cavalier à l'extérieur le port P1.28 au port P0.0. En effet le port P1.28 peut lui être piloté par MAT0.0.
Les programmations suivantes sont donc nécessaires :
- port P1.28 piloté par MAT0.0 => programmation du registre PINSEL
- port P0.0 en entrée pour ne pas être en sortie et en conflit avec P1.28 qui lui sera connecté
- programmation du timer0 en mode waveform et toogle pour générer le signal MAT0.0.

Tout cela a déjà été vu dans le TD 2 du cours introduction aux micro-controleurs. La seule nouveauté est d'utiliser des bibliothèques pour écrire ces programmes, ce que nous allons détailler maintenant.

2 Compréhension du fichier LPC17xx.h et de la bibliothèque sur le GPIO.

Les composants à base de cœur ARM sont très nombreux, chacun intégrant des périphériques et des mémoires différents, avec son architecture interne particulière (périphériques intégrés, mapping mémoire …).
Afin d'aider les programmeurs à écrire leur code, les constructeurs de ces composants fournissent des fichiers .h qui vont définir un tas de choses très utiles pour le programmeur, moyennant qu'il prenne le temps de comprendre la logique de ces fichiers.

Le fichier LPC17xx.h va contenir la définition de noms symboliques, de type de données, de structures qui permettront ensuite d'accéder aux différents registres du microcontrôleur par des noms symboliques proches de ceux de la documentation technique du microcontrôleur.

2.1 Les adresses de base.

Vous savez que chaque périphérique est géré par des registres, mappés dans l'espace mémoire. Plutôt que d'obliger le programmeur à chercher ces adresses dans la documentation technique du microcontrôleur et de le mettre dans son code, le fichier LPC17xx.h contient les adresses physiques de base des périphériques. Cherchez dans ce fichier ce qui concerne les 5 GPIO leurs noms et leurs valeurs. Vérifiez une dernière fois que cela est correct par rapport au mapping mémoire donné dans la documentation du LPC1768. Ensuite vous prendrez l'habitude de faire confiance à ce fichier et d'utiliser les noms qui y sont définis.

2.2 Les structures de données.

Vous savez qu'un périphérique est géré par un ensemble de bits contenus dans des registres. Ces ensembles de registres vont être déclarés comme des structures C, chaque élément de la structure étant en fait un registre car la structure va être placée précisément à l'adresse des registres du périphérique.
Trouvez dans le fichier LPC17xx.h, la définition d'un type de structure C pour les registres des GPIO et faites le lien avec la question 1.
Toujours dans ce fichier, quel est le rôle de la ligne :
#define LPC_GPIO0 ((LPC_GPIO_TypeDef *) LPC_GPIO0_BASE) ?
Comprenez maintenant les lignes du fichier pinconf.c qui configurent le gpio.

2.3 Utilisation de la bibliothèque de fonctions lpcxx_gpio.c.

Les lignes précédemment utilisées pour programmer les gpio demandent quand même au programmeur de connaitre le nom des registres et leur fonctionnement. Par exemple ici il faut connaitre le nom et le fonctionnement du registre FIODIR pour choisir si un port va être en entrée ou en sortie.
Afin de simplifier encore le travail du programmeur, une bibliothèque de fonctions de gestion du gpio est fournie dans le fichier lpcxx_gpio.c.
Regardez par exemple la fonction GPIO_SetDir. Comprenez précisément ses paramètres, son fonctionnement et comprenez la ligne qui l'utilise dans le fichier pinconf.c.
Prenez conscience du nombre d'instructions générées par l'utilisation des fonctions de bibliothèques en comparant le nombre d'instructions assembleur exécutées avec la première version qui écrit dans la structure associée aux registres et la version qui utilise la fonction gpio_SetDir.

