

**MINF**  
**Mise en œuvre**  
**des microcontrôleurs PIC32MX**

# **Chapitre 2**

**Prise en main MPLAB X**  
**avec XC32 et Harmony**

**✕ T.P. PIC32MX**

**Christian HUBER (CHR)**  
**Serge CASTOLDI (SCA)**  
**Version 1.81 novembre 2018**



## CONTENU DU CHAPITRE 2

<b>2. Prise en main MPLAB X avec XC32 et Harmony</b>	<b>2-1</b>
<b>2.1. Désinstallation des anciennes versions</b>	<b>2-1</b>
<b>2.2. Installation des programmes</b>	<b>2-1</b>
2.2.1. Installation de MPLAB X	2-2
2.2.2. Installation du compilateur XC32	2-2
2.2.3. Installation de Harmony	2-3
2.2.4. Intégration de Harmony dans MPLAB X	2-3
2.2.4.1. Preuve de l'intégration	2-5
2.2.5. Sélection version du XC32 dans MPLAB X	2-5
2.2.5.1. Etablissement V1.42 par défaut	2-6
2.2.6. Intégration du BSP pic32mx_skes dans Harmony	2-6
2.2.6.1. Copie du BSP	2-6
2.2.6.2. Mise à jour de la liste des BSP	2-6
<b>2.3. Création d'une application avec MHC</b>	<b>2-7</b>
2.3.1. Création d'un projet Harmony	2-7
2.3.1.1. Choose Project	2-7
2.3.1.2. Name and Location	2-8
2.3.2. Configuration du projet Harmony	2-10
2.3.2.1. BSP Configuration	2-10
2.3.2.2. Device configuration	2-10
2.3.2.3. L'aide du Clock Diagram	2-12
2.3.2.4. Harmony Framework configuration	2-13
2.3.2.5. Génération et sauvegarde de la configuration	2-14
2.3.3. Compléments de configuration	2-15
2.3.3.1. Niveau d'optimisation	2-16
2.3.3.2. Avertissements à la compilation	2-17
2.3.4. Build du projet	2-17
<b>2.4. Adaptation du canevas</b>	<b>2-18</b>
2.4.1. Vérification du BSP	2-18
2.4.1.1. Localisation du BSP dans les source Files	2-18
2.4.1.2. Localisation du BSP dans Header Files	2-18
2.4.1.3. Vérification du chemin include	2-19
2.4.1.4. Control avec Build	2-19
2.4.2. Contenu du BSP	2-19
<b>2.5. Adaptation de l'application</b>	<b>2-20</b>
2.5.1. Contenu des logical folder app	2-20
2.5.1.1. Contenu de main.c	2-20
2.5.2. Contenu du logical folder system_config	2-21
2.5.2.1. Contenu du fichier system_init.c	2-21
2.5.2.2. Contenu du fichier system_interrupt.c	2-21
2.5.2.3. Contenu du fichier system_tasks.c	2-21
2.5.2.4. Contenu du fichier system_exceptions.c	2-21
2.5.3. Situation au niveau system_init.c	2-22
2.5.3.1. Contenu de BSP_Initialize	2-22

2.5.4.	Gestion de la machine d'état de l'application	2-23
2.5.4.1.	Complément de APP_STATE dans app.h	2-23
2.5.4.2.	Ajout d'une fonction de mise à jour de APP_UpdateState	2-23
2.5.4.3.	Mise à jour de la fonction APP_Tasks	2-23
2.5.4.4.	Ajout de la fonction APP_UpdateState dans app.c	2-25
2.5.4.5.	Ajout include dans app.c	2-25
2.5.4.6.	Utilisation de la fonction APP_UpdateState dans system_interrupt.c	2-26
2.5.5.	Application, test & conclusion	2-26
<b>2.6.</b>	<b>Adaptation d'une application exemple</b>	<b>2-27</b>
2.6.1.	Emplacement de la copie de l'exemple	2-27
2.6.2.	Copie de l'exemple adc_pot	2-27
2.6.3.	Ouverture du projet	2-27
2.6.4.	Situation de l'exemple adc_pot	2-28
2.6.5.	Remplacement du BSP	2-28
2.6.5.1.	Suppression du BSP	2-28
2.6.5.2.	Utilisation du BSP du kit ES	2-29
2.6.6.	Vérification du remplacement du BSP	2-29
2.6.6.1.	Situation arborescence BSP	2-29
2.6.7.	Mise en ordre de la configuration	2-30
2.6.7.1.	Ajustement de la configuration	2-30
2.6.7.2.	Nettoyage de l'arborescence du projet	2-30
2.6.8.	Modification de l'Harmony Framework Configuration	2-31
2.6.8.1.	Ajout d'un timer	2-31
2.6.8.2.	Suppression du driver ADC	2-31
2.6.8.3.	Test de compilation	2-32
2.6.8.4.	Vérification pin configuration	2-32
2.6.8.5.	Contrôle device configuration	2-33
2.6.8.6.	Contrôle situation system_init	2-33
2.6.8.7.	Contrôles définitions dans system_config.h	2-34
2.6.9.	Modification de l'application	2-34
2.6.9.1.	Ajout dans app.h	2-34
2.6.9.2.	Ajout/modification dans APP_Tasks	2-35
2.6.9.3.	Ajout dans app.c	2-36
2.6.9.4.	Modification dans system_interrupt	2-37
2.6.9.5.	Test de fonctionnement	2-37
2.6.9.6.	Adaptation exemple, conclusion	2-37
<b>2.7.</b>	<b>Gestion des projets Harmony</b>	<b>2-38</b>
2.7.1.	Sauvegarde d'un projet	2-38
2.7.1.1.	Fonction de package	2-38
2.7.1.2.	Ajout d'un fichier descriptif de projet	2-39
2.7.1.3.	sauvegarde du répertoire du projet	2-40
2.7.1.4.	Sauvegarde du BSP	2-40
2.7.1.5.	Sauvegarde d'un exercice ou TP	2-40
2.7.2.	Restauration d'un projet	2-41
2.7.2.1.	Copie du projet dans l'arborescence harmony	2-41
2.7.2.2.	Copie du BSP	2-41
2.7.2.3.	Restauration d'un exercice ou TP	2-41
2.7.3.	Bsp pic32mx_skis	2-42
2.7.4.	Utilitaires application	2-42
<b>2.8.</b>	<b>Conclusion</b>	<b>2-42</b>
<b>2.9.</b>	<b>Historique des versions</b>	<b>2-43</b>

2.9.1.	V1.0 Janvier 2013	2-43
2.9.2.	V1.5 Septembre 2014	2-43
2.9.3.	V1.6 Septembre 2015	2-43
2.9.4.	V1.7 Novembre 2016	2-43
2.9.5.	V1.8 novembre 2017	2-43
2.9.6.	V1.81 novembre 2018	2-43



## **2. PRISE EN MAIN MPLAB X AVEC XC32 ET HARMONY**

Dans ce chapitre, nous allons étudier comment installer et configurer MPLAB X et son compagnon, le compilateur XC32 de Microchip. Nous allons encore intégrer Harmony à l'environnement MPLAB X.

Le compilateur XC32 offre l'avantage d'être un compilateur C/C++ et d'être basé sur le compilateur gnu.

Les apports importants du software framework Harmony sont notamment le MHC (Microchip Harmony Configurator), qui est un assistant pour la création des projets, ainsi que les différents exemples fonctionnels traitants différents aspects et possibilités du PIC32.

### **2.1. DÉSINSTALLATION DES ANCIENNES VERSIONS**

Il n'est pas indispensable de désinstaller une ancienne version du MPLAB X. On peut la conserver en parallèle, ainsi que les anciennes versions du compilateur XC32.

Si vous changez de version d'Harmony, prenez garde à ne pas effacer l'ancienne, afin de ne pas perdre les projets qui sont dans son arborescence.

### **2.2. INSTALLATION DES PROGRAMMES**

Voici la séquence d'installation :

1. Installation de MPLAB X IDE
2. Installation du compilateur XC32
3. Installation de Harmony
4. Intégration du MHC de Harmony dans MPLAB X
5. Sélection de la version du XC32 dans MPLAB X (optionnel)
6. Intégration du BSP du kit ES dans Harmony (optionnel)

Remarque : Ce chapitre a été réalisé avec les versions suivantes :

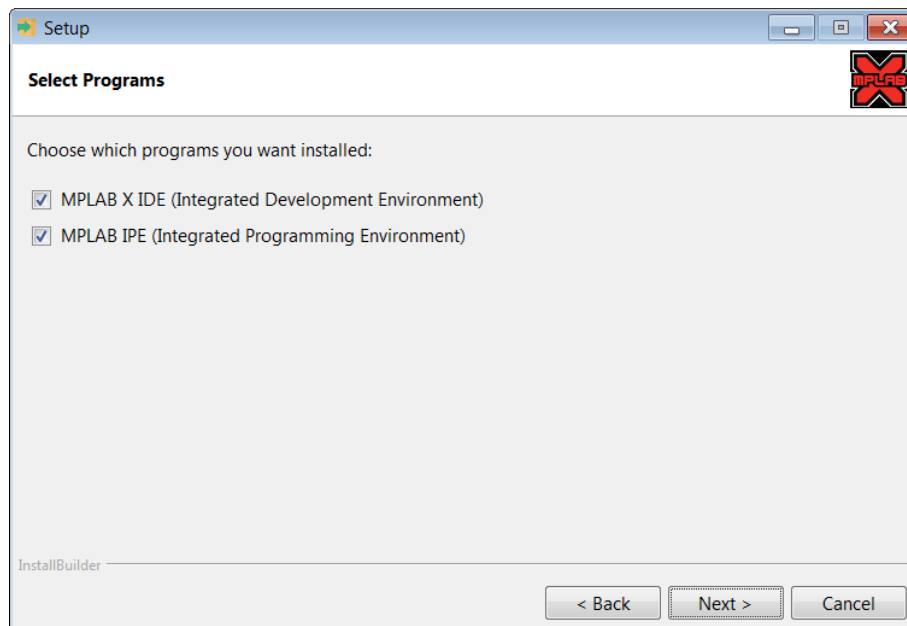
- MPLAB X 3.40
- XC32 1.42
- Harmony 1.08\_01

### 2.2.1. INSTALLATION DE MPLAB X

Le fichier d'installation MPLABX-v3.40-windows-installer.exe, se trouve sous  
...\Maitres-Eleves\Install\PICS\MPLAB\MPLABX\_IDE&XC32\MPLABX\v3\_40

Installation de la version 3.40 de MPLAB X avec les settings par défaut.

Sélectionner aussi l'IPE !

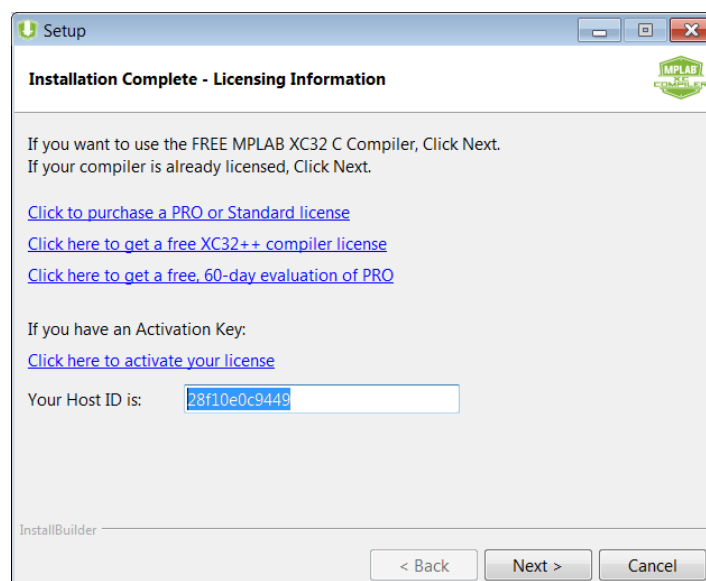


### 2.2.2. INSTALLATION DU COMPILATEUR XC32

Le fichier xc32-v1.42-full-install-windows-installer.exe, se trouve dans le répertoire  
...\Maitres-Eleves\Install\PICS\MPLAB\MPLABX\_IDE&XC32\XC32\v1\_42.

Réalisez l'installation avec les settings par défaut.

Microchip met à disposition une version libre, qui est limitée au niveau de l'optimisation du code.



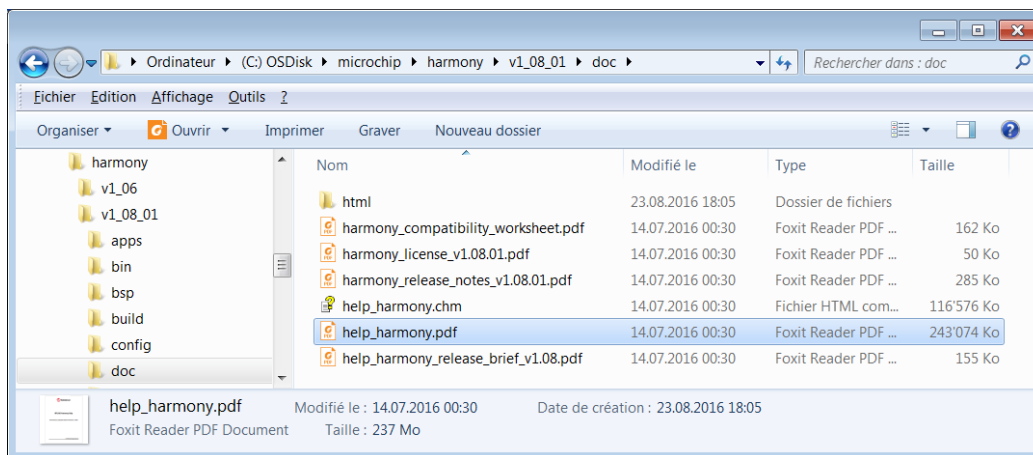


### 2.2.3. INSTALLATION DE HARMONY

Le fichier d'installation harmony\_v1\_08\_01\_windows\_installer.exe, se trouve sous ...\\Maitres-Eleves\\Install\\PICS\\MPLAB\\Harmony\\v1\_08\_01.

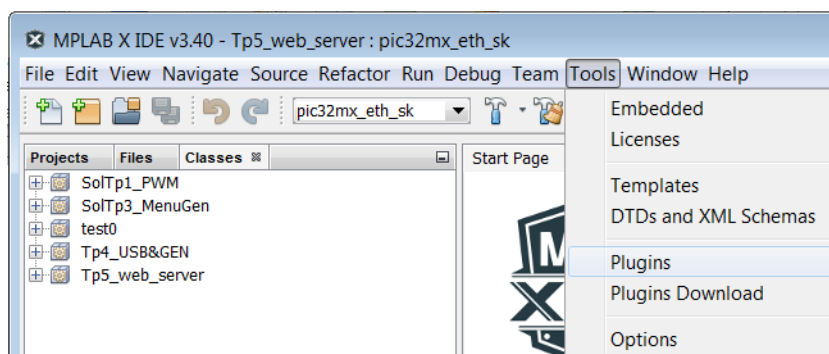
Réalisez l'installation avec les settings par défaut.

☺ La documentation se trouve dans le sous-répertoire "doc" après l'installation.

