Microcontrôleur

Introduction

L'objectif de cette série de travaux pratiques est de vous faire découvrir l'usage d'un système à microprocesseur 32 bits en mode *bare metal*, c'est à dire sans utiliser de bibliothèque d'abstraction. L'intérêt est double :

- comprendre les mécanismes bas-niveau pour vous permettre un usage plus optimal dans votre vie profesionnelle
- se passer des bibliothèques toutes faites pour pouvoir en être indépendant, mais aussi parce qu'il existe une probabilité non négligeable que vous soyez précisément en charge d'écrite/maintenir ce genre de bilbiothèques.

Pour mener à bien cette série de TP, il vous sera nécessaire d'aller chercher des informations dans différentes datasheet. Cette recherche fait partie du travail d'un concepteur, et donc des objectifs de ces TPs.

1) Environnement de travail

La carte de travail est la BASYS MX3, équipée du microcontroleur PIC32MX370. Ce microcontrôleur est basé sur un processeur 32bits, fonctionne à 120MHz (sur la carte d'utilisation) et possède 128kio de mémoire.

Développée par la société Digilent, cette carte offre des points communs avec les cartes FPGA utilisées à l'ENSEIRB-MATMECA. Notamment :

- Le programmateur est intégré à la carte (ce n'est pas systématique)
- Des composants d'entrées/sorties sont disponibles sur la carte
- Des connecteurs *Pmod* permettent l'utilisation de modules complémentaires.

L'environnement de développement est MPLAB X, fourni par la société Microchip. C'est un environnemnent de développement intégré (IDE). Il permet l'édition des programmes, leur compilation et le debuggage sur la carte.

2) Manipulation des bits en C

La plupart du temps, les paramètres qui permettent de configurer un périphérique sont contrôlés par un sous-ensemble de bits d'un registre donné. Les autres bits du registre codant d'autres paramètres, il faut faire attention à ne les modifier que si nécessaire. C'est pourquoi il est essentiel de savoir manipuler les bits d'un entier individuellement en C.

a) masquage de bits

La première source d'erreur est la confusion entre les numéros de bits et leur position. Comme pour les tableaux en C, les bits sont numérotés à partir de 0. Donc le premier bit est le numéro 0, le 32^e^

bit le numéro 31. De façon générale : le n^{ieme} bit est le bit numéro *n-1*.

Seuls des accès par octets ou par mots sont possibles, donc, pour modifier un seul bit d'un registre, il faut :

- lire la valeur du registre entier,
- modifier le bit correspondant grâce à des opération logiques
- écrire la valeur modifiée du registre entier.

Par exemple, si on veut mettre à 1 le 5e bit d'un registre et à 0 son 7e bit.

- on lit d'abord la valeur du registre que l'on met dans une variable temporaire (tmp)
  tmp vaut : xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx (32 bits de valeur inconnue)
- on modifie le bit numéro 4 (qui est le 5e bit)
  - il faut créer la valeur A = 00000000 00000000 00000000 00010000
  - l'opération bit à bit 'tmp OU A' va donner :
    tmp = xxxxxxxx xxxxxxxx xxxxxxxx xxx1xxxx
- on modifie le bit numéro 6 (qui est le 7e bit)
  - il faut créer la valeur B = 00000000 00000000 00000000 01000000
  - l'opération bit à bit 'tmp ET NON B' va donner :
    tmp = xxxxxxxx xxxxxxxx xxxxxxxx x0x1xxxx
- il ne reste qu'à écrire tmp dans le registre concerné.

A et B sont appelés masques. Ce sont des valeurs exprimées sur 32 bits qui permettent de ne manipuler qu'un bit à la fois. Il est possible de faire un masque pour plusieurs bits à la fois..

b) syntaxe de gestion des masques en C

Il y a deux représentations pour les opérations logiques en C. Chacune de ces représentations code un opérateur différent.