3 Compréhension de l'utilisation d'une structure de configuration sur l'exemple du pinsel.

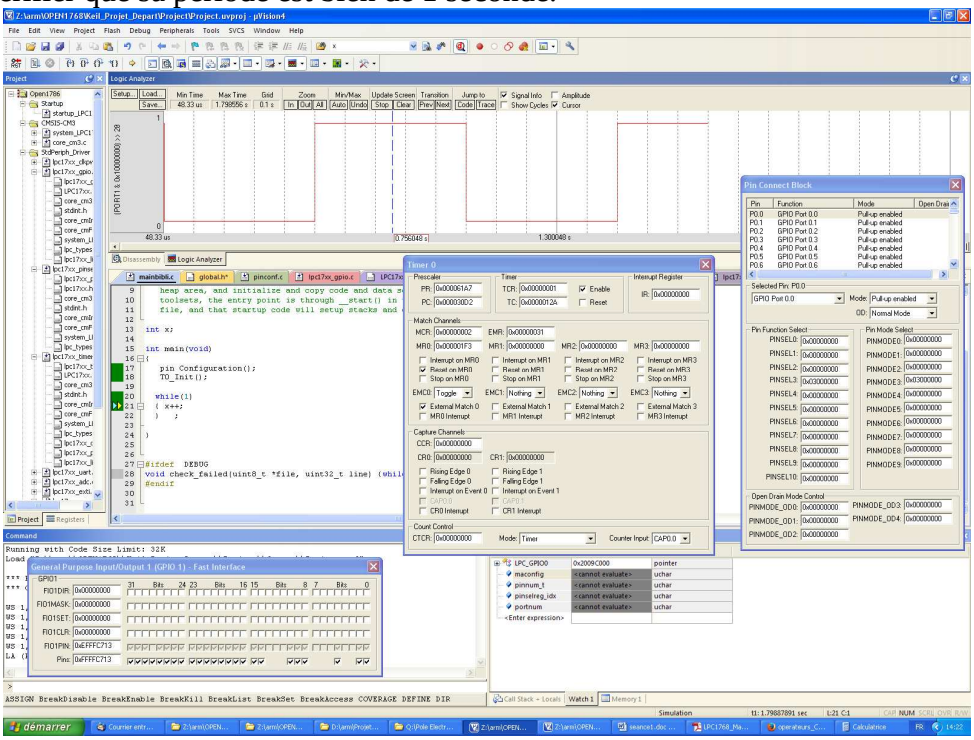
Quand il y a plusieurs registres à programmer pour un périphérique, la bibliothèque propose aussi d'utiliser des structures de configuration.
Le principe est de préparer dans une structure de configuration les valeurs que l'on souhaite programmer dans les différents registres du périphérique, et de passer cette structure en paramètre d'une fonction de configuration. C'est plus rentable puisqu'on ne passe que le pointeur sur la structure.
C'est ce qui est utilisé ici pour la programmation du pinsel.
La ligne « PINSEL_CFG_Type maconfig; » déclare une structure « maconfig » de type structure de configuration d'un pinsel.
Les lignes « maconfig.Portnum = PINSEL_PORT_1;

maconfig.Pinnum =PINSEL_PIN_28; etc »
remplissent les différents champs de cette structure.
Enfin la ligne « PINSEL_ConfigPin(&maconfig); » appelle la fonction de la bibliothèque qui va réellement écrire dans les registres du PINSEL. Notez que l'on passe l'adresse de la structure à cette fonction, puisqu'elle attend un pointeur sur une structure de configuration.
Vous pouvez observer en pas à pas le fonctionnement de cette partie du code. Pour cela, ouvrez la fenêtre permettant de voir le contenu des variables (menu « View » puis « watchs windows/ watch1 ») et entrer « maconfig » sur « enter expression ». Comme maconfig est déclarée en variable locale (puisqu'elle ne sert qu'une fois , pour la configuration), elle ne sera visualisable que quand le code sera dans pin_configuration. Vérifiez le remplissage de la structure de configuration. Pour vérifier l'effet sur le pin connect bloc, ouvrez la fenêtre du périphérique «Pin connect bloc » (menu « Peripherals ») et faite du pas à pas pour voir à quel moment la broche P1.28 est configurée pour être pilotée par le MAT0.0.