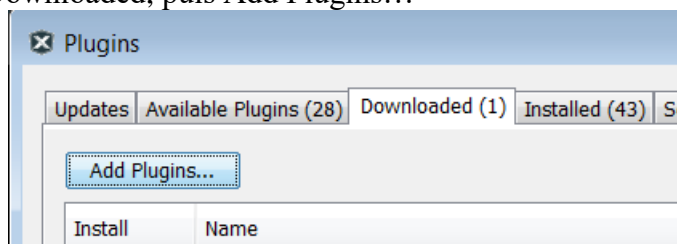


### 2.2.4. INTEGRATION DE HARMONY DANS MPLAB X

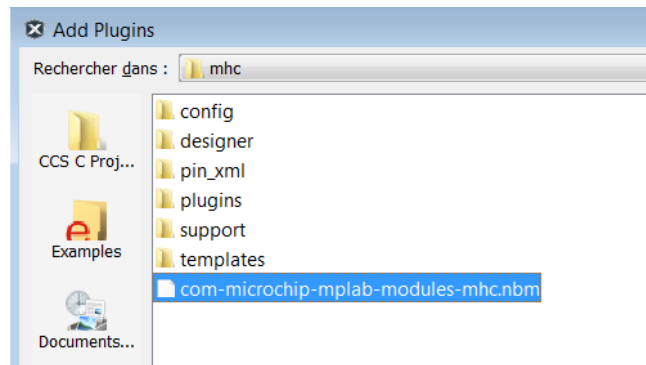
On lance MPLAB X, puis on ouvre l'onglet Tools > Plugins



Ouvrir l'onglet Downloaded, puis Add Plugins...

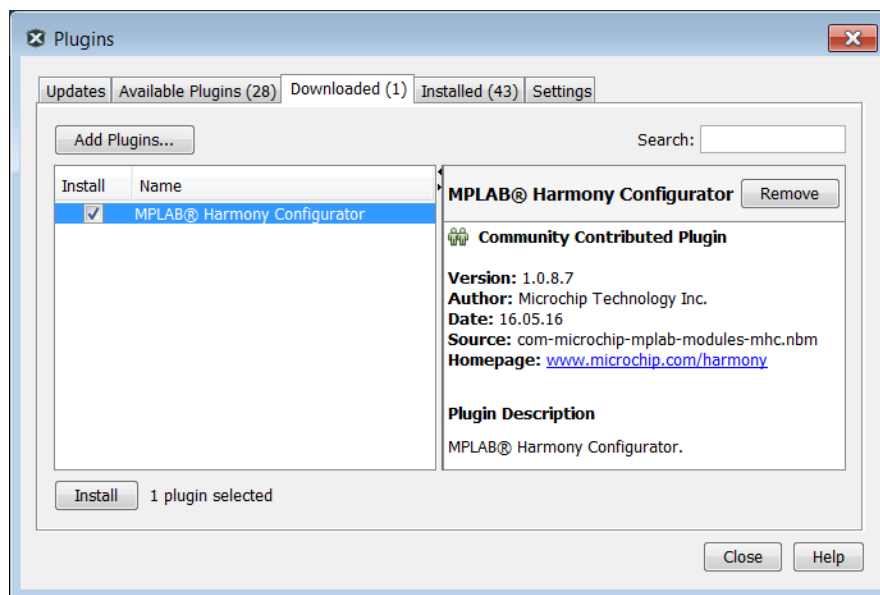


Et aller chercher le fichier ci-dessous :



Ce fichier se trouve sous : <Répertoire Harmony>\v<n>\utilities\mhc

Cliquez sur le bouton Ouvrir et le plugin est sélectionné, prêt à être installé.

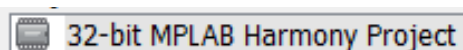


Il faut installer le plugin avec le bouton Install.

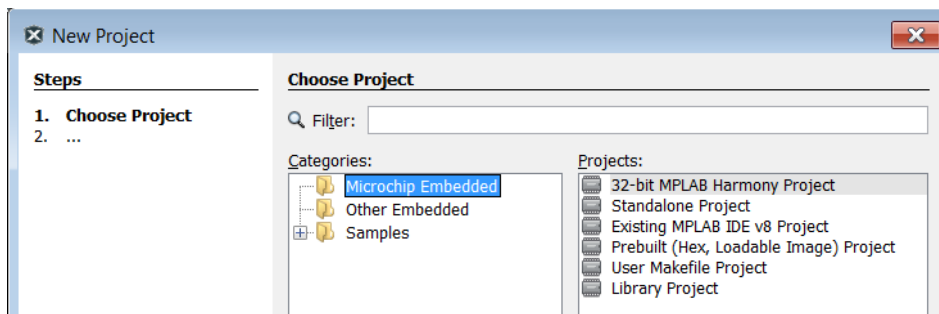
Après l'installation du plugin, il est nécessaire de redémarrer MPLAB X.

### 2.2.4.1. PREUVE DE L'INTEGRATION

Avec File New Project, on doit disposer du type



32-bit MPLAB Harmony Project

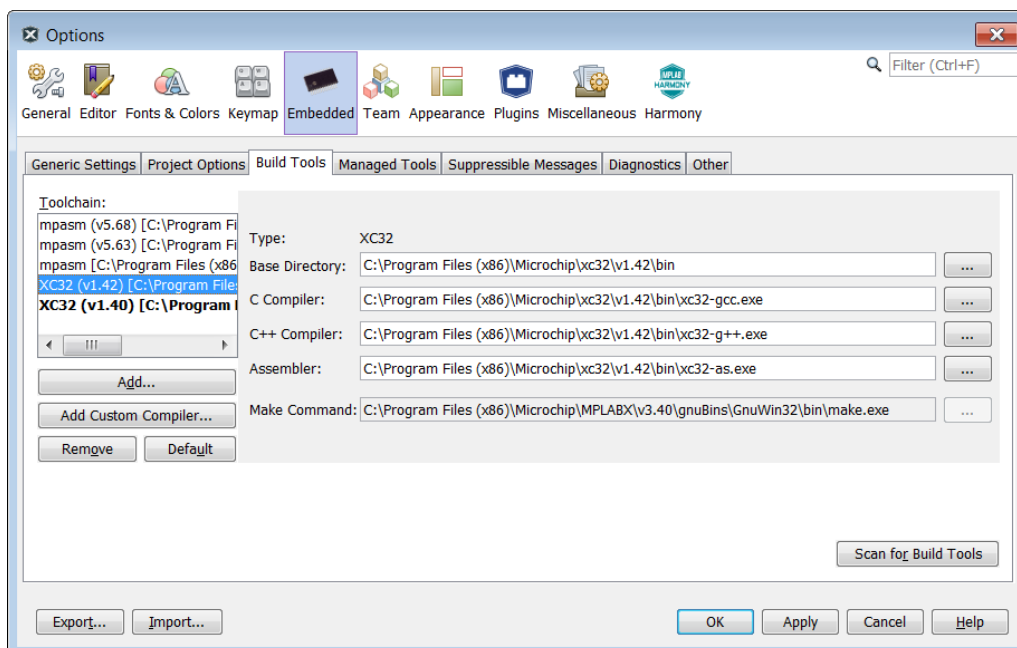


### 2.2.5. SELECTION VERSION DU XC32 DANS MPLAB X

Cette étape n'est nécessaire que si plusieurs versions du compilateur XC32 sont installées.

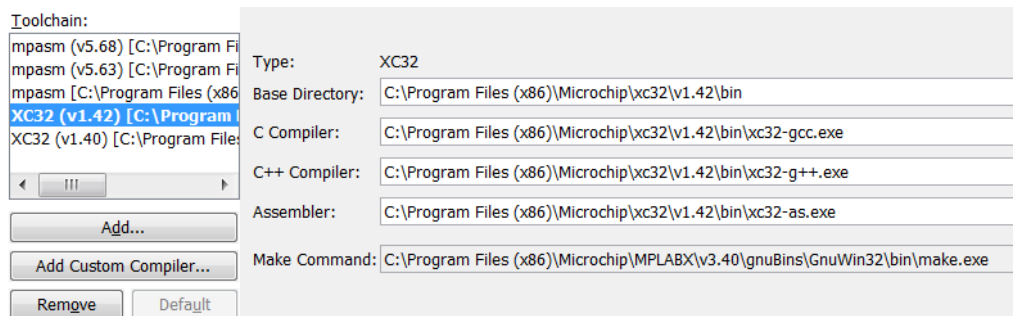
Ouvrir l'onglet Tools, sélectionnez Options. Dans la nouvelle fenêtre qui s'est ouverte sélectionnez Embedded.

Vérifiez que l'on ait bien la présence du XC32 v1.42.



### 2.2.5.1. ETABLISSEMENT V1.42 PAR DEFAULT

Il faut sélectionner comme ci-dessus la version 1.42 et activer le bouton 



### 2.2.6. INTEGRATION DU BSP PIC32MX\_SKES DANS HARMONY

Cette étape n'est nécessaire que si vous travaillez avec le starter-kit ES et souhaitez intégrer son BSP dans Harmony. Les BSP des kits de Microchip sont déjà intégrés dans Harmony. Un BSP vous est fourni pour le starter-kit ES.

#### 2.2.6.1. COPIE DU BSP

Il suffit de copier le répertoire **pic32mx\_skes** que l'on trouve sous :  
 ...\\Maitres-Eleves\\SLO\\Modules\\SL229\_MINF\\Harmony\_ES\\v1\_08  
 dans le répertoire  
 <Répertoire Harmony>\\v<n>\\bsp

#### 2.2.6.2. MISE A JOUR DE LA LISTE DES BSP

Il est nécessaire de remplacer un fichier Harmony. Cela s'effectue en copiant le fichier déjà modifié DS60001156.hconfig de  
 ...\\Maitres-Eleves\\SLO\\Modules\\SL229\_MINF\\Harmony\_ES\\v1\_08\\config  
 dans  
 <Répertoire Harmony>\\v<n>\\bsp\\config

☛ On remplace un fichier unique qui provient de l'installation de Harmony !

## 2.3. CRÉATION D'UNE APPLICATION AVEC MHC

Le MHC (MPLAB Harmony Configurator) permet la création d'une application avec un certain automatisme.

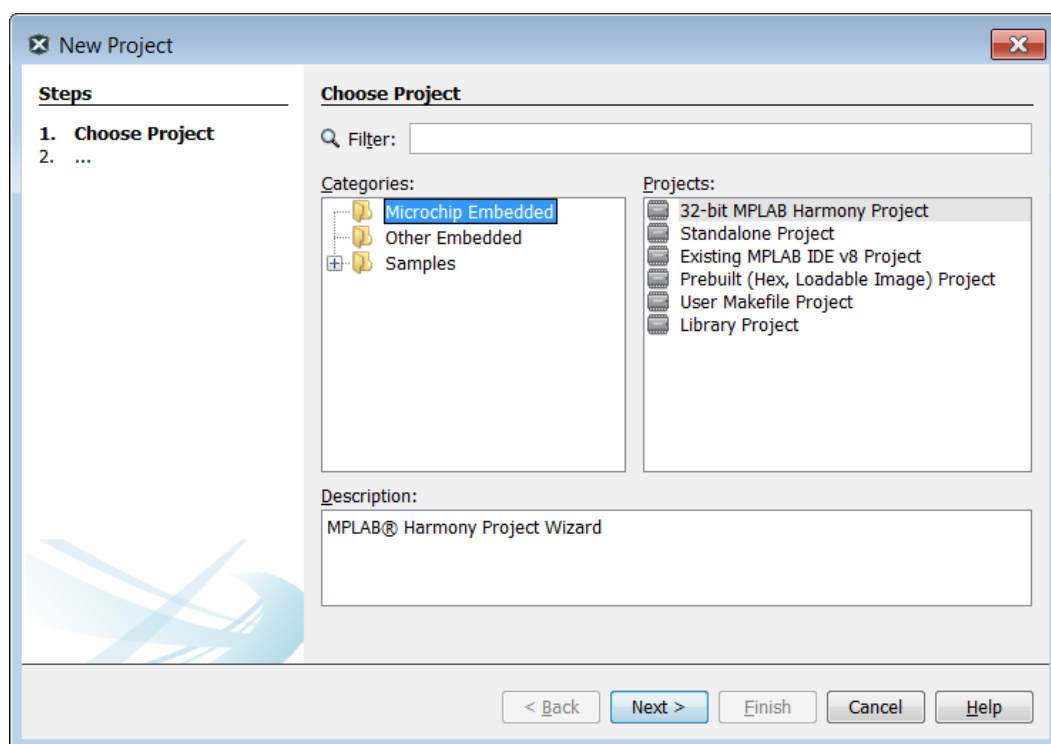
En plus de la création, le MHC permet la configuration graphique des fusibles, le choix d'un BSP et l'ajout de pilotes de périphériques.

Voici les étapes de la réalisation.

### 2.3.1. CRÉATION D'UN PROJET HARMONY

#### 2.3.1.1. CHOOSE PROJECT

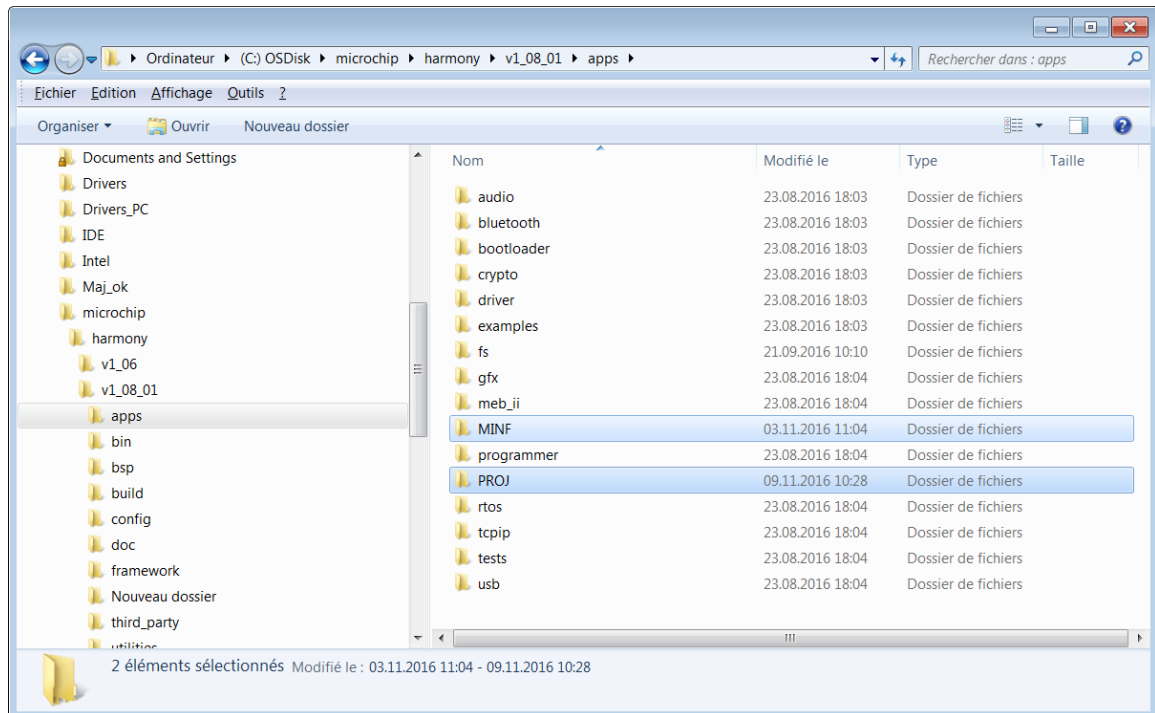
Il s'agit de créer un nouveau projet, mais au lieu de sélectionner un "Standalone Project", on va sélectionner 32 bit MPLAB Harmony Project.



