1 - Prise en main

Objectifs

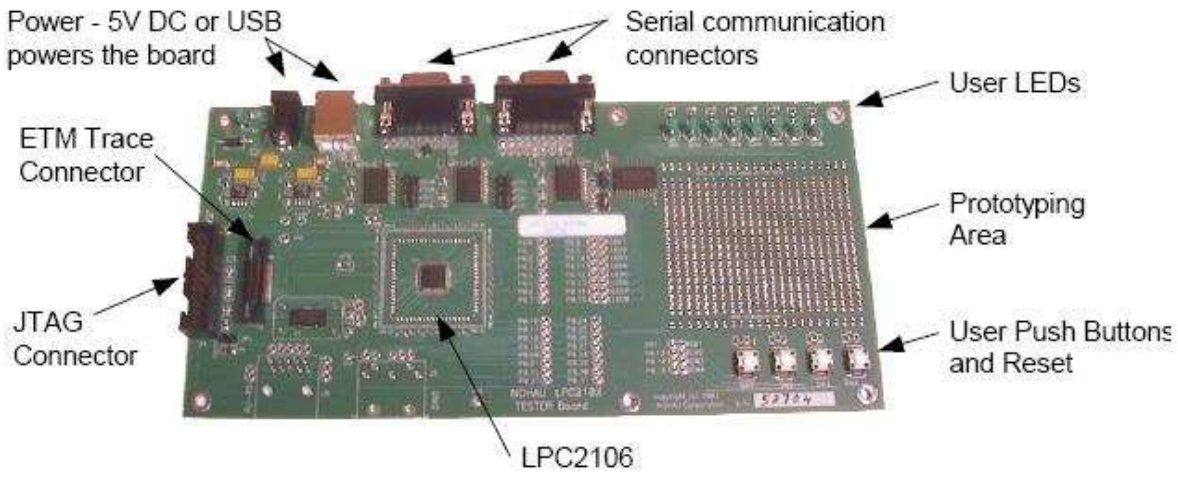
- Le premier objectif est de se familiariser avec les outils de développement et d'émulation du processeur ARM que nous allons utiliser, ainsi que les différentes étapes du processus de compilation et leur utilité.
- Le second objectif est la maîtrise des différentes fonctionnalités à votre disposition qui vous permettront une mise au point rapide et efficace de vos programmes (debug).

1.1 Environnement de travail

L'environnement matériel est composé d'une carte de développement Nohau LPC 2106 qui intègre un cœur processeur ARM7TDMI. Cette carte dispose d'un émulateur intégré In-Circuit qui permet d'assurer des fonctions de mise au point temps réel. Il est ainsi possible de suivre pas à pas l'évolution de code exécutable embarqué et de plusieurs paramètres tels que l'état des registres, de la mémoire, des variables, etc. à chaque stade de l'exécution. Ces fonctions sont possibles grâce à des sondes JTAG qui communiquent les informations de la carte au PC via une interface USB. Seehau est l'interface intuitive, sous Windows, qui permet d'accéder à ces fonctionnalités de debug logiciel.

Les outils à votre disposition comprennent également un environnement intégré de développement pour l'écriture et la mise au point de logiciel embarqué à partir de code C, C++ ou assembleur. Cet environnement (IAR Embedded Workbench IDE 6.10) inclue un compilateur ainsi qu'un simulateur qui permet une vérification rapide du code développé au sein du même environnement (sans nécessité de télécharger un exécutable sur la carte).

La carte LPC 2106 est présentée ci-dessous. Elle est construite autour d'un microcontrôleur ARM7TDMI-S (60 MHz sous 1.8V) et intègre plusieurs périphériques sur la même puce (GPIO, 2 UARTs, 2 timers, PWM, etc.). La carte comprend également trois boutons poussoirs, huit LEDs, deux connecteurs RS232, de la mémoire Flash programmable ainsi qu'une zone de prototypage. Le manuel utilisateur et les fichiers de configuration nécessaires à la programmation du microcontrôleur et de ses périphériques sont disponibles à l'URL http://elec.polytech.unice.fr/~bilavarn/ (login : elec3, mdp : nios2007)



Instructions pour les séances de TP

A chaque séance, vous vous connecterez avec le login elec3 et travaillerez à l'emplacement *C:\Elec3\TP_ARM* où vous créerez un répertoire *tpX_nom1_nom2*. Vous y placerez également un fichier de compte-rendu que vous appellerez tp1_nom1_nom2.doc indiquant vos noms, prénoms.