- opérateurs booléens (logiques) : ces opérateurs sont prévus pour agir sur des entiers considérés comme valeurs booléennes (un entier de valeur 0 est interprété comme FAUX et toute autre valeur interprétée comme VRAI). S'ils renvoient FAUX, ces opérateurs renvoient 0, sinon, ils renvoient 1.
  - || représente l'opération OU
  - && représente l'opération ET
  - ! représente l'opération NON

ATTENTION : ces opérateurs ne doivent pas être utilisés avec des masques car ils considèrent les entiers (sur 32 bits) comme une seule valeur.

- opérateurs bits à bits : ces opérateurs sont prévus pour agir sur des entiers considérés comme supports de bits. L'opération est effectuée sur le n^{eme} bit de chaque argument et le résultat est renvoyé dans le n^{eme} bit de l'entier renvoyé.
  - | représente l'opération OU,
  - & représente l'opération ET,
  - ^ représente l'opération OU EXCLUSIF
  - ~ représente le NON

exemple de différence de comportement :

avec un ET bit à bit : 0010 ET 0001 donne 0000
avec un ET logique : 0010 ET 0001 (*VRAI* ET *VRAI*) donne 0001 (*VRAI*)

Le code C de la partie a) est donc :

```
tmp = REGISTRE ;
tmp = tmp | 0x00000010 ;
tmp = tmp & (~ 0x00000040) ;
REGISTRE = tmp ;
```

Pour produire un code plus efficace, il existe deux stratégies :

- REGISTRE = (REGISTRE | 0x00000010) & (~ 0x00000040);
  - on laisse le compilateur gérer la variable tmp, généralement il sait bien faire. :-)

- REGISTRE |= 0x00000010;
  REGISTRE &= ~ 0x00000040;
  - cette version agit directement sur le registre. Inconvénient, les deux bits ne sont pas modifiés au même moment, ce qui peut être gênant dans certains cas.

La meilleure de ces deux stratégies dépend du jeu d'instruction du processeur. Cela ne figure pas parmis les objectif de ce module.

c) <u>création d'un masque</u>

La création d'un masque se fait soit en écrivant une constante (par exemple A= 0x0010) soit en effectuant des décalages de bits (A = 1 << 4).
Les opérateurs << et >> effectuent des décalages de bits vers la gauche ou vers la droite respectivement. Étant donné que la valeur 1 se code en binaire : 0000000000000001 (sur 16 bits par exemple), en effectuant des décalages de la valeur '1' on obtient une valeur binaire ne contenant qu'un seul bit à 1 (les autres bits étants tous à '0').
Par exemple :
(1 << 0) représente un masque dont seul le bit 0 est à 1.
(1 << 2) représente un masque dont seul le bit 2 est à 1 (le troisième bit).
(1 << n) représente un masque dont seul le bit n est à 1 (le $(n+1)^{eme}$ bit).

D'un point de vue mathématique, << est équivalent à une multiplication par une puissance de deux. En effet :

n<<0 = n = $n*2^0$,
1<<3 = 2<<2 = 4<<1 = 8<<0 = 8 = $1*2^3$,
n << m = 2*n << m-1 = … = $n*2^m$;

de même, l'opérateur >> est équivalent à une division par une puissance de deux :

8>>3 = 4 >> 2 = 2 >> 1 = 1 >> 0 = 1;
n >> m = n/2 >> m-1 = … = $n/2^m$;

d) <u>combinaison de masques</u>

Il est possible de créer un nouveau masque de bits en combinant plusieurs autres masques. Par exemple, s'il faut mettre a 1 des bits représentés par les masques A, B et C. Il est plus immédiat d'effectuer une seule opération de masquage avec un masque construit par (A | B | C). L'avantage de cette notation est qu'elle est facile à lire/écrire/comprendre, mais que le calcul du nouveau masque est calculé par le compilateur, et ne consomme donc pas de ressources du microcontrôleur.

e) <u>test d'un unique bit à l'aide d'un masque</u>

Il est souvent nécessaire de tester un bit bien défini dans un entier de 32 bits. Pour cela les masques associés au fonctionnement du C sont d'une grande aide.