4 Structure pour le timer et utilisation de fonctions fournies.

Regardez dans le fichier LPC17xx.h (dont vous connaissez le principe maintenant !), la structure pour gérer les registres d'un timer.
Analysez les fonctions utilisées dans le code pour gérer le timer et commentez les lignes de code une fois que vous les aurez comprises.
Vérifiez en simulation que le code génère bien un signal de 1ms de période sur P1.28. Pour cela, utilisez le logic analyzer :
- menu View -> analysis window -> Logic Analyzer
- dans la fenêtre du logic analyzer, cliquez sur « setup » puis sur New(insert) (petit icône en haut à gauche à côté de l'icône delete)
- entrez le nom du signal à visualiser : PORT1.28
- pensez à mettre le display type sur « bit » puisque c'est un signal logique

Le signal sera visualisé dans la fenêtre quand vous lancerez le RUN. Comme le main est très court, il faudra laisser le programme tourner un certain temps pour visualiser le créneau. Vous pouvez vérifier que sa période est bien de 1 seconde.



Si cela fonctionne en simulation, recompilez en ayant changé la cible (baguette magique « Target option » puis dans l'onglet debbug choisir Use J-LINK/J-Trace Cortex). Branchez la carte, vérifiez que le cavalier est bien présent pour relier P0.0 et P1.28. Téléchargez et constatez avec bonheur le clignotement de la led ☺.

5 Utilisation d'une interruption.

Si on ne souhaite pas faire une connexion extérieure entre la broche MAT 0 et la led, on peut utiliser une interruption timer dans laquelle on va positionner le port qui est relié à la led. Expliquez comment mettre en place une interruption timer et ce qui sera fait dans cette interruption. Utilisez les bibliothèques pour écrire ce code.

Cet exemple vous a permis de découvrir les bibliothèques qui sont à votre disposition pour vous aider à écrire votre code.
Retenez que :

- le fichier LPC17xx.h contient plein de définitions qu'il faut utiliser.
- une bibliothèque, ça s'analyse avant de l'utiliser, car il faut bien comprendre quels sont les paramètres attendus par les fonctions.
- Une bibliothèque n'affranchit pas de devoir comprendre comment fonctionne le périphérique, mais elle facilite l'écriture du code
- l'utilisation de certaines fonctions ajoute un peu inutilement des instructions assembleur et l'écriture directe dans les structures des périphériques est quelquefois plus efficace
- un code se valide d'abord en simulation avant d'être testé sur la carte
- on n'attend pas d'avoir écrit un très long code très compliqué avant de le valider en simulation et sur la carte (intégration continue).

Vous devez déposer sur le moodle un compte-rendu synthétique mais avec du contenu au plus tard pour le lundi 12 mars 8h00.
Ce compte-rendu devra vous servir de « pense-bête » à vous-même pour le projet et nous montrer que vous êtes prêts à utiliser efficacement une bibliothèque. Il contiendra également le listing des fichiers C fournis avec un commentaire expliquant chaque ligne de code.
Ce compte-rendu est individuel et fera partie de la notation finale.

Vous devez ensuite commencer à réfléchir à la façon dont vous allez gérer la partie de code qui vous est attribuée et à regarder les bibliothèques associées aux périphériques que vous allez devoir programmer.
Si vous avez avancé là-dessus, vous indiquerez dans votre compte-rendu où vous en êtes sur cette compréhension de votre partie.

Si l'équipe s'est concertée, le chef de projet intègrera au début de son compte-rendu les noms des étudiants de son équipe et sur quelle partie ils travaillent, ainsi qu'un échéancier prévisionnel sur les étapes du développement (versions visées et pour quelles séances).