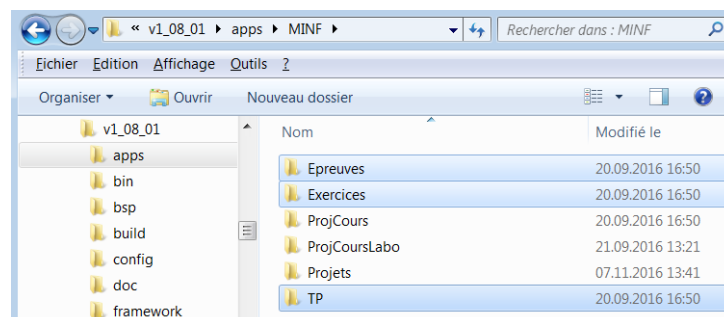
### 2.3.1.2. NAME AND LOCATION

Comme on peut le constater, le projet est créé dans la structure obtenue lors de l'installation de Harmony, ce qui va nous obliger à travailler d'une manière un peu particulière.

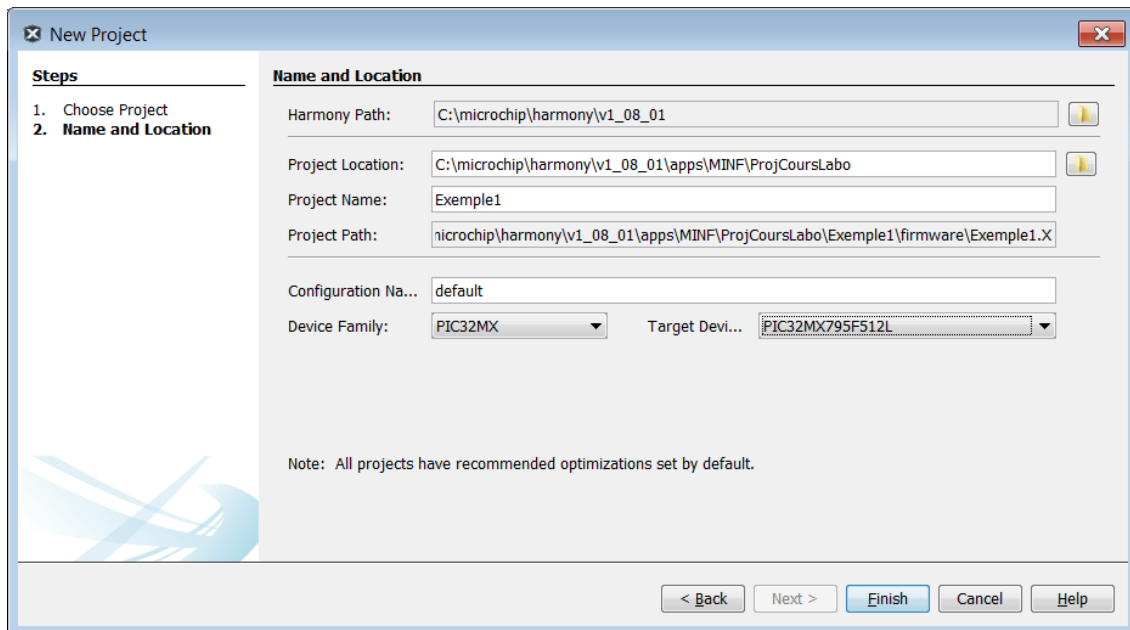
Pour éviter d'avoir trop de fichiers éparpillés sous apps, il est demandé de travailler avec les sous-répertoires MINF et PROJ, comme ci-dessous :



Le répertoire MINF contient les trois sous-répertoires suivants (TP, Exercices et Epreuves) :



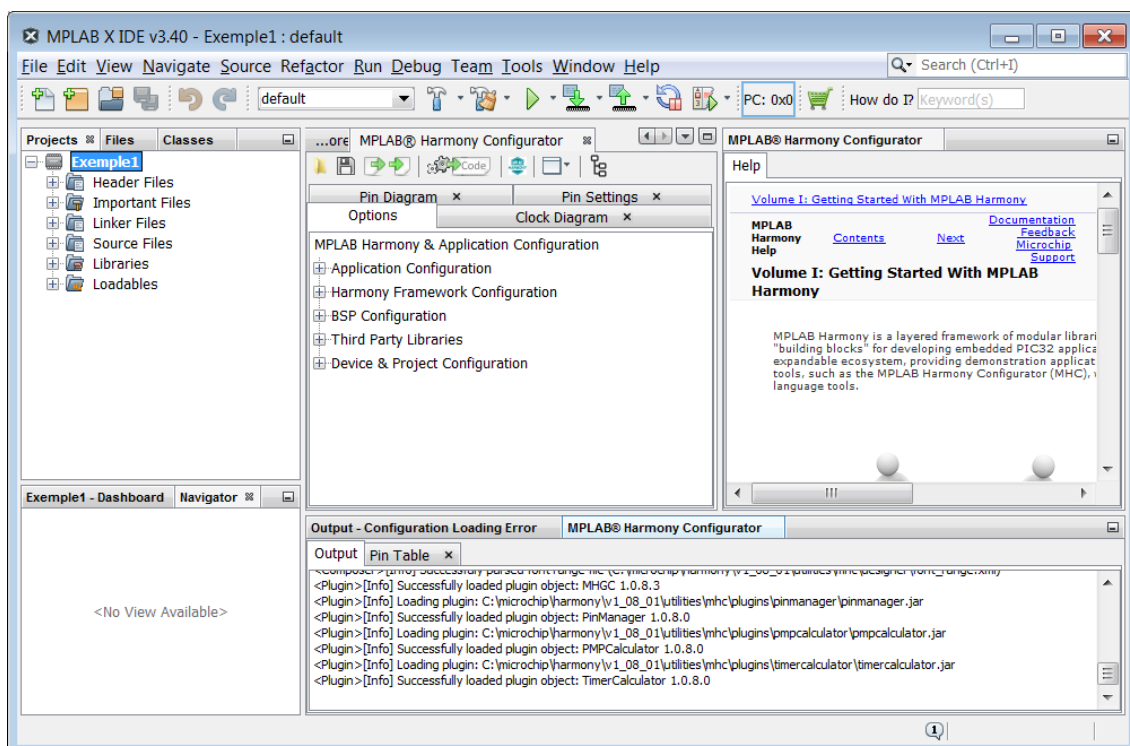
Pour les besoins de la réalisation des chapitres de cours pour le laboratoire, utilisation du répertoire apps\MINF\ProjCoursLabo. D'où :



Ne pas oublier d'indiquer le bon modèle de processeur !

☞ Dans le cas d'un projet de semestre ou de diplôme, faites une copie d'écran de la situation **Name and Location** et placez-la dans le rapport.

Avec Finish, on obtient :



La première étape est achevée, il est nécessaire maintenant de configurer les différents éléments en développant l'arborescence.

### 2.3.2. CONFIGURATION DU PROJET HARMONY

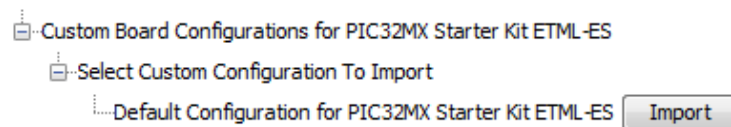
Il est important de commencer par la sélection du BSP, car cela impacte la section "Device & Project Configuration".

#### 2.3.2.1. BSP CONFIGURATION

Nous disposons d'une liste de BSP, dont celui du starter-kit ES :



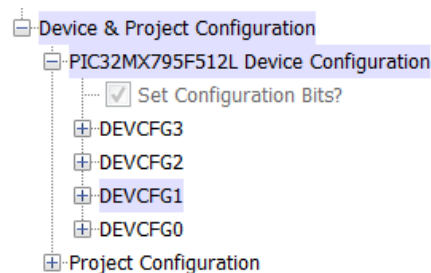
Au-dessous de la sélection du BSP, effectuer l'importation des réglages par défaut pour le kit :



#### 2.3.2.2. DEVICE CONFIGURATION

Dans le code, la configuration des fusibles est réalisée sous forme de directives `#pragma config`. Ce code est généré automatiquement par le MHC.

Sous Device & Project Configuration > PIC32MX795F512L Device Configuration, on trouve les 4 registres DEVCFG0 à DEVCFG3 qui correspondent aux configuration bits :



De par le choix du BSP du starter-kit ES, la partie "Device configuration" sera automatiquement configurée avec les valeurs par défaut pour le kit.



### 2.3.2.2.1. Section DEVCFG3

Cette section concerne les parties Ethernet, CAN et USB du PIC32MX.

Il est possible de conserver la proposition par défaut. Si on le souhaite, on peut modifier les 2 config USB que l'on met à OFF.

DEVCFG3

- User ID (USERID) 0xffff
- SRS Select (FSRSSEL) PRIORITY\_7
- Ethernet RMII\MII Enable (FMIIEN) OFF
- Ethernet I/O Pin Select (FETHIO) ON
- CAN I/O Pin Select (FCANIO) ON
- USB USID Selection (FUSBIDIO) ON
- USB VBUS ON Selection (FVBUSONIO) ON

### 2.3.2.2.2. Section DEVCFG2

Cette section concerne la configuration des PLL.

La configuration par défaut du BSP permet une configuration qui convient au kit PIC32MX avec un quartz à 8 MHz.

DEVCFG2

- PLL Input Divider (FPLLIDIV) DIV\_2
- PLL Multiplier (FPLLMUL) MUL\_20
- USB PLL Input Divider (UPLLIDIV) DIV\_2
- USB PLL Enable (UPLLEN) ON
- System PLL Output Clock Divider (FPLLODIV) DIV\_1

### 2.3.2.2.3. Section DEVCFG1

Cette section concerne la configuration des oscillateurs.

DEVCFG1

- Oscillator Selection Bits (FNOSC) PRIPLL
- Secondary Oscillator Enable (FSOSCEN) OFF
- Internal\External Switch Over (IESO) OFF
- Primary Oscillator Configuration (POSCMOD) XT
- CLKO Output Signal Active on the OSCO Pin (OSCIOFNC) OFF
- Peripheral Clock Divisor (FPBDIV) DIV\_1
- Clock Switching and Monitor Selection (FCKSM) CSECMD
- Watchdog Timer Postscaler (WDTPS) PS1048576
- Watchdog Timer Enable (FWDTEN) OFF

### 2.3.2.2.4. Section DEVCFG0

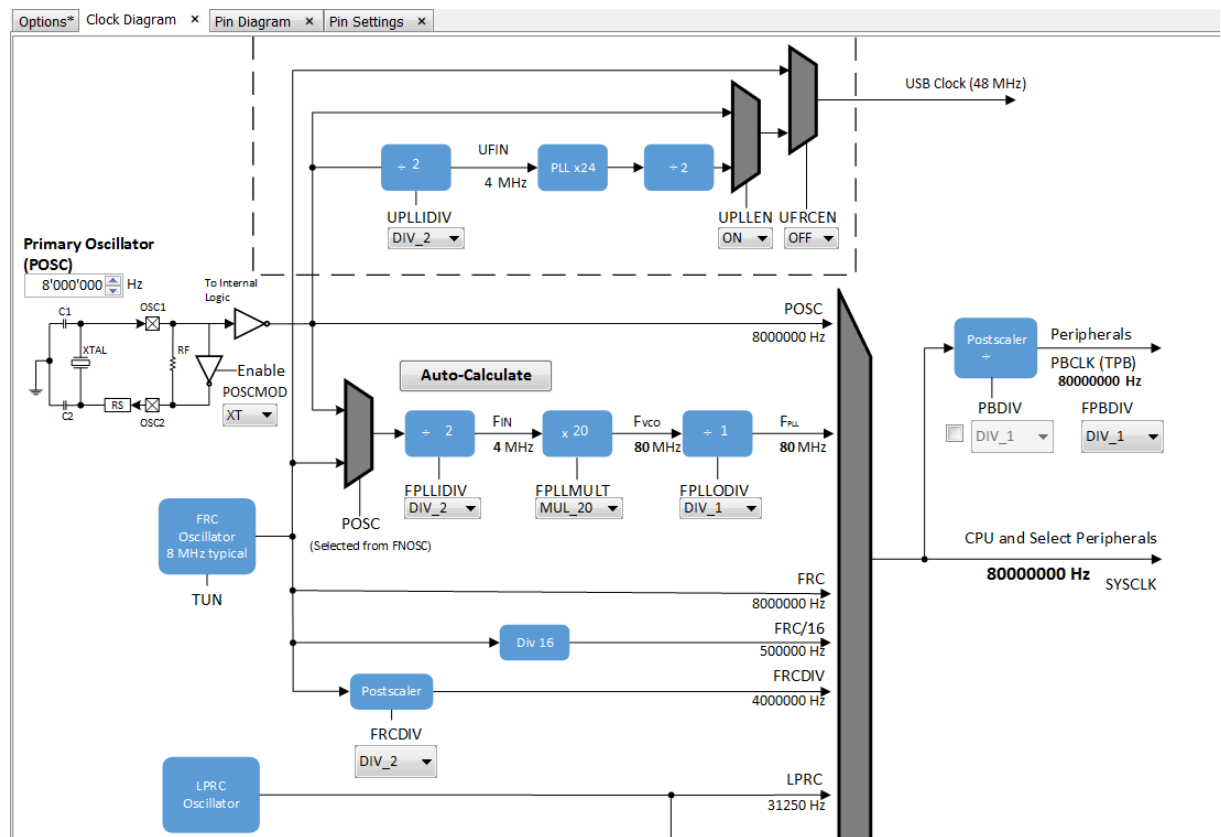
Cette section concerne le debugger et la mémoire programme.

DEVCFG0

- Background Debugger Enable (DEBUG) ON
- ICE\ICD Comm Channel Select (ICESEL) ICS\_PGx2
- Program Flash Write Protect (PWP) OFF
- Boot Flash Write Protect bit (BWP) OFF
- Code Protect (CP) OFF

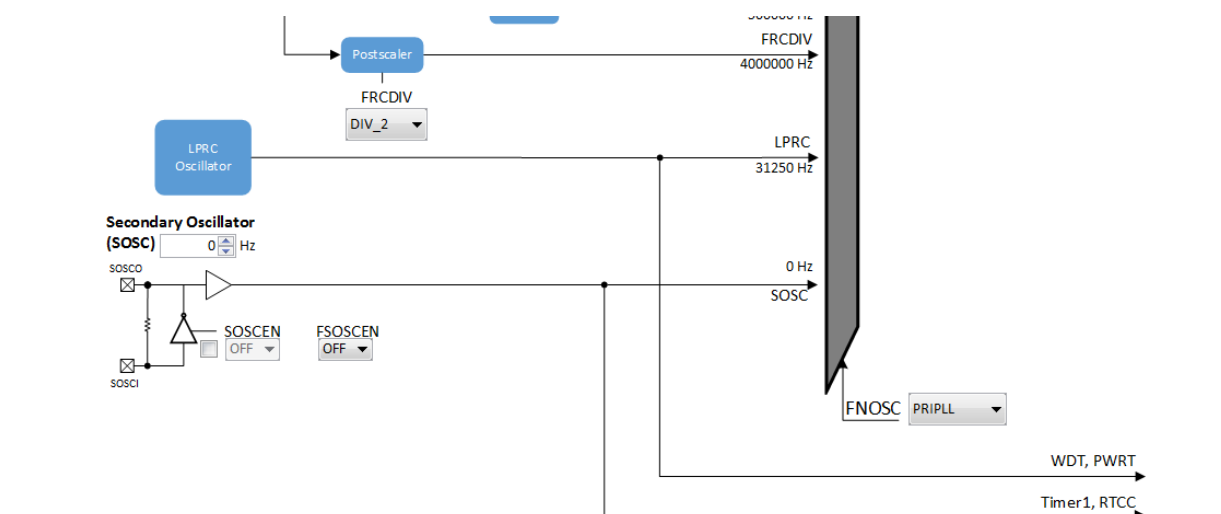
### 2.3.2.3. L'AIDE DU CLOCK DIAGRAM

Le Clock Diagram permet d'établir ou de vérifier aisément les paramètres PLL et oscillateur.