Supposons qu'il faille tester le bit 9 de tmp dont la valeur est inconnue : Le masque de ce bit est M=1<<9. L'opération `tmp2 = (tmp & M)` met à 0 tous les autres bits (ceux qui ne nous intéressent pas). *tmp2* est donc nul si et seulement si le bit 9 est nul.

Le test est donc : `if (tmp & M)`

<u>Attention</u> : Il serait tentant d'écrire `if (tmp & 1 << 9)`, mais la priorité des opérateurs en C calculerait d'abord `tmp & 1`, ce qui ne correspond pas à l'opération souhaitée. Pensez toujours à mettre les parenthèses : `if (tmp & (1 << 9))`, vous vous ferez au moins avoir une fois. Cette remarque n'est là que pour vous aider à trouver le problème plus vite.

6) Lexique Anglais/Français pour apprivoiser la datasheet

To set / To assert : mettre à 1 / activer
To reset / To deassert : mettre à zéro / désactiver
a nible : un quartet (4 bits) / un digit hexadécimal
a field : un champ (ensemble de bits dans un registre)

Microprocesseur

TP1 : Prise en main / Entrées-sorties

Cette séance vous permettra de prendre en main le système de façon rapide.

Les étapes de cette séance seront :
- découverte de l'environnement de travail
- écriture de séquences animées sur les sorties (LEDs)
- gestion simple d'entrées/sorties

1) Création d'un projet

Pour lancer MPLAB X, vous pouvez au choix chercher son icône dans le menu (rubrique programmation), ou lancer la commande `mplab_ide &` dans un shell (plus efficace).

L'environnement MPLAB X n'est pas conçu pour simplement compiler un fichier .c. Il contient un ensemble d'outils pour aider à gérer un projet complet contenant plusieurs fichiers sources, ainsi que leurs options de compilation. Les étapes qui suivent sont à reproduire à chaque fois que vous voudrez créer un nouveau projet.

a) Ouverture et choix du type de projet

Pour créer un nouveau projet, sélectionnez dans le menu principal : File > New Project

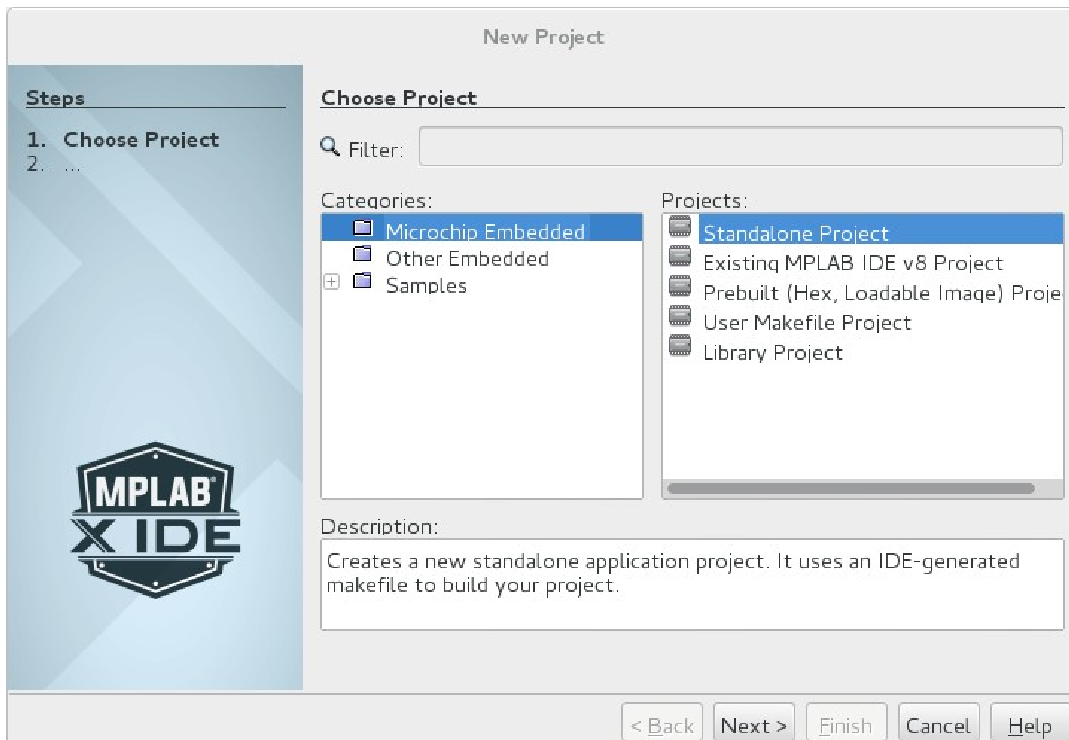

Figure 1: première étape de la création de projet

.