Compte-rendu : Votre compréhension des séances de TP est évaluée sur la base de compte-rendus. Ils ont également pour but de vous familiariser avec les contraintes de communication et de documentation qui sont un aspect important de l'organisation du travail d'ingénieur. Le soin avec lequel vous le rédigez est important et sera pris en compte dans votre évaluation. Vous devez répondre aux questions qui vous sont posées en incluant quelques explications et une conclusion qui doivent refléter votre compréhension des TP.

Précaution : Toute manipulation sur la cible (mise en place des câbles, etc.) doit s'effectuer hors tension pour ne pas risquer d'endommager la carte.

Dans ce TP, nous allons aborder sur des exemples deux des possibilités les plus courantes de développement de code embarqué : à partir de code assembleur ou à partir de code C. Vous créerez deux répertoires *tp1_asm* et *tp1_c*. Ensuite vous suivrez les instructions décrites au long de ce tutorial en répondant aux différentes questions qui vous sont posées dans votre compte-rendu.

1.2 Développement de code assembleur

Cette première partie a pour objectif de vous familiariser avec l'environnement de développement IAR IDE (Integrated Development Environment).

1.2.1 Saisie et compilation

Le développement de code embarqué pour la carte cible nécessite quelques étapes préalables liées à l'environnement logiciel (création et organisation du projet) et matériel (configuration des options pour la carte cible).

Création d'un projet :

- Ouvrir IAR IDE et sélectionner *Project -> Create new project*.
- Sélectionner *asm* comme type de projet indiquant une entrée en assembleur pur.
- Sauvegarder le projet dans le répertoire *C:\Elec3\TP_ARM\tp1_nom1_nom2\tp1_asm* sous le nom *tp1_asm*.

a) Décrire les répertoires / fichiers créés et leur contenu.

Configuration des options du projet :

Un grand nombre de cartes peuvent être ciblées avec cet environnement. Les opérations décrites dans ce qui suit servent à configurer l'environnement IAR spécifiquement pour la carte LPC2106.

- Dans le menu *Projet / Options*, configurer les options suivantes :
  - *General Options/Target*
    - *Processor variant* : Core ARM7TDMI
  - *C/C++ Compiler/Code*

        - *Processor mode*: Arm
    - *Linker* (éditeur de liens)
        - *Config*: dans *Linker configuration file*, cocher *Override default* et sélectionner le fichier *evba7.icf* (que vous devez télécharger et placer par exemple dans *C:\Elec3\TP_ARM*)

La commande *Compile* permet de compiler un seul fichier. Un projet est toujours constitué de plusieurs fichiers qu'il faut prendre en compte globalement dans la construction du projet. Cette étape s'appelle l'édition de liens et peut être invoquée par la commande *Make*.

b) Lancer la commande *Make* (F7) et décrire les nouveaux répertoires / fichiers créés. En particulier, comment s'appelle le fichier exécutable produit?

Saisie de code assembleur :

Dans le fichier assembleur créé par défaut (*asm.s*), ajouter le code suivant :

```
;-------------------------------------------------------------------
; Nom: main.s
; Auteur(s): …
; Version: 1.0
; Description: programme principal du calcul max min
; Fonctions: main
;-------------------------------------------------------------------
          NAME    main

          PUBLIC  __iar_program_start

          SECTION .intvec : CODE (2)
          CODE32
          EXTERN min
          EXTERN max
__iar_program_start
          B       main


          SECTION .text : CODE (2)
          CODE32

main:
          MOV     R2, #3
          MOV     R1, #4
          BL      max
          MOV     R2, #5
          MOV     R1, #4
          BL      min
exit:     B       exit

          END
```