Si la configuration est correcte, il ne doit pas y avoir de valeur en rouge !

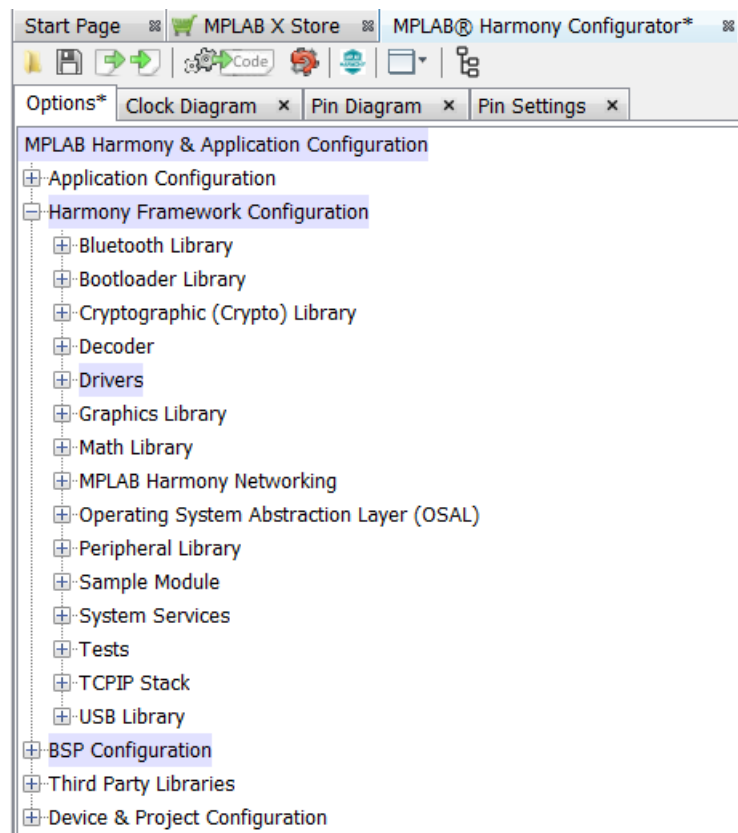
Avec un quartz à 8 MHz comme c'est le cas sur le starter-kit ES, il faut POSCMOD = XT. Avec le DIV\_2 puis le MUL\_20 on obtient au final 80 MHz pour SYSCLK, ce qui est le maximum pour le modèle de PIC32 du kit. Il faut encore FNOSC en PRIPLL.



Cela permet de vérifier que nous avons bien sélectionné l'oscillateur principal et que finalement PBCLK et SYSCLK sont les deux à 80 MHz.

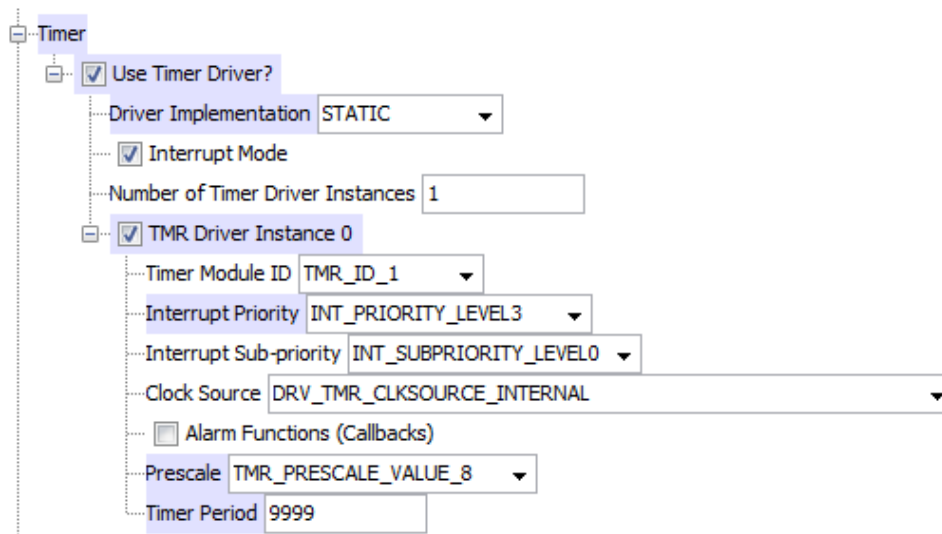
### 2.3.2.4. HARMONY FRAMEWORK CONFIGURATION

Cela va nous permettre de choisir les librairies, et surtout les drivers que l'on veut utiliser.



#### 2.3.2.4.1. Drivers, timer

Dans le cadre de notre exemple, au niveau Driver, nous sélectionnons un timer que nous configurons pour une période de 1 ms. Avec une horloge 80 MHz, cela s'obtient par exemple avec une division (prescaler) par 8, et un comptage sur 10'000 pas (d'où une valeur de 9'999 : un comptage de 0 à 9'999 donne bien 10'000 pas) :



#### 2.3.2.4.2. Peripheral Library

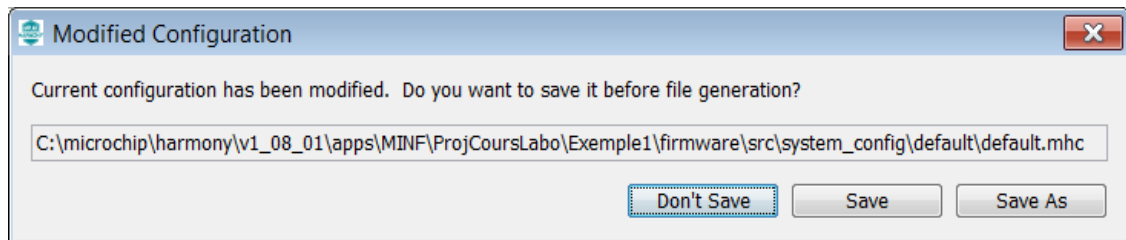
Elle est imposée par le choix du PIC.



#### 2.3.2.5. GENERATION ET SAUVEGARDE DE LA CONFIGURATION

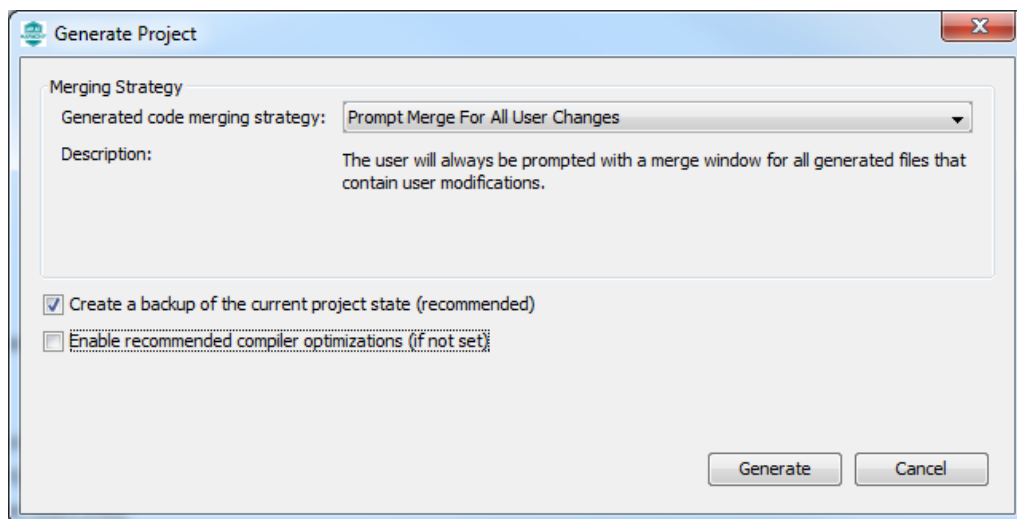
Génération avec le bouton Generate...

La demande de sauvegarde s'affiche, répondre Save.



Au menu suivant :

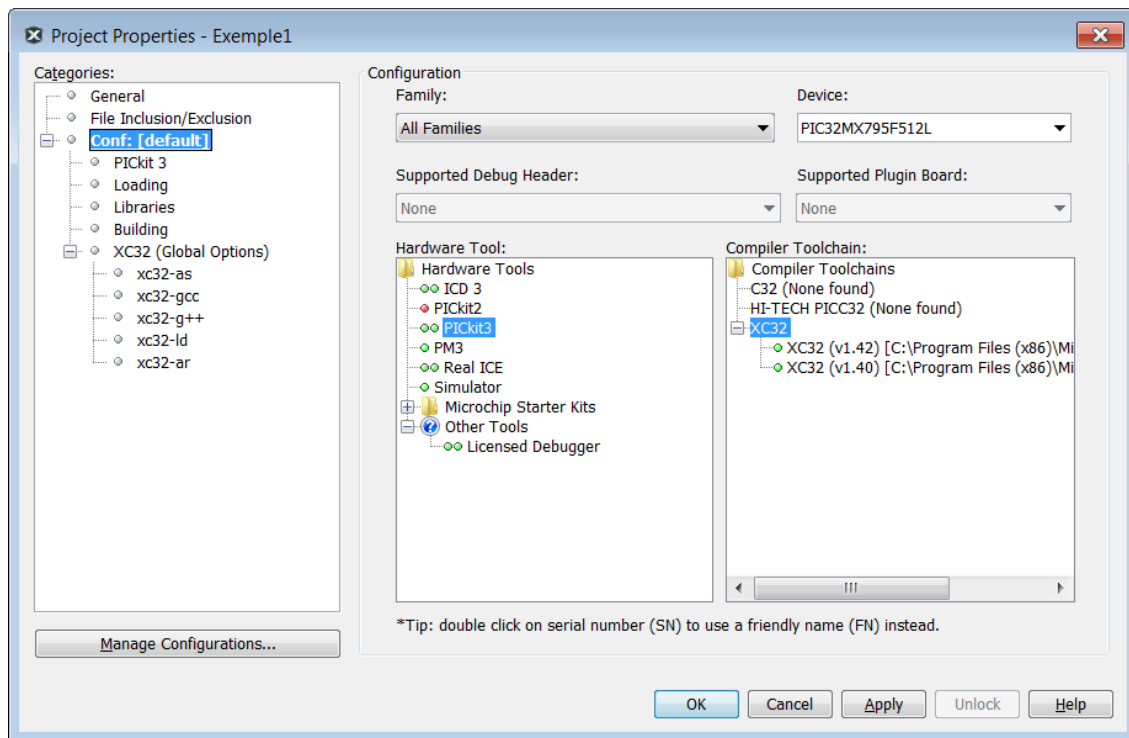
- Enlever la coche "Enable recommended compiler optimizations (if not set)", sans quoi l'optimisation de la compilation sera remise à 1 à chaque génération de code par le MHC.
- Laisser le reste par défaut



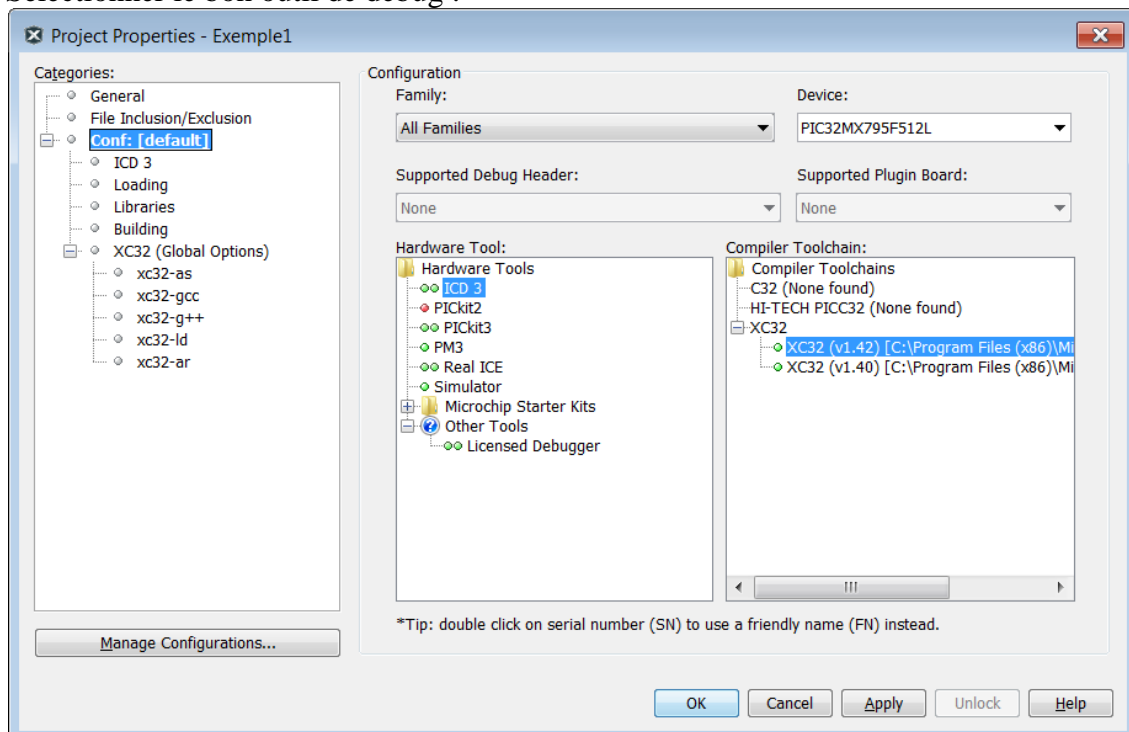
Avec ce choix, une demande de fusion apparaîtra pour chaque modification.

### 2.3.3. COMPLEMENTS DE CONFIGURATION

En observant les propriétés, on constate que certains éléments ne correspondent pas à la configuration voulue, en particulier pour le debug tool.



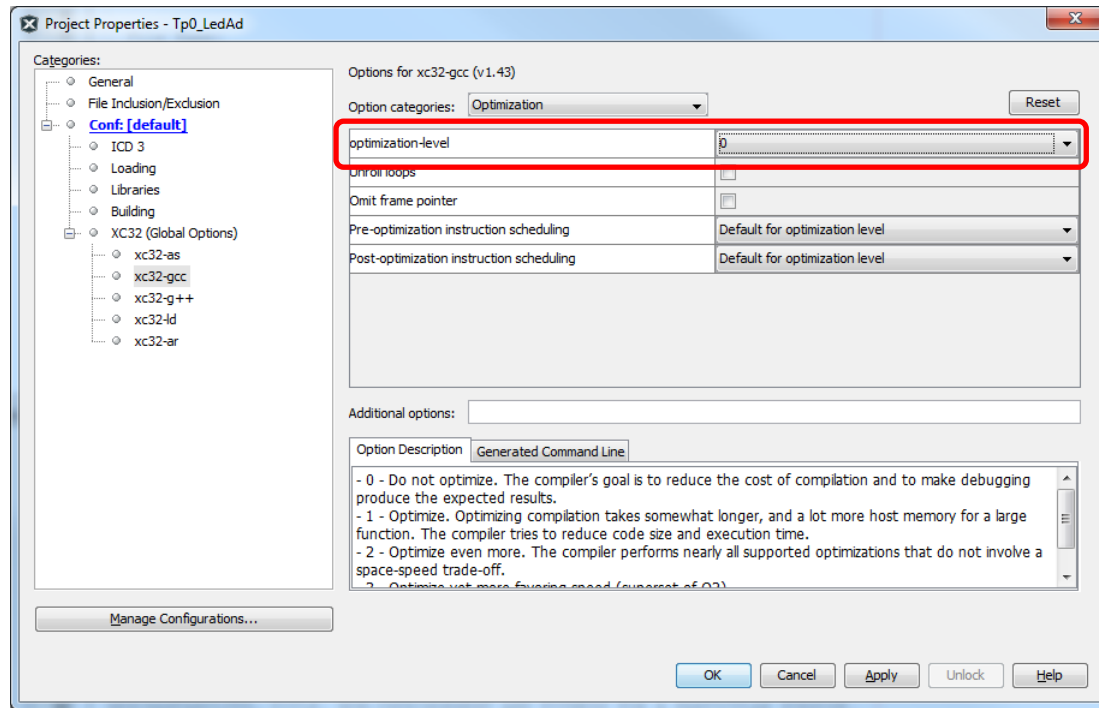
Sélectionner le bon outil de debug :



### 2.3.3.1. NIVEAU D'OPTIMISATION

Dans les propriétés, sous xc32-gcc > optimization, par défaut le niveau d'optimisation est 1.