La fenêtre de la figure 1 apparaît, vérifiez que la sélection est bien *Microchip Embedded* pour la catégorie, et *Standalone Project* dans la section de type de projet.

Vous pouvez cliquer sur *Next*.

b) Sélection du microcontrôleur cible

la fenêtre suivante demande le microcontrôleur cible (celui sur lequel on veut faire tourner le projet). Sélectionnez PIC32MX370F512L dans la catégorie *device*.

Remarque : il est beaucoup plus rapide/efficace d'écrire la référence au clavier plutôt que de la rechercher en faisant défilier toutes les références disponibles. De même, sélectionner une famille réduit le nombre de références proposées, mais leur nombre reste trop élevé pour que la recherche par défilement devienne pratique.

Cliquez sur *Next* pour passer à l'étape suivante.

c) Sélection de l'outil de configuration

Comme indiqué dans l'introduction, le boitier de programmation est directement intégré à la carte du microcontrôleur. Il faut donc indiquer que c'est lui que nous allons utiliser.

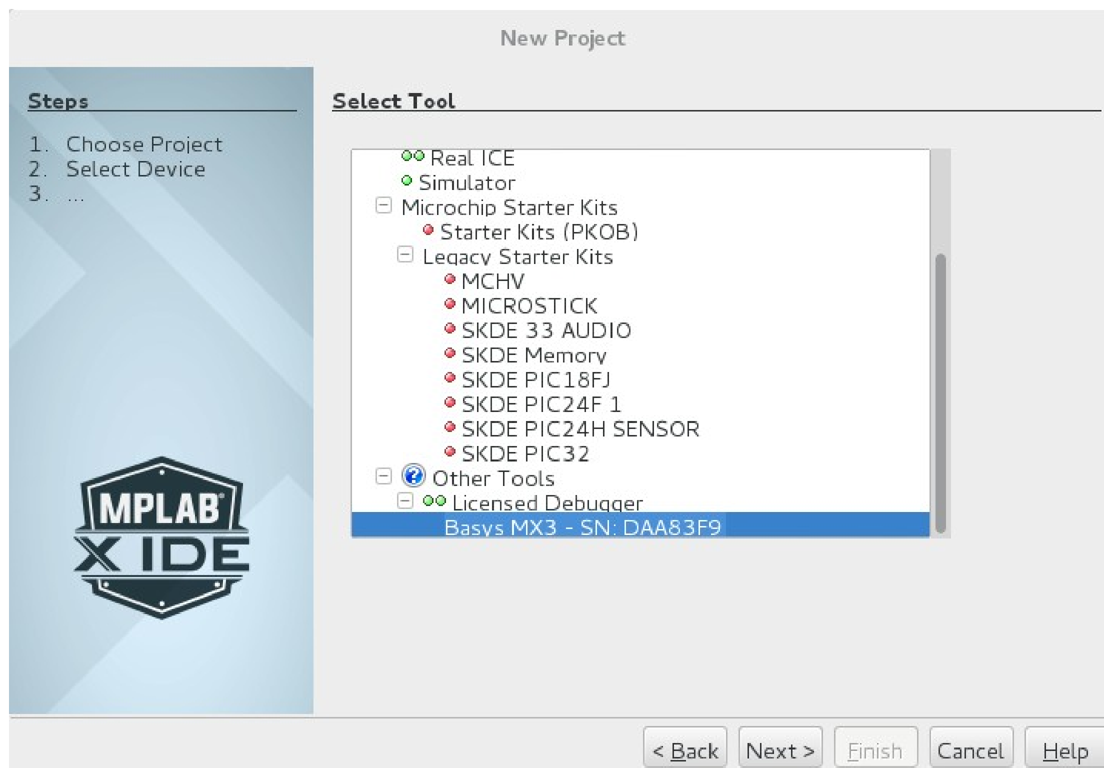

Figure 2: sélection de l'outil de programmation / debuggage de la carte

Comme indiqué sur la figure 2, dans *Select Tool*, sélectionnez la ligne "Basys MX3 – SN. XXXXXX" dans *Other Tools > Licensed debugger* . La ligne se termine par un numéro de série hexadécimal qui est différent pour chaque carte (utile uniquement si deux cartes basys MX3 sont connectées sur le même poste de travail).

Cliquez sur *Next* pour passer à l'étape suivante.

d) Sélection du compilateur

Le compilateur utilisé par MPLAB X n'est pas GCC. En effet, GCC ne permet pas de produire du code pour le processeur qui équipe les séries PIC32. Le compilateur utilisé est xc32.

La fenêtre suivante vous demande préciser le compilateur à utiliser, seul xc32 est disponible. Donc sélectionnez-le, et cliquez sur *Next*.

e) Nom et emplacement

La dernière fenêtre vous demande le nom du projet et son emplacement. Donnez un nom EXPLICITE à votre projet. Si ce n'est pas encore fait, choisissez (créez) un dossier pour contenir vos projets de microcontrôleur (que vous prendrez soin de partager avec votre binôme). La fenêtre *Project Folder* devrait se remplir automatiquement. (Fig 3).