- Sauvegarder ce programme sous le nom *main.s*, remplacer le fichier *asm.s* par ce nouveau fichier dans le projet (menu *Projet – Remove / Add Files …*).
- *Compiler* ce programme, puis lancer la commande *Make*.

c) Quels sont les erreurs renvoyées dans la construction du projet, et pourquoi ?

d) En cherchant dans le *Assembler Reference Guide* (rubrique *Help*), expliquer l'utilité des directives *PUBLIC* et *EXTERN*.

- Créer deux autres fichiers que vous appellerez *max.s* et *min.s* et les ajouter au projet. Ecrire le code assembleur suivant dans les fichiers respectifs et reconstruire le projet:

```
;-------------------------------------------------------------
; Nom : max.s
; Auteur(s): …
; Version: 1.0
; Description : fonction de calcul du maximum
;-------------------------------------------------------------

                MODULE          max
                PUBLIC          max
                SECTION         MYCODE : CODE (2)
                CODE32
max:            CMP             R1, R2
                BLT             endmax
                MOV             R1, R2
endmax:
                MOV             PC, LR

                END
```

```
;-------------------------------------------------------------
; Nom : min.s
; Auteur(s): …
; Version: 1.0
; Description : fonction de calcul du minimum
;-------------------------------------------------------------

                MODULE          min
                PUBLIC          min
                SECTION         MYCODE : CODE (2)
                CODE32
min:            CMP             R2, R1
                BGT             endmin
                MOV             R1, R2
endmin:
                MOV             PC, LR

                END
```

e) Décrire la structure et les fonctions du programme. Quel type de passage de paramètre est utilisé ? Quels sont les registres d'entrée et de sortie de chaque fonction ?

f) Ajouter une version commentée des programmes assembleur dans votre compte-rendu.

L'exemple précédent illustre bien le cas d'un projet contenant plusieurs fichiers. Un projet de manière générale inclue des fichiers source ainsi que des librairies et la configuration du processeur cible. Ces informations permettent entre autre au compilateur de répartir correctement en mémoire les différentes parties du programmes (modules, zone de code, zone de données, constantes) dans les différents segments mémoire réservés en fonction de leur type (ROM, RAM, table d'interruption, zone réservée à la pile).

g) Dans les options du projet (catégorie *Linker*), chercher et cocher la case *Generate linker map file*. Reconstruire le projet, ouvrir et décrire le nouveau fichier généré. Donner les emplacements en mémoire des segments *main*, *max* et *min*.

1.2.2 Simulation

La mise au point des programmes constitue une étape délicate du développement. Celle-ci peut se baser dans un premier temps sur l'utilisation du simulateur intégré à l'IDE, ce qui présente l'avantage d'être rapide puisqu'il n'est pas nécessaire de télécharger de code exécutable sur la carte. Néanmoins, le code est *simulé* par un processeur (celui du PC) qui est différent du processeur cible (on parle de *cross-développement*). Cela implique que la simulation comporte certaines limites, en particulier pour le développement de code faisant intervenir des périphériques et/ou interruptions. Dans ce cas l'utilisation de l'émulateur (*Seehau*) pour la mise au point des programmes est inévitable. Dans tous les cas, la validation définitive de code développé doit être réalisée par la vérification de l'exécution correcte du programme sur la carte de développement (Emulation).

Simulation

Le simulateur permet l'exécution en mode pas à pas et l'utilisation de points d'arrêt. Il est également possible de visualiser différentes informations tels que l'état des registres et de la mémoire, ou encore d'en modifier le contenu en cours d'exécution. Pour lancer la simulation, aller dans le menu *Project* / *Debug without Downloading*.

Si vous lancez une simulation telle quel, le programme s'arrête à la première instruction. Pour effectuer une simulation pas à pas ou utiliser des points d'arrêts, il faut aller dans le menu *Debug*.