Pour faciliter le debugging il faut **régler le niveau d'optimisation à "0 - Do not optimize"** :



Un niveau d'optimisation à zéro ne sera pas toujours possible avec des programmes volumineux.



## 2.4. ADAPTATION DU CANEVAS

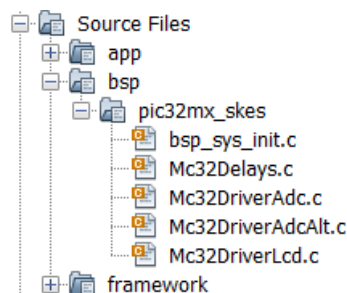
Voici quelques étapes nécessaires avant d'obtenir une application capable d'exécuter une action sur le kit PIC32MX de l'ES. Maintenant avec l'obtention du BSP `pic32mx_skes`, il s'agit seulement de vérifier si tout est correct dans l'arborescence du projet.

### 2.4.1. VERIFICATION DU BSP

Il s'agit de vérifier si on a bien la présence des fichiers du BSP dans les fichiers source et header.

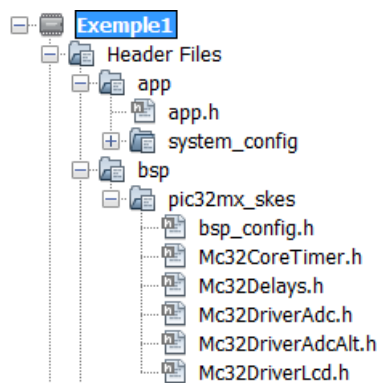
#### 2.4.1.1. LOCALISATION DU BSP DANS LES SOURCE FILES

Il faut vérifier la présence du répertoire BSP et du sous répertoire `pic32mx_skes`, qui doit contenir les fichiers comme ci-dessous :



#### 2.4.1.2. LOCALISATION DU BSP DANS HEADER FILES

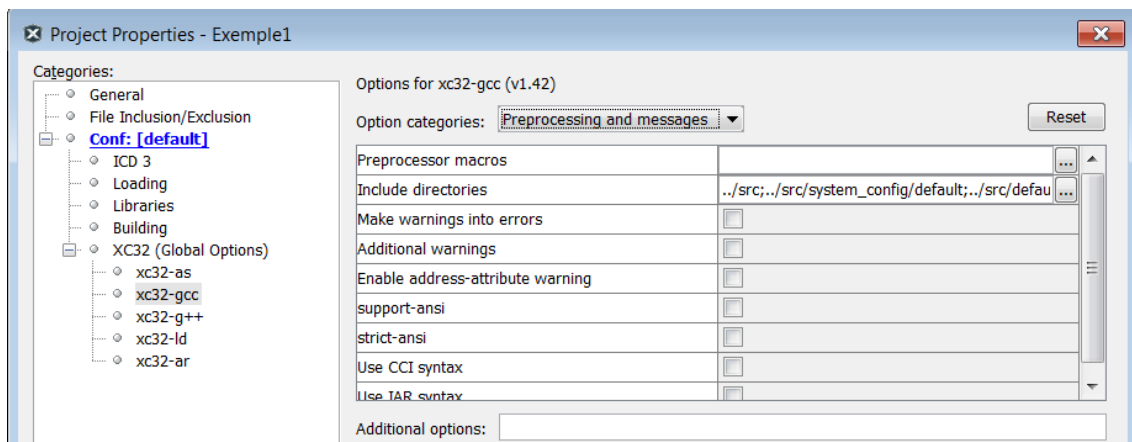
Il faut vérifier la présence du répertoire BSP et du sous répertoire `pic32mx_skes`, qui doit contenir les fichiers comme ci-dessous :



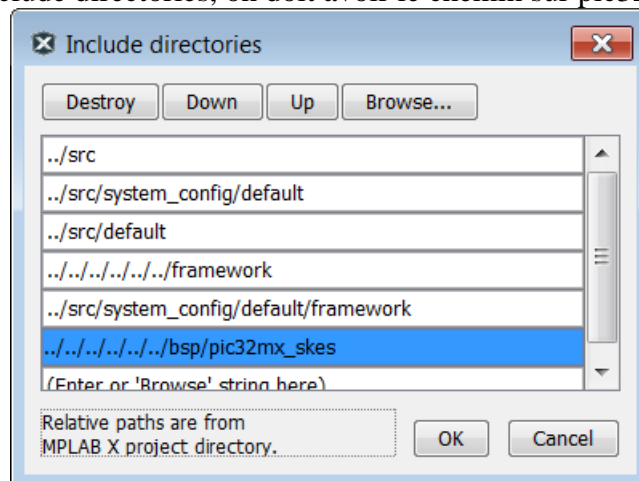


### 2.4.1.3. VERIFICATION DU CHEMIN INCLUDE

Pour vérifier si on dispose du bon chemin d'include, on utilise les propriétés du projet, sous xc32-gcc :

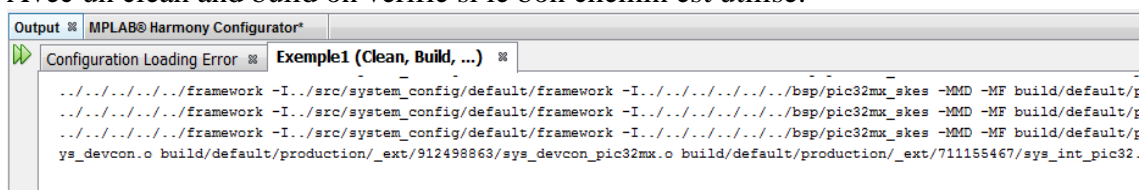


Puis en ouvrant Include directories, on doit avoir le chemin sur pic32mx\_skes :



### 2.4.1.4. CONTROL AVEC BUILD

Avec un clean and build on vérifie si le bon chemin est utilisé.



## 2.4.2. CONTENU DU BSP

Le détail du contenu du BSP spécifique seront traités dans le chapitre 4.

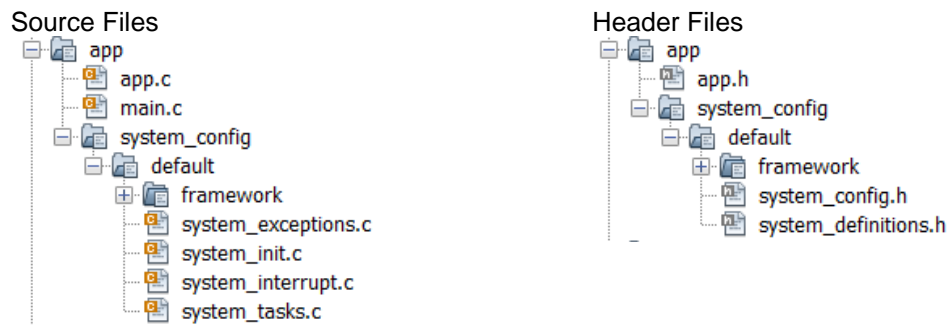
## 2.5. ADAPTATION DE L'APPLICATION

Dans le canevas généré par le MHC, on trouve sous "Sources Files" 3 logical folder, nommées app, bsp et framework. On retrouve la même organisation dans les "Header Files" :



### 2.5.1. CONTENU DES LOGICAL FOLDER APP

Voici le contenu des logical folder app dans les Source et les Header Files



Les adaptations de l'application se font principalement dans app.c et app.h. Au niveau du sous répertoire system\_config. Selon les situations, il sera nécessaire de modifier system\_init.c et system\_interrupt.c.

#### 2.5.1.1. CONTENU DE MAIN.C

Le programme principal sera en principe toujours le même.

```

int main ( void )
{
    /* Initialize all MPLAB Harmony modules, including application(s). */
    SYS_Initialize ( NULL );

    while ( true )
    {
        /* Maintain state machines of all polled MPLAB Harmony modules. */
        SYS_Tasks ( );
    }

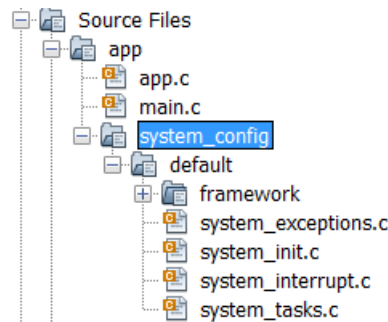
    /* Execution should not come here during normal operation */

    return ( EXIT_FAILURE );
}
  
```

La fonction SYS\_Tasks(), qui se trouve dans le fichier system\_tasks.c, appelle la fonction APP\_Tasks() qui elle se trouve dans le fichier app.c.

### 2.5.2. CONTENU DU LOGICAL FOLDER SYSTEM\_CONFIG

Le "logical folder" system\_config contient 4 fichiers nommés system\_exceptions.c, syst\_init.c, syst\_interrupts.c et system\_tasks.c.



#### 2.5.2.1. CONTENU DU FICHIER SYSTEM\_INIT.C

Le fichier system\_init.c, dont le contenu varie en fonction de la configuration réalisée, contient principalement :

- La section "*Configuration bits*" (liste des `#pragma config`)
- La fonction `SYS_Initialize`. Cette fonction appelle les sous-fonctions d'initialisation comme `BSP_Initialize`, ainsi que les fonctions d'initialisation des drivers (dans notre exemple l'initialisation du timer1).

#### 2.5.2.2. CONTENU DU FICHIER SYSTEM\_INTERRUPT.C

Le fichier system\_interrupt.c, dont le contenu varie en fonction de la configuration réalisée, contient la ou les ISR (Interrupt Service Routine). Il n'y a que le squelette de la réponse avec l'action de clear du flag d'interruption.

#### 2.5.2.3. CONTENU DU FICHIER SYSTEM\_TASKS.C

Le fichier system\_tasks.c, contient essentiellement la fonction `SYS_Tasks ()`, qui elle-même appelle la fonction `APP_Tasks()` qui elle se trouve dans le fichier app.c.

```
void SYS_Tasks ( void )
{
    /* Maintain system services */
    SYS_DEVCON_Tasks(sysObj.sysDevcon);

    /* Maintain Device Drivers */

    /* Maintain Middleware & Other Libraries */

    /* Maintain the application's state machine. */
    APP_Tasks();
}
```

#### 2.5.2.4. CONTENU DU FICHIER SYSTEM\_EXCEPTIONS.C

Le fichier system\_exceptions.c, contient les textes des messages d'exception ainsi qu'une fonction `_general_exception_handler`.

### 2.5.3. SITUATION AU NIVEAU SYSTEM\_INIT.C

Comme le nouveau BSP fournit les définitions grâce au fichier bsp.xml, il est possible d'exploiter la fonction SYS\_PORTS\_Initialize. Il faut vérifier si le système appelle bien la fonction SYS\_DEVCON\_JTAGDisable.

```
void SYS_Initialize ( void* data )
{
    /* Core Processor Initialization */
    SYS_CLK_Initialize( NULL );
    sysObj.sysDevcon = SYS_DEVCON_Initialize(SYS_DEVCON_INDEX_0, (SYS_MODULE_INIT*)&sysDevconInit);
    SYS_DEVCON_PerformanceConfig(SYS_CLK_SystemFrequencyGet());
    SYS_DEVCON_JTAGDisable();
    SYS_PORTS_Initialize();
    /* Board Support Package Initialization */
    BSP_Initialize();

    /* Initialize Drivers */
    /*Initialize TMR0 */
    DRV_TMR0_Initialize();

    /* Initialize System Services */

    /*** Interrupt Service Initialization Code ***/
    SYS_INT_Initialize();

    /* Initialize Middleware */

    /* Enable Global Interrupts */
    SYS_INT_Enable();

    /* Initialize the Application */
    APP_Initialize();
}
```

#### 2.5.3.1. CONTENU DE BSP\_INITIALIZE

La fonction BSP\_Initialize configure en analogique ou digital les pins du port qui contient les entrées analogiques.

```
void BSP_Initialize(void )
{
    // Pour ne pas entrer en conflit avec le JTAG
    SYS_DEVCON_JTAGDisable(); // par sécurité

    // Config AN0 et AN1 en Analogique et les autres
    // en digital
    // Nécessaire de le faire à cause des éléments
    // non configurés
    PLIB_PORTS_AnPinsModeSelect(PORTS_ID_0, 0x0003,
                                PORTS_PIN_MODE_ANALOG);
    PLIB_PORTS_AnPinsModeSelect(PORTS_ID_0, ~0x0003,
                                PORTS_PIN_MODE_DIGITAL);
}
```

Le reste de la fonction contient d'anciennes actions en commentaire.

### 2.5.4. GESTION DE LA MACHINE D'ÉTAT DE L'APPLICATION

Le fichier généré app.c contient l'embryon de la fonction APP\_Tasks avec deux états (APP\_STATE\_INIT & APP\_STATE\_SERVICE\_TASKS). Nous allons introduire un état supplémentaire : **APP\_STATE\_WAIT**.

Ensuite, nous ajouterons un mécanisme basé sur l'interruption d'un timer pour faire passer cycliquement l'état de l'application de WAIT à SERVICE\_TASKS.

Voici les étapes des modifications :

#### 2.5.4.1. COMPLEMENT DE APP\_STATE DANS APP.H

Nous complétons de la manière suivante le type énuméré APP\_STATE qui est dans app.h.

```
typedef enum
{
    /* Application's state machine's initial state. */
    APP_STATE_INIT=0,
    APP_STATE_WAIT,          // CHR ajout de WAIT
    APP_STATE_SERVICE_TASKS,

    /* TODO: Define states used by the application
       state machine. */
} APP_STATES;
```

Nous complétons la définition de la structure APP\_DATA en ajoutant un compteur.

```
typedef struct
{
    /* The application's current state */
    APP_STATES state;

    /* TODO: Define any additional data used by the application. */
    int32_t Count;

} APP_DATA;
```

#### 2.5.4.2. AJOUT D'UNE FONCTION DE MISE A JOUR DE APP\_UPDATESTATE

Nous ajoutons le prototype de la fonction dans le fichier app.h.

```
void APP_UpdateState ( APP_STATES NewState ) ;
```

#### 2.5.4.3. MISE A JOUR DE LA FONCTION APP\_TASKS

Nous modifions le corps de la fonction APP\_Tasks dans le fichier app.c de la manière suivante (les modifications sont en gras).

```
void APP_Tasks ( void )
{
    /* Check the application's current state. */
    switch ( appData.state )
```

```
{
    /* Application's initial state. */
    case APP_STATE_INIT:
        // bool appInitialized = true;
        BSP_LEDOn(BSP_LED_0);
        BSP_LEDOn(BSP_LED_1);
        BSP_LEDOn(BSP_LED_2);
        BSP_LEDOn(BSP_LED_3);
        BSP_LEDOn(BSP_LED_4);
        BSP_LEDOn(BSP_LED_5);
        BSP_LEDOn(BSP_LED_6);
        BSP_LEDOn(BSP_LED_7);

        // Init du LCD
        lcd_init();
        lcd_bl_on();
        // Start du Timer1
        DRV_TMR0_Start();
        printf_lcd("App Exemple1");
        lcd_gotoxy(1,2);
        printf_lcd("C. Huber 21.09.2016");

        appData.Count = 0;
        appData.state = APP_STATE_WAIT;

        /* NON utilisé
           if (appInitialized)
           {

               appData.state = APP_STATE_SERVICE_TASKS;
           }
        */
        break;

    case APP_STATE_WAIT :
        // nothing to do
        break;

    case APP_STATE_SERVICE_TASKS:
        BSP_LEDToggle(BSP_LED_0);
        BSP_LEDToggle(BSP_LED_3);
        appData.Count++;
        lcd_gotoxy(1,3);
        printf_lcd("Count = %5d", appData.Count);
        if (appData.Count > 999999) appData.Count = 0;
        appData.state = APP_STATE_WAIT;
        break;
```

```
        /* The default state should never be executed. */
        default:
        {
            /* TODO: Handle error in application's state
machine. */
            break;
        }
    }
}
```

On peut observer la modification de la machine d'état ainsi que l'utilisation des fonctions introduites dans le BSP.