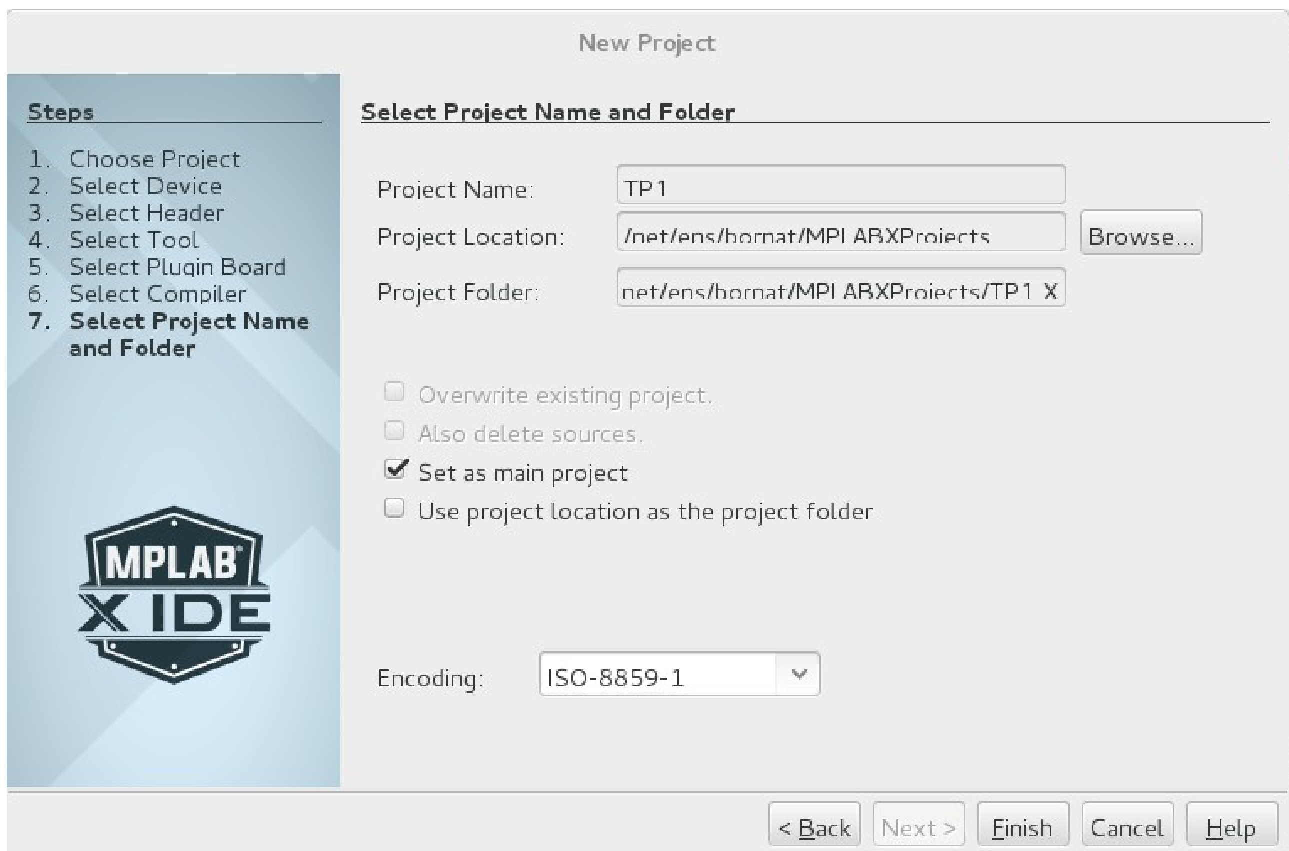

Figure 3: fenêtre de nommage du nouveau projet

L'option *Set as main project* est très utile pour bénéficier des raccourcis icône et clavier.
L'option *Use project location as the project folder* est déconseillée (cela perturbe l'organisation des fichiers quand il y a plusieurs projets)

Vous pouvez cliquer sur *Finish* .

e) Options de compilation

Vous vous retrouvez maintenant dans votre environnement de développement (IDE). Il est composé

de 5 parties principales :

- les menus et icones : tout en haut
- l’espace de code : zone supérieure droite (la plus grande)
- l’espace de navigation de projet:zone supérieure gauche
- les paramètres et ressources du projet : zone inférieure gauche
- la console : zone inférieure droite

Si vous voulez utiliser les avantages de la norme c99, il faut rajouter cette option de compilation. Faites un clic gauche sur le nom de votre projet qui apparaît dans l’espace de navigation de projet (en haut à gauche). Selectionnez *Properties* (l’environnement graphique n’est pas pensé pour un menu aussi long, il sera plus pratique de sélectionner *Properties* avec les flèches du clavier).