Testez les différents modes pas à pas ainsi que l'utilisation des points d'arrêts. Visualiser les registres (menu *View / Register*), vérifier les résultats, trouver et corriger l'erreur du programme.

h) Ajouter une nouvelle fonction *mean* qui calcule la valeur moyenne de deux entiers. Donner le code assembleur et le résultat du calcul de la moyenne de 532 et 651. Quels sont les problèmes rencontrés ? Pourquoi ?

1.2.3 Emulation sur la carte LPC2106

Nous allons maintenant faire la même chose que précédemment, mais cette fois en exécutant le code réellement sur la carte. Pour cela nous allons utiliser un autre logiciel appelé émulateur (Seehau). Connecter la carte LPC2106 comme suit :

- Brancher le boitier EMUL-PC / JTAG-USB sur le connecteur adéquat de la carte.
- Brancher le câble USB sur le boitier EMUL-PC / JTAG-USB d'un côté et sur le connecteur USB du PC (connecteur face avant en haut de votre PC hôte).
- Brancher l'alimentation sur la carte.
- Ouvrir Seehau. Si le logiciel ne détecte pas la connexion avec la carte, il se referme automatiquement.

i) Télécharger l'exécutable (généré précédemment par la compilation IAR) sur la carte LPC2106 par le menu Seehau *File* / *Load Code*. Vous avez également accès à des

outils de debug accessibles par le menu *Run*. Vérifiez que les résultats du programme sont bien mêmes que ceux obtenus à la question précédente et donnez une conclusion.

Précaution : Lorsque vous quittez l'émulateur Seehau, ne pas sauvegarder la configuration.

1.3 Développement à partir de code C

Le développement de code embarqué peut également se faire à partir du langage C/C++. Le développement à partir d'un langage de haut niveau présente l'avantage d'être plus « portable » (réutilisable) par rapport à du code assembleur. Cette approche est donc très courante dans l'industrie. Dans tous les prochains TPs, la programmation de la carte se fera à partir de code C.

1.3.1 Simulation

Il faut comme précédemment créer un projet et configurer correctement le compilateur IAR, cette fois pour une entrée de type C.

- Créer un nouveau projet (menu *Project/Create New Project*). Sélectionner une entrée de type C (pas C++) et sauvegarder le projet dans le répertoire *C:\Elec3\TP_ARM\tp1_nom1_nom2\tp1_c* sous le nom *tp1_c*.
- Configurer les options du projet comme précédemment :
  - *General Options/Target*
    - *Processor variant* : Core ARM7TDMI
  - *C/C++ Compiler/Code*
    - *Processor mode*: Arm
  - *Linker* (éditeur de liens)
    - *Config*: dans *Linker configuration file*, cocher *Override default* et sélectionner le fichier *evba7.icf* (que vous devez télécharger et placer par exemple dans *C:\Elec3\TP_ARM*)

Compiler le squelette de programme C créé en même temps que le projet pour vérification. Ensuite écrire et compiler le code suivant :

```
//-------------------------------------------------------------
// Nom : Maxmin.c
// Auteur(s): …
// Version: 1.0
// Description : …
// Fonctions : main, min, max
//-------------------------------------------------------------

int max(int x, int y) {
  return (x>y) ?x:y;
}

int min(int x, int y) {
  return (x<y) ?x:y;
}

void main() {
  int a = max(3, 4);
  int b = min(5, 4) ;
```

```
}
```

j) *Simuler* le code en l'exécutant pas à pas et vérifier les résultats.
k) *Emuler* le code en l'exécutant pas à pas et vérifier les résultats.

l) Il est possible de visualiser le code assembleur généré par le compilateur. Chercher et sélectionner cette possibilité dans les options du projet. Quelles sont les différences par rapport au code assembleur de la section 1.2.1 ?

m) Ajouter et tester une fonction *mean* de calcul de la valeur moyenne de deux entiers.

n) Comparez la version assembleur obtenue par compilation du code C avec celui que vous avez écrite précédemment. Quelles sont les principales différences que vous relevez ?