☞ La fonction d'initialisation des drivers de timer laisse le timer stoppé, il faut donc le starter en utilisant **DRV\_TMR0\_Start()**.

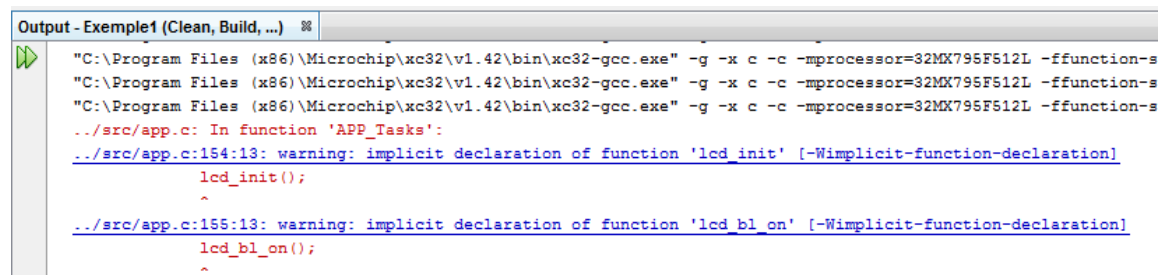
#### 2.5.4.4. AJOUT DE LA FONCTION APP\_UPDATESTATE DANS APP.C

Nous introduisons l'implémentation de la fonction APP\_UpdateState dans le fichier app.c.

```
void APP_UpdateState ( APP_STATES NewState )
{
    appData.state = NewState;
}
```

#### 2.5.4.5. AJOUT INCLUDE DANS APP.C

Si on a coché "Additional warnings" dans les options du compilateur, comme recommandé au §2.3.3.2 "Avertissements à la compilation", on obtient les warnings suivants :



```
Output - Exemple1 (Clean, Build, ...) %
"C:\Program Files (x86)\Microchip\xc32\v1.42\bin\xc32-gcc.exe" -g -x c -c -mprocessor=32MX795F512L -ffunction-s
"C:\Program Files (x86)\Microchip\xc32\v1.42\bin\xc32-gcc.exe" -g -x c -c -mprocessor=32MX795F512L -ffunction-s
"C:\Program Files (x86)\Microchip\xc32\v1.42\bin\xc32-gcc.exe" -g -x c -c -mprocessor=32MX795F512L -ffunction-s
../src/app.c: In function 'APP_Tasks':
../src/app.c:154:13: warning: implicit declaration of function 'lcd_init' [-Wimplicit-function-declaration]
    lcd_init();
    ^
../src/app.c:155:13: warning: implicit declaration of function 'lcd_bl_on' [-Wimplicit-function-declaration]
    lcd_bl_on();
    ^
```

Que l'on corrige par :

```
#include "Mc32DriverLcd.h"
```

#### 2.5.4.6. UTILISATION DE LA FONCTION APP\_UPDATESTATE DANS SYSTEM\_INTERRUPT.C

Nous modifions l'ISR du timer1 de la manière suivante dans le fichier system\_interrupt.c (les ajouts sont en gras). La période du timer1 est prévue pour 1 ms.

```
void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
    IntHandlerDrvTmrInstance0(void)
{
    static int16_t waitCount = 0;

    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
    waitCount++;
    // Attentes de 2 secondes, puis cycle de 100 ms
    if (waitCount >= 2000) {
        APP_UpdateState(APP_STATE_SERVICE_TASKS);
        waitCount = 1900;
    }
    // Pour test de la période du Timer1
    BSP_LEDToggle(BSP_LED_6);
}
```

L'interruption cyclique du timer 1 se produit à chaque ms, et après une attente de 2 secondes l'application passe en **APP\_STATE\_SERVICE\_TASKS** chaque 100 ms.

#### 2.5.5. APPLICATION, TEST & CONCLUSION

Après chargement avec l'ICD (voir chapitre 3) on obtient bien le fonctionnement prévu. L'afficheur fonctionne et les périodes de clignotement des leds sont conformes.

☺ Par le fait que dans les trois fichiers générés par Harmony (syst\_init.c, syst\_interrupts.c et system\_tasks.c) on trouve un #include de app.h, cela permet d'étendre facilement le principe utilisé d'ajouter des fonctions qui permettent d'agir sur l'application depuis certains fichiers systèmes.



## 2.6. ADAPTATION D'UNE APPLICATION EXEMPLE

Lorsqu'au lieu de créer un projet avec le MHC, on adapte un projet exemple, il est nécessaire de remplacer le BSP utilisé par l'exemple.

Les étapes sont les suivantes :

- Ouvrir la configuration
- Supprimer l'usage des BSP et générer le code
- Réactiver les BSP et choisir celui du kit ES puis générer le code.  
Attention à l'importation des réglages par défaut du nouveau BSP : cela risque d'écraser certains réglages.
- Vérifier la "Device configuration" et retoucher si nécessaire.

Les explications sont réalisées sur la base de l'exemple `adc_pot` que l'on trouve sous `<Répertoire Harmony>\v<n>\apps\examples\peripheral\adc`.

### 2.6.1. EMBLACEMENT DE LA COPIE DE L'EXEMPLE

👉 Il est important de ne jamais modifier les exemples fournis, mais de les copier.

Pour éviter des problèmes de chemin et d'include non résolus il n'y a que 2 solutions fiables :

- Copier au même endroit et renommer le répertoire du projet exemple (la plus sûre).
- Copier dans un sous répertoire de **apps** en respectant le même nombre de niveaux de sous répertoires.

☛ Les exemples sont déplaçables pour autant que l'on ait le même nombre de répertoires par rapport à **apps**, ceci à cause des chemins relatifs.

### 2.6.2. COPIE DE L'EXEMPLE ADC\_POT

Pour valider la solution du déplacement, on copie l'exemple sous

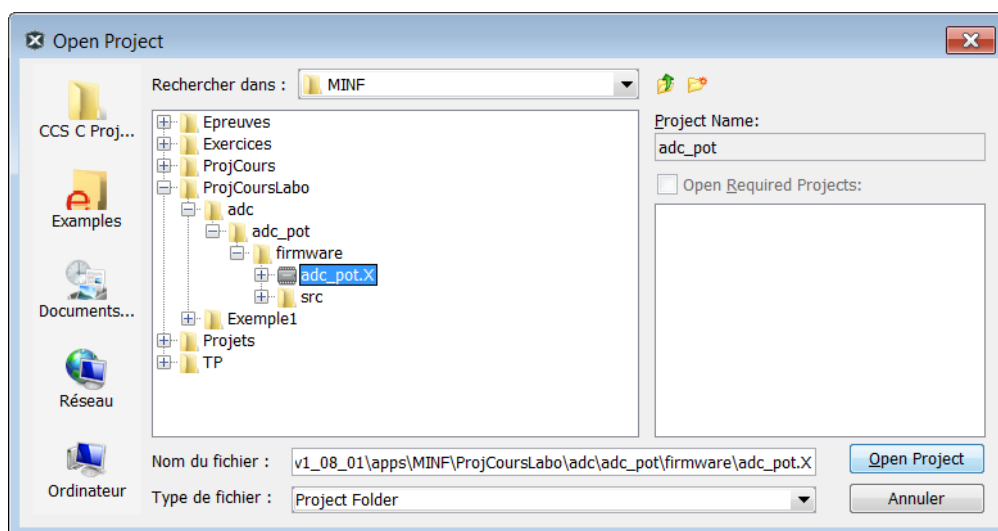
`<Répertoire Harmony>\v<n>\apps\MINF\ProjCoursLabo\adc`,

ce qui correspond au même nombre de sous-répertoire que l'original qui est sous :

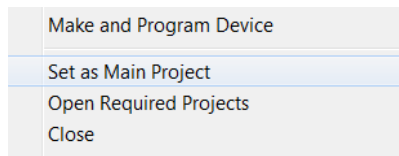
`<Répertoire Harmony>\v<n>\apps\examples\peripheral\adc`


### 2.6.3. OUVERTURE DU PROJET

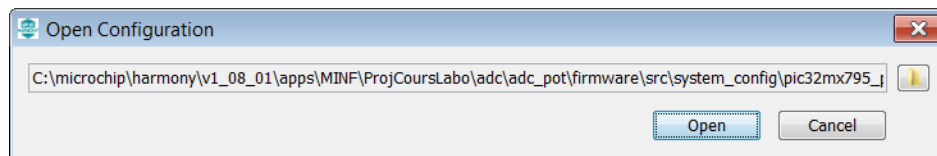
Ouverture du projet avec File, Open Project



Après ouverture ne pas oublier l'action "Set as main Project"

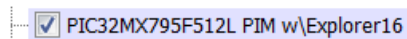


Cela a pour effet de lancer le  MPLAB® Harmony Configurator et de demander l'ouverture de la configuration.

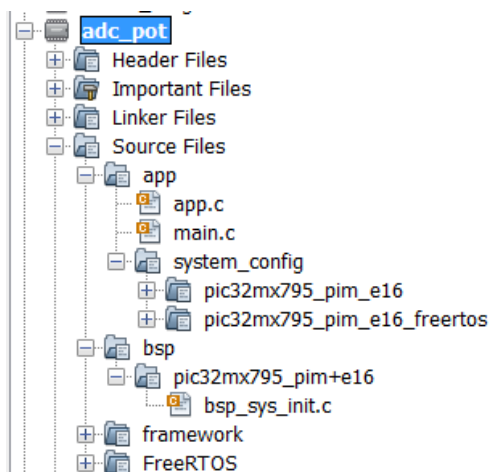


## 2.6.4. SITUATION DE L'EXEMPLE ADC\_POT

Au niveau du BSP celui qui est sélectionné est :



Au niveau du projet, on observe :

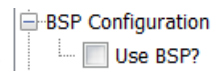


On dispose de 2 configurations, dont une avec FreeRTOS.

## 2.6.5. REMPLACEMENT DU BSP

### 2.6.5.1. SUPPRESSION DU BSP

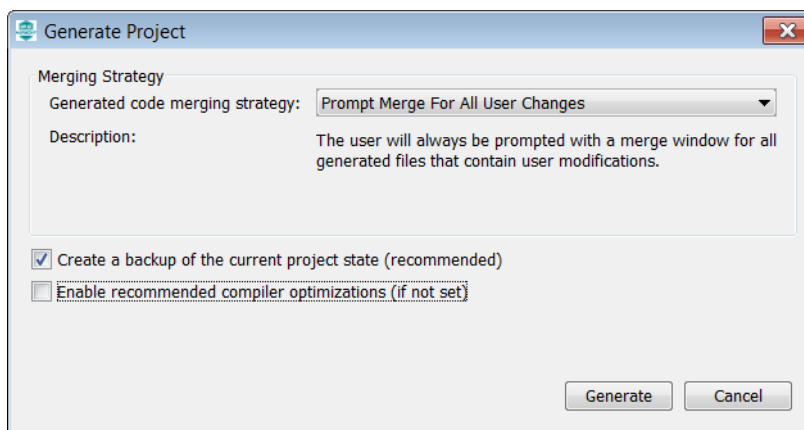
Pour changer de BSP, il faut décocher Use BSP



et générer le code



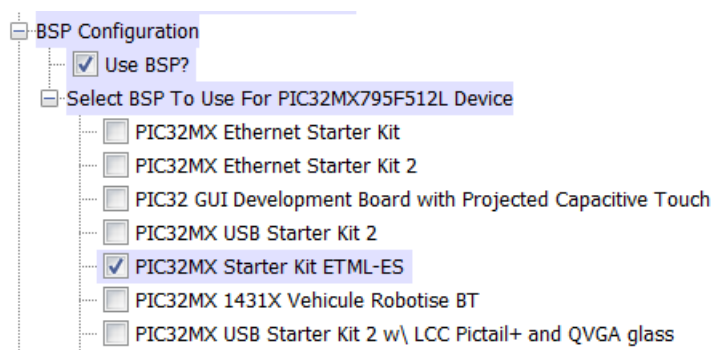
Au niveau Generate Project, on choisit Prompt Merge For All User Changes (par défaut) :



Lors du Merge, il n'est pas nécessaire d'écraser le contenu des fichiers (utilisez close).

### 2.6.5.2. UTILISATION DU BSP DU KIT ES

Il suffit de cocher Use BSP? et de sélectionner "PIC32MX starter-kit ETML-ES" :

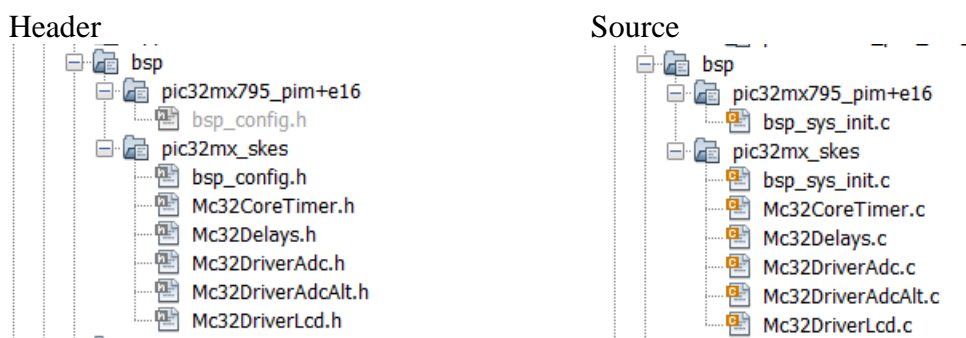


Ensuite on génère à nouveau le code.

## 2.6.6. VERIFICATION DU REMPLACEMENT DU BSP

### 2.6.6.1. SITUATION ARBORESCENCE BSP

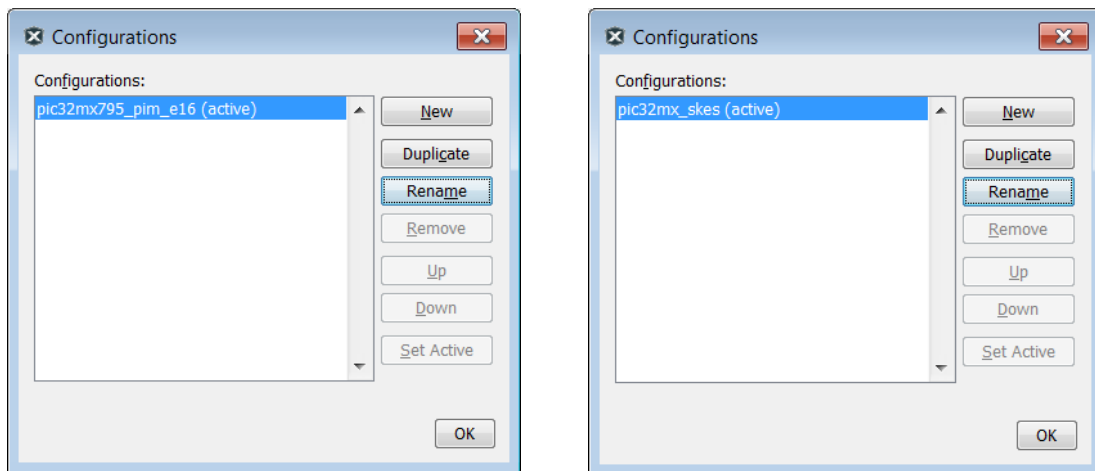
On vérifie que l'on a bien notre BSP et les fichiers associés.



On constate la présence du BSP du kit mais aussi des reliquats de l'ancien BSP.

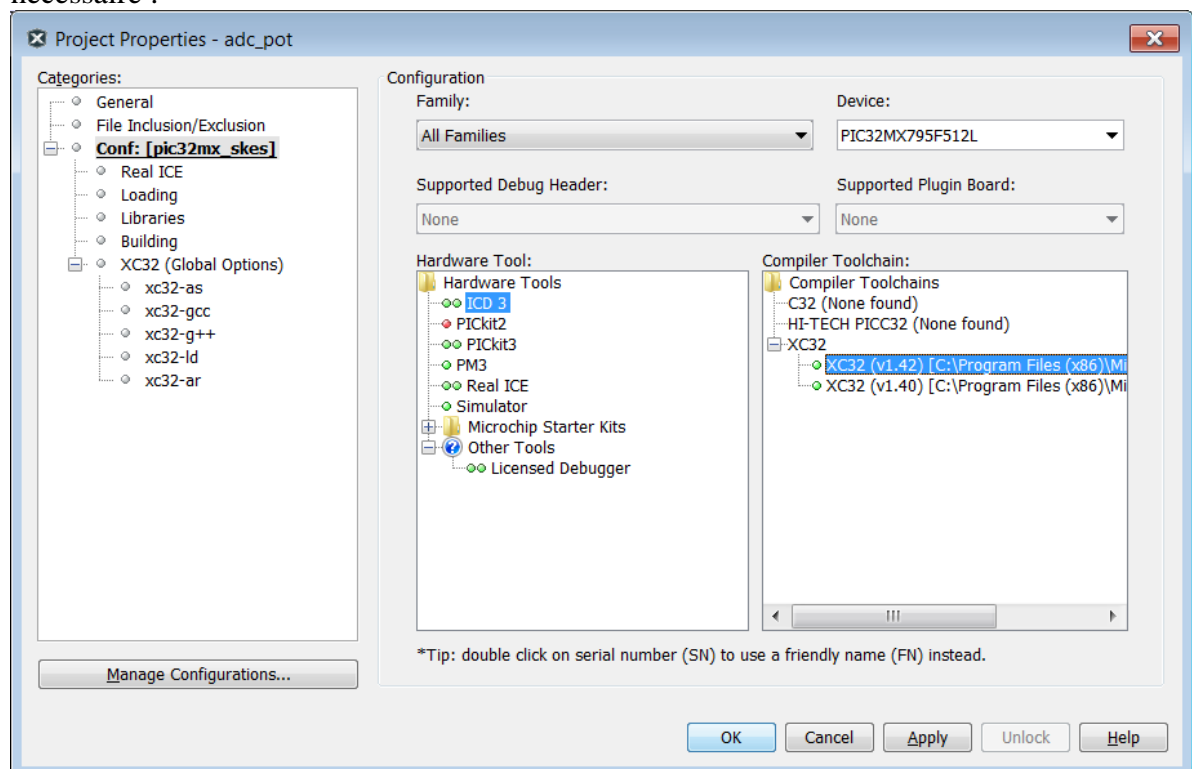
## 2.6.7. MISE EN ORDRE DE LA CONFIGURATION

Il est possible en utilisant  de supprimer une des configurations et de renommer l'autre en utilisant Rename.



### 2.6.7.1. AJUSTEMENT DE LA CONFIGURATION

On en profite pour indiquer l'utilisation de l'ICD et de la version du compilateur si nécessaire :



Dans les options du compilateur xc32-gcc, on réduit encore le niveau d'optimisation à 0 et on enlève la coche "Make warnings into errors".

### 2.6.7.2. NETTOYAGE DE L'ARBORESCENCE DU PROJET

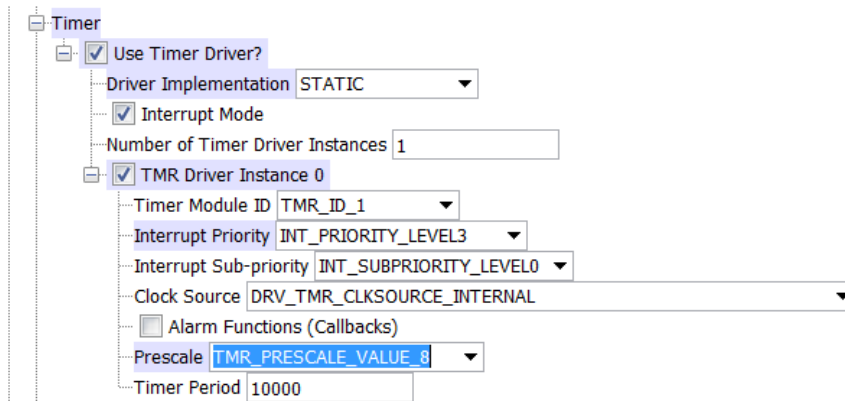
En utilisant "Remove from project", on supprime le répertoire de l'autre BSP et de la configuration avec FreeRTOS.

## 2.6.8. MODIFICATION DE L'HARMONY FRAMEWORK CONFIGURATION

Nous allons ajouter un driver de timer.

### 2.6.8.1. AJOUT D'UN TIMER

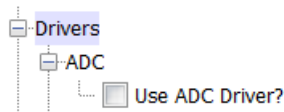
Nous introduisons un driver de timer dans la configuration :



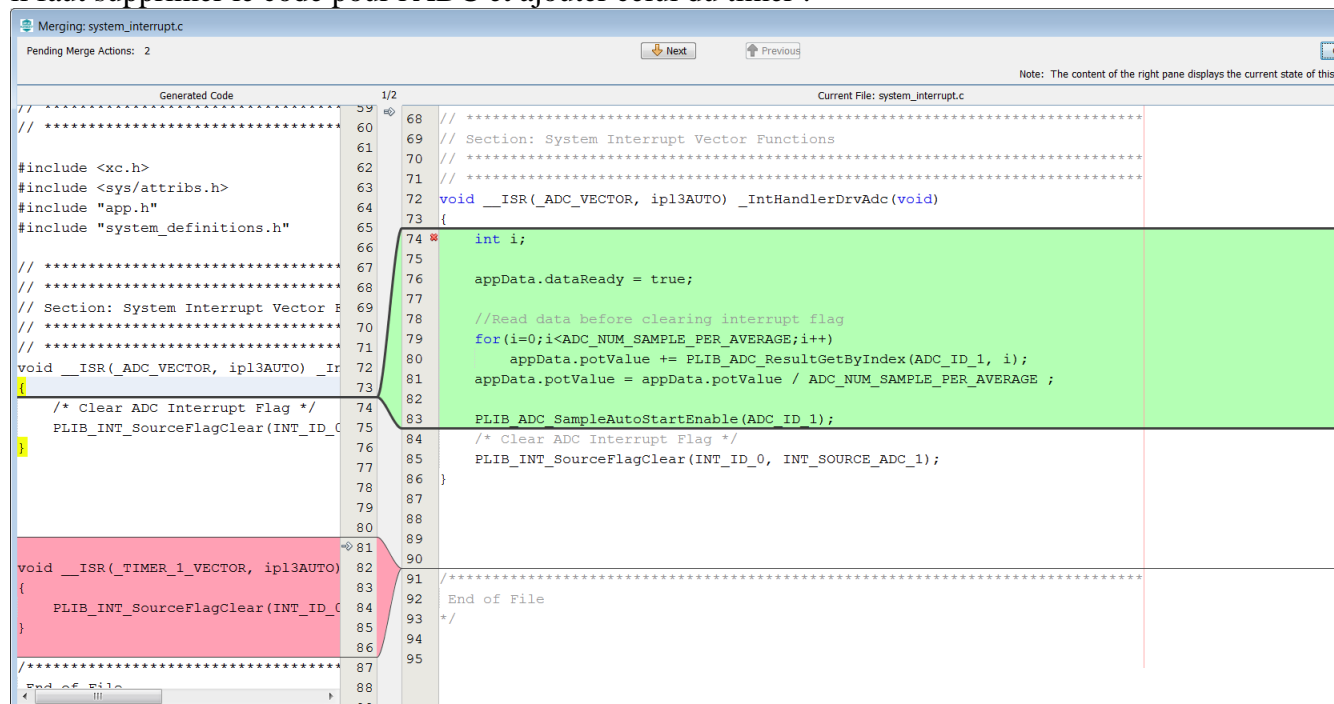
Cela nous fournira une interruption cyclique à 1 ms.

### 2.6.8.2. SUPPRESSION DU DRIVER ADC


Nous supprimons le driver ADC, car nous allons le remplacer par le mécanisme fourni par notre BSP :



Il est nécessaire de régénérer le code. Au niveau du merge du fichier system\_interrupt.c, il faut supprimer le code pour l'ADC et ajouter celui du timer :



### 2.6.8.3. TEST DE COMPILE

Avec un  (Clean and build main project), tout va bien, sauf un appel à DRV\_ADC\_Open :

```
nbproject/Makefile-impl.mk:39: recipe for target '.build-impl' failed
build/pic32mx_skes/production/_ext/1360937237/app.o: In function 'APP_Tasks':
c:/microchip/harmony/v1_08_01/apps/minf/projcourse/adc/adc_pot/firmware/src/app.c:168: undefined reference to 'DRV_ADC_Open'
collect2.exe: error: ld returned 255 exit status
make[2]: *** [dist/pic32mx_skes/production/adc_pot.X.production.hex] Error 255
```

Il faut le mettre en commentaire dans app.c

```
/* Application's initial state.
case APP_STATE_INIT:
    /* Enable ADC */
    // DRV_ADC_Open();
    appData.state = APP_STATE_SF

break;
```

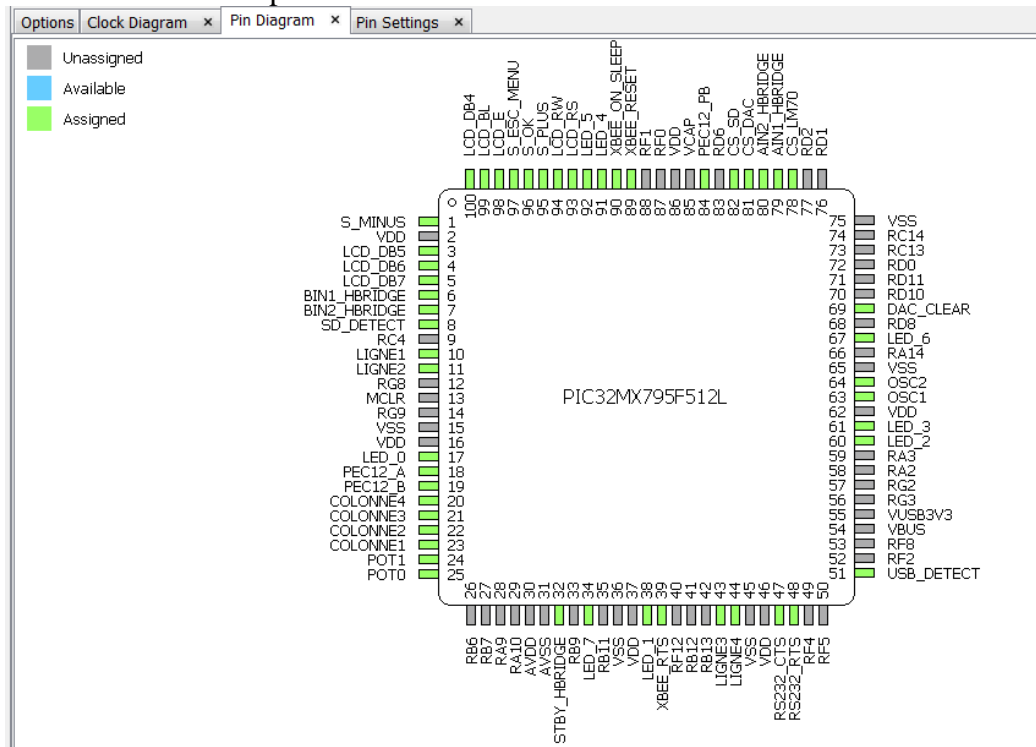
Et on obtient une compilation sans erreur ni warnings.

On observe que le BSP pic32mx\_skes est bien utilisé.

```
_e16/framework -I../../../../../../../../bsp/pic32mx_skes -Wall -MMD -MF build/pic32mx_skes/prodi
_e16/framework -I../../../../../../../../bsp/pic32mx_skes -Wall -MMD -MF build/pic32mx_skes/prodi
_e16/framework -I../../../../../../../../bsp/pic32mx_skes -Wall -MMD -MF build/pic32mx_skes/prodi
_e16/framework -I../../../../../../../../bsp/pic32mx_skes -Wall -MMD -MF build/pic32mx_skes/prodi
```

### 2.6.8.4. VERIFICATION PIN CONFIGURATION

On doit obtenir la description des entrées-sorties du kit ES.



On contrôle encore au niveau "Pin Settings" que les sorties sont bien marquées out.

89	RG1	5V	XBEE_RESET	Out	High	<input type="checkbox"/>	Digital
90	RG0	5V	XBEE_ON_SLEEP	Out	Low	<input type="checkbox"/>	Digital
91	RA6	5V	LED_4	Out	High	<input type="checkbox"/>	Digital
92	RA7	5V	LED_5	Out	High	<input type="checkbox"/>	Digital
93	RE0	5V	LCD_RS	Out	High	<input type="checkbox"/>	Digital
94	RE1	5V	LCD_RW	Out	High	<input type="checkbox"/>	Digital
95	RG14	5V	S_PLUS	In	Low	<input type="checkbox"/>	Digital

### 2.6.8.5. CONTROLE DEVICE CONFIGURATION

Avec le clock diagram, on vérifie qu'on utilise bien POSC et les bons diviseurs pour obtenir 80 MHz.

Au niveau DEVCFG1 on vérifie que l'on ait bien :

Watchdog Timer Enable (FWDTEN) OFF

An niveau DEVCFG0 on impose :

DEVCFG0  
Background Debugger Enable (DEBUG) ON

☹ Normalement, avec le BSP du kit, il devrait être sur ON automatiquement.

👉 Ne pas oublier de générer à nouveau.

### 2.6.8.6. CONTROLE SITUATION SYSTEM\_INIT

On vérifie si la fonction SYS\_Initialize utilise JTAGDisable.

```
void SYS_Initialize ( void* data )
{
    /* Core Processor Initialization */
    SYS_CLK_Initialize( NULL );
    sysObj.sysDevcon = SYS_DEVCON_Initialize(SYS_DEVCON_INDEX_0, (SYS
    SYS_DEVCON_PerformanceConfig(SYS_CLK_SystemFrequencyGet()));
    SYS_DEVCON_JTAGDisable();
    SYS_PORTS_Initialize();
    /* Board Support Package Initialization */
    BSP_Initialize();

    /* Initialize Drivers */
    /*Initialize TMR0 */
    DRV_TMR0_Initialize();
}
```

### 2.6.8.7. CONTROLES DEFINITIONS DANS SYSTEM\_CONFIG.H

On vérifie si les définitions des TRIS et des LAT ont bien des valeurs plausibles.

```

/** Ports System Service Configuration */
#define SYS_PORT_AD1PCFG    ~0x3ac3
#define SYS_PORT_CNPUE      0x0
#define SYS_PORT_CNEN       0x0

#define SYS_PORT_A_TRIS     0x460c
#define SYS_PORT_A_LAT      0x80c0
#define SYS_PORT_A_ODC      0x0

#define SYS_PORT_B_TRIS     0xfaff
#define SYS_PORT_B_LAT      0x400
#define SYS_PORT_B_ODC      0x0

#define SYS_PORT_C_TRIS     0xf018
#define SYS_PORT_C_LAT      0x0
#define SYS_PORT_C_ODC      0x0

```

Remarque : on atteint ce fichier en remontant à partir de la fonction SYS\_PORTS\_Initialize().

### 2.6.9. MODIFICATION DE L'APPLICATION

Nous allons modifier l'application afin d'afficher un message et les valeurs lues sur les deux potentiomètres.

Pour cela, il faut mettre de côté la gestion de l'ADC prévue par Microchip dans son driver qui utilise l'interruption du convertisseur.

#### 2.6.9.1. AJOUT DANS APP.H

##### 2.6.9.1.1. Modification APP\_STATES

Remplacement des états liés au driver ADC par ceux proposés dans l'exemple 1.

Situation de l'exemple :

```

typedef enum
{
    /* Application's state machine's initial state. */
    APP_STATE_INIT=0,
    APP_STATE_SPIN,
    APP_STATE_UPDATE_ADC_AVERAGE
} APP_STATES;

```

Nouvelle situation :

```

typedef enum
{
    /* Application's state machine's initial state. */
    APP_STATE_INIT=0,
    APP_STATE_WAIT,          // CHR ajout de WAIT
    APP_STATE_SERVICE_TASKS,
} APP_STATES;

```



### 2.6.9.1.2. Modification de APP\_DATA

On remplace la situation :

```
typedef struct
{
    /* The application's current state */
    APP_STATES state;
    /* Values for the conversions */
    int potValue;
    int ledMask;
    bool dataReady;
} APP_DATA;
```

Par :

```
typedef struct
{
    /* The application's current state */
    APP_STATES state;
    /* Valeur des ADC */
    S_ADCResults AdcRes;
} APP_DATA;
```

Besoin d'un include :

```
#include "Mc32DriverAdc.h"
```

### 2.6.9.1.3. Ajout du prototype de la fonction APP\_UpdateState

Ajout tout à la fin du fichier :

```
void APP_UpdateState ( APP_STATES NewState ) ;
```

### 2.6.9.2. AJOUT/MODIFICATION DANS APP\_TASKS

Pour vérifier le fonctionnement de l'ADC et du LCD, nous introduisons les éléments suivants dans l'application:

```
void APP_Tasks ( void )
{
    /* check the application state*/
    switch ( appData.state )
    {
        /* Application's initial state. */
        case APP_STATE_INIT:
            // Init  ADC
            BSP_InitADC10();
            // Init du LCD
            lcd_init();
            lcd_bl_on();
            // Start du Timer1
            DRV_TMR0_Start();
            printf_lcd("App Exemple adc_pot");
            lcd_gotoxy(1,2);
            printf_lcd("C. Huber 14.11.2016");
    }
```

```

        appData.state = APP_STATE_WAIT;
        break;

    case APP_STATE_WAIT:
    {

    }
        break;

    case APP_STATE_SERVICE_TASKS:
    {
        BSP_LEDToggle(BSP_LED_1);
        appData.AdcRes = BSP_ReadAllADC();
        lcd_gotoxy(1,3);
        printf_lcd("Ch0      %4d      Ch1      %4d",
appData.AdcRes.Chan0, appData.AdcRes.Chan1);
        appData.state = APP_STATE_WAIT;
    }
        break;

    /* The default state should never be executed. */
    default:
        break;
    }
}

```

### 2.6.9.3. AJOUT DANS APP.C

Nous avons besoin de :

```
#include "Mc32DriverLcd.h"
```

Et de :

```
#include "driver/tmr/drv_tmr_static.h"
```

Pour faire disparaître le warning :

```

"C:\Program Files (x86)\Microchip\xc32\v1.42\bin\xc32-gcc.exe" -g -x c -c -mprocessor=32MX795F512L -ffunction-sections
../src/app.c: In function 'APP_Tasks':
../src/app.c:157:13: warning: implicit declaration of function 'DRV_TMR0_Start' [-Wimplicit-function-declaration]
    DRV_TMR0_Start();
    ^

```

☹ Ce qui est bizarre, car normalement cela devrait être fait automatiquement !

Et de la fonction APP\_UpdateState

```

void APP_UpdateState ( APP_STATES NewState )
{
    appData.state = NewState;
}

```

#### 2.6.9.4. MODIFICATION DANS SYSTEM\_INTERRUPT

Ajout du mécanisme permettant d'activer l'application toutes les 100 ms.

```
void __ISR(_TIMER_1_VECTOR, ip13AUTO)
    _IntHandlerDrvTmrInstance0(void)
{
    static int16_t count = 0;

    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);

    count++;
    // Attente 2 secondes, puis cycle de 100 ms
    if (count > 2000 ) {
        // Etablit etat d'execution
        APP_UpdateState ( APP_STATE_ SERVICE_TASKS );
        count = 1900;
    }
    // Test de la période du timer1
    BSP_LEDToggle(BSP_LED_0);
}
```

#### 2.6.9.5. TEST DE FONCTIONNEMENT

On obtient bien l'affichage de la valeur des 2 pots ainsi que l'inversion de la LED\_0 toutes les 1 ms et celle de la LED\_1 toutes les 100 ms.

#### 2.6.9.6. ADAPTATION EXEMPLE, CONCLUSION

Cela montre que notre BSP s'est bien intégré dans cet exemple assez simple. Mais que par contre la modification d'une configuration existante présente quelques petits défauts que la création à partir de zéro n'a pas.

## 2.7. GESTION DES PROJETS HARMONY

### 2.7.1. SAUVEGARDE D'UN PROJET

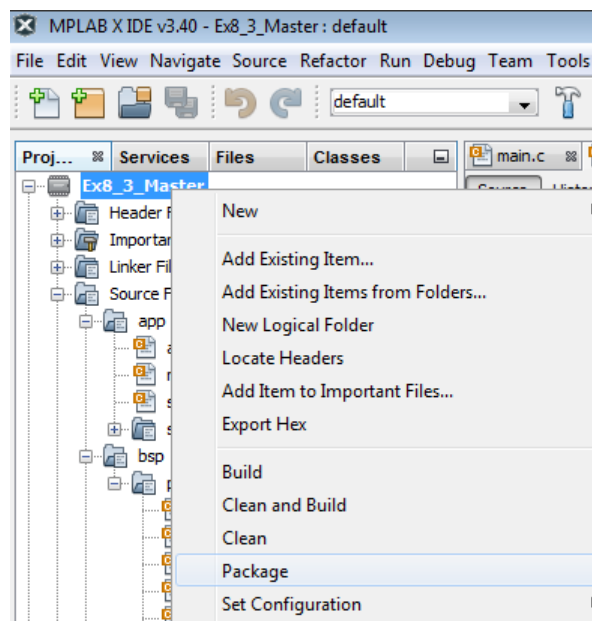
Il s'agit ici de définir une méthode de travail pour sauvegarder un projet Harmony afin d'y inclure tout le nécessaire à la reprise d'un projet.

Voici les étapes qui permettent d'archiver un projet MPLAB X.

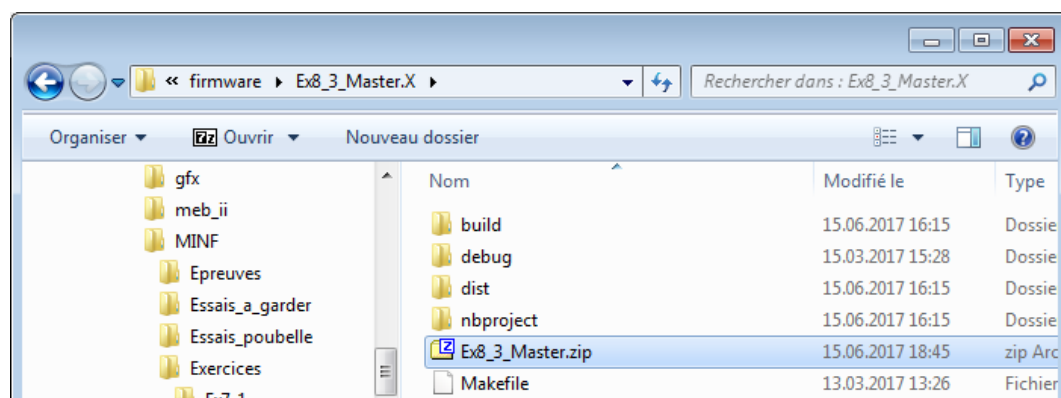
#### 2.7.1.1. FONCTION DE PACKAGE

La **fonction "package"** de MPLAB X permet de zipper simplement tous les fichiers de code source du projet, y compris ceux qui ne se trouvent pas dans le répertoire du projet (BSP).

Elle est accessible avec un simple clic droit sur le projet :



Il en résulte la création d'un fichier zip dans le répertoire du projet :

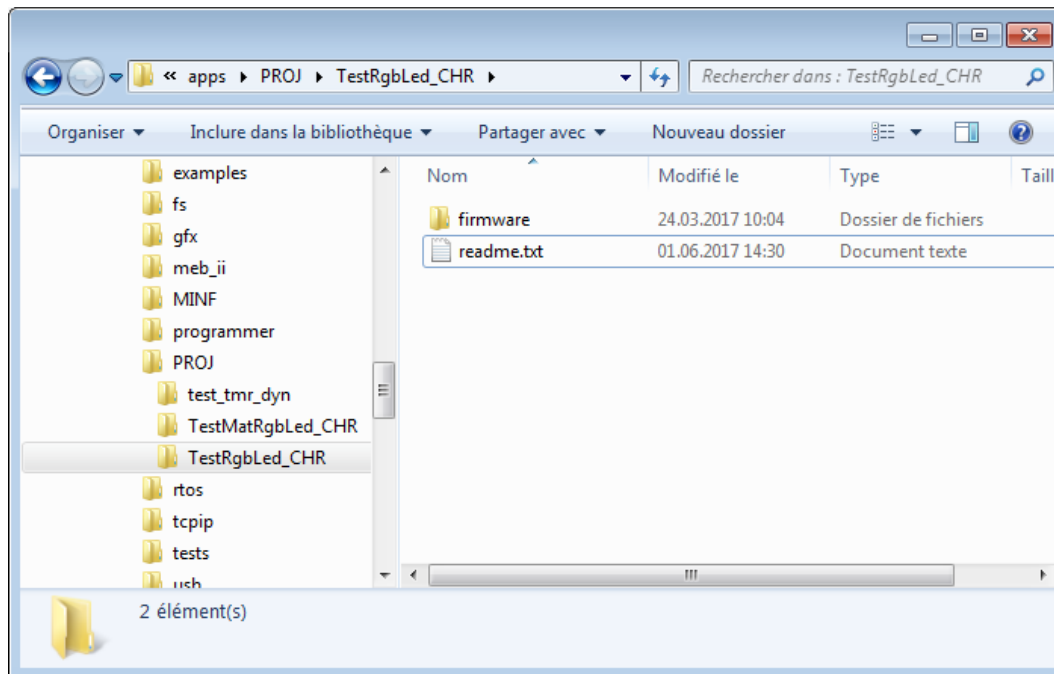


Sauvegarder uniquement les fichiers de code source ne suffit pas. Il manque par exemple le fichier de configuration du MHC et les fichiers web (dans le cas d'un projet serveur web).

### 2.7.1.2. AJOUT D'UN FICHIER DESCRIPTIF DE PROJET

Dans tous les cas, une bonne pratique est d'ajouter un fichier readme listant les versions de logiciels (MPLAB X, xc32, Harmony).

Ajouter un simple **fichier texte "readme"** dans le répertoire du projet :



Y inclure au minimum :

- L'emplacement d'origine du projet dans l'arborescence du disque.
- Les versions des logiciels utilisés.
- Nom de l'auteur et date.

Exemple de readme.txt :

```
Emplacement de ce fichier:
C:\microchip\harmony\v1_08_01\apps\MINF\Exercices\Ex8_3_Master

Versions des logiciels:
MPLABX 3.40
xc32 1.42
Harmony 1.08

Auteur:
15.06.2017
S. Castoldi
```

Cela permettra le cas échéant de réinstaller les mêmes versions de logiciels et de restaurer le projet au bon niveau hiérarchique.

### 2.7.1.3. SAUVEGARDE DU REPERTOIRE DU PROJET

**Après avoir réalisé les 2 étapes précédentes**, on peut **copier le répertoire complet du projet** dans la sauvegarde.

Ce répertoire inclut le zip et le fichier texte de descriptif créé précédemment.

Tout le nécessaire à la reprise du projet est donc facilement copié en même temps que le répertoire.

### 2.7.1.4. SAUVEGARDE DU BSP

Dans certains cas, il peut s'avérer nécessaire de sauvegarder le BSP du projet. Par exemple dans le cas d'un projet utilisant un BSP non standard.

Dans ce cas, il faut :

- Copier dans la sauvegarde le répertoire des sources du BSP, par exemple :  
<Répertoire Harmony>\v<n>\bsp\bsp\_1526x\_CmdBTDirigeable
- Copier le fichier .hconfig de configuration incluant la référence au BSP. Par exemple :  
<Répertoire Harmony>\v<n>\bsp\DS60001143.hconfig

Sans le BSP correct et son fichier de configuration, on ne pourra pas relancer le MHC lors de la reprise du projet.

### 2.7.1.5. SAUVEGARDE D'UN EXERCICE OU TP

Pour les besoins des étudiants, lorsqu'il s'agit de sauvegarder un projet simple réalisé avec un BSP standard (par exemple sur le starter-kit ES), tels qu'un exercice ou un TP, on peut se contenter de sauvegarder le répertoire racine du projet.

Remarque : il est important lors de la création des exercices et des TP de les placer dans les sous-répertoires prévus. Si ce n'est pas le cas, un readme est nécessaire pour avoir le chemin.

## **2.7.2. RESTAURATION D'UN PROJET**

Si on souhaite reprendre un projet, on procédera ainsi :

### **2.7.2.1. COPIE DU PROJET DANS L'ARBORESCENCE HARMONY**


En se basant sur le fichier readme, copier le répertoire racine du projet en recréant si nécessaire sous \apps de la bonne version d'Harmony les mêmes-répertoires.

Dans le cas où la version d'Harmony a évolué entretemps, on peut soit porter le projet vers la nouvelle version, soit installer la bonne version.

Dans le cas où le projet contenait des fichiers sources non standards, ils se trouvent dans le package réalisé. Les restaurer au bon endroit.

### **2.7.2.2. COPIE DU BSP**

Si le projet utilisait un BSP spécifique, il faut alors également restaurer ce dernier :

- Restaurer les sources du BSP :  
Il suffit de copier le répertoire correspondant au nom du BSP sous  