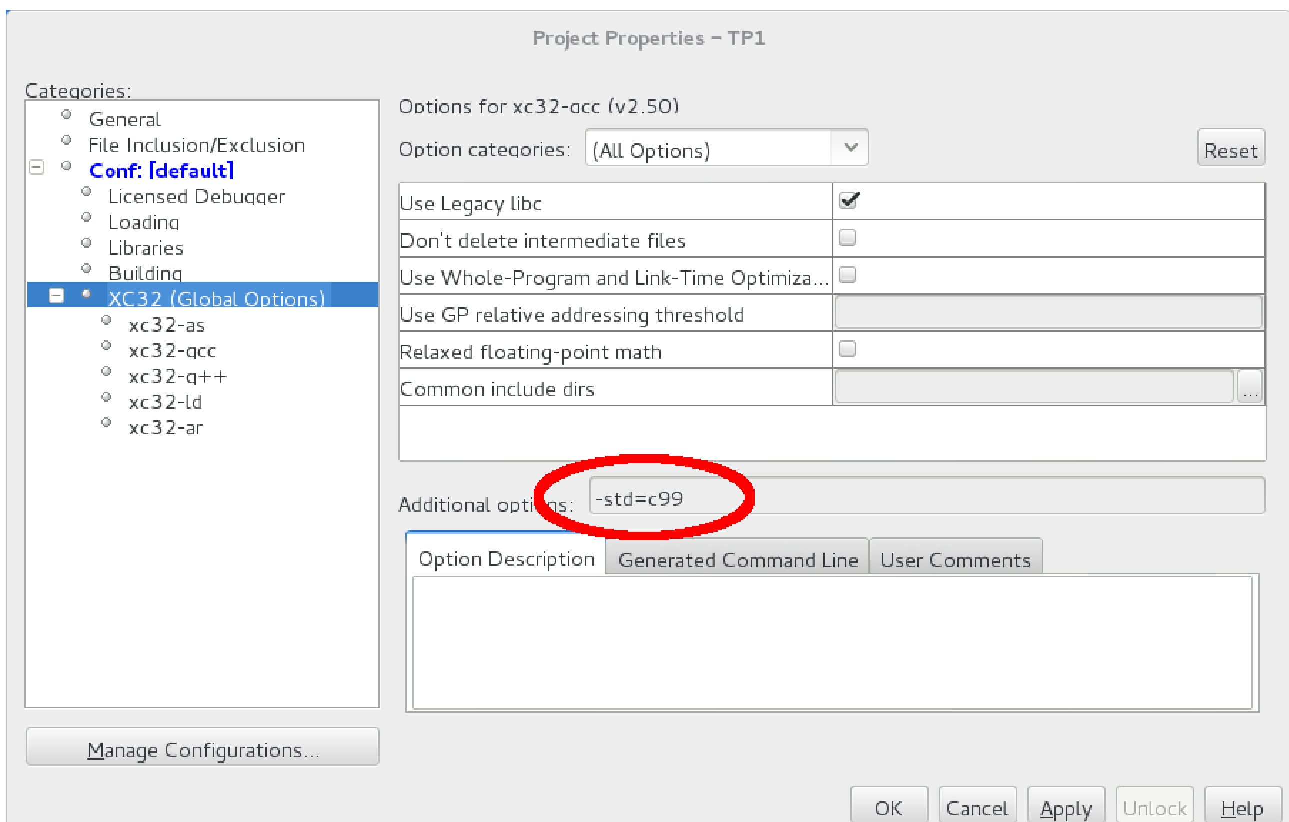

Figure 4: activation de la norme c99 pour le compilateur XC32

Comme indiqué sur la figure 4, dans l’onglet *Conf [defaults] / XC32*, ajoutez l’option `-std=c99` dans la fenêtre *additionnal options*.
Cliquez sur *Apply* pour valider votre modification.

f) Chargement des fichiers de base

En explorant le contenu de votre projet (zone en haut à gauche), vous trouverez toutes les catégories de ressources que vous pouvez inclure. Mais il n’y en a encore aucune. Téléchargez le fichier `config_bits.h` sur le site de ressources en lignes du TP et ajoutez le dans les *Header Files*. Pour cela, cliquez droit sur *Header Files*, puis sélectionnez *Add Existing Item*.

Sélectionnez le fichier `config_bits.h` , vérifiez que les options *Store Path as Relative* et *Copy* sont sélectionnées (sur la partie droite de la fenêtre).

Pour le premier TP, nous allons partir d'un programme déjà fonctionnel : `chenillard.c`. Pour l'ajouter, téléchargez-le, cliquez droit sur *Source Files* et sélectionnez *Add Existing Item*. Il vous suffit alors de le sélectionner. Vérifier encore les options *Store Path as Relative* et *Copy*.

g) Compilation et éxécution

Pour compiler votre projet, clickez sur l'icône représentant un triangle vert dans la partie haute de l'interface (analogue au bouton de lecture sur un lecteur média).
Cette action lance une série d'opérations :

- compilation du programme
- chargement du programme sur la carte
- éxécution du programme

Le programme est stocké de façon non volatile sur la carte (mémoire Flash), il restera donc chargé, même après l'avoir débranchée.

2) Programme de démonstration

Le programme de démonstration montre une animation sur les afficheurs 7 segments. Ces afficheurs sont techniquement un simple assemblage de LEDs. Ils controlés par 8 cathodes (une par segment et le point) et 4 anodes (une par afficheur). Ces 12 signaux sont communs, donc pour afficher des données, il ne faut activer qu'un afficheur à la fois pour ne voir la même valeur s'afficher partour. Pouru afficher une valeur sur les 4 afficheurs, on effectue un balayage rapide, la persistence rétinienne de l'oiel fait le reste. La manipulation des afficheurs 7 segments se fait en logique inversée.

Chez Microchip, les entrées/sorties sont gérées par les *ports*.Chaque port contient jusqu'à 16 entrées sorties (numérotées de 0 à 15), et les ports sont identifiés par une lettre de l'alphabet (de A à G sur le PIC32MX370). Les ports ont tous le même fonctionnement, on utilisera donc la lettre générique 'x' pour les explications concernant les registres. Par exemple, pour définir si une broche doit être utilisée en entrée ou en sortie, on utilise le registre TRISx (TRISA pour le port A, TRISB pour le port B, *etc*). Pour une broche du port x (de 0 à 15), elle sera configurée comme une entrée si le bit correspondant de TRISx est à 1, et en sortie si le bit correspondant de TRISx est à 0.

Finalement, pour chaque broche configurée en sortie, le bit correspondant dans le registre LATx permet de définir quelle sera la valeur de la sortie (plus de détails chapitre 12 de la datasheet du microcontrôleur dans les ressources).

Le tableau 1 montre comment les afficheurs 7-segments sont connectés au microcontrôleur..

Table 1: connections des afficheurs 7 segments

Port	Numéro	fonction		Port	Numéro	Fonction
G	12	Segment A		B	12	Anode 0
A	14	Segment B		B	13	Anode 1
D	6	Segment C		A	9	Anode 2
G	13	Segment D		A	10	Anode 3
G	15	Segment E				

D	7	Segment F				
D	13	Segment G		G	14	point

En lisant le programme de démonstration, on retrouve les fonctionnalités suivantes :

- ligne 1 : inclusion du fichier d'entête (qui en appelle lui-même d'autres)
- lignes 3 à 14 : définition des masques pour manipuler les bits plus lisiblement
- lignes 18 à 20 : fonction d'attente pour ralentir l'animation et la rendre visible (on se contente d'occuper le processeur en faisant une boucle inutile)
- ligne 25:déclaration de la variable qui identifie l'étape du chenillard
- ligne 26 : invalidation des fonctions analogiques sur les broches 12 et 13 du port B[1]
- lignes 27 à 30 : configuration des broches nécessaires en sorties
- ligne 33 : boucle infinie dans laquelle se produit l'animation
    - ligne 35 : appel de la fonction d'attente
    - ligne 36:passage à l'étape suivante
    - lignes 37 à 40 : extinction de tous les afficheurs
    - lignes 42 à 49 : extinction de tous les segments
    - ligne 51 : selon la valeur de fsm
        - allumage de l'afficheur et du segment voulus.

Vous remarquerez que tous les accès aux registres pour piloter les sorties utilisent des masques pour manipuler les données au bit près.

3) Premières modifications

Nous entrons enfin dans la partie active du TP.

Modifiez le programme chenillard.c pour afficher une animation en va-et-vient sur les LEDs (LED0, puis LED1, puis LED2, … puis LED6, puis LED7, puis LED6, puis LED5, …, puis LED1, et retour au début de l'animation).

Le schéma de connection des LEDS est beaucoup plus simple que pour les afficheurs : elles occupent les bits 0 à 7 du port A, la LED i étant connectée au bit i du port A pour i allant de 0 à 7.

4) Lecture d'entrées

Lorsqu'une broche est configurée en entrée (son bit dans TRISx est à '1'), sa valeur est accessible dans le bit qui lui correspond au niveau du registre PORTx.

Modifiez votre programme pour que la vitesse de l'animation change en fonction de la position du switch 0 (connecté en F3) : 0 : vitesse lente, 1 : vitesse rapide.

Modifiez votre programme pour que le sens de rotation du chenillard s'inverse à chaque appui sur le bouton de droite (B8).

1 Certaines broches sont reliées à un système de conversion analogique/numérique en parallèle, il faut désactiver ce système pour ne pas interférer avec les fonctions numériques recherchées. C'est le cas des broches A9, A10, B0 à B15, D1 à D3, E2, E4 à E7, F0 à F8, F12, F13, G6 à G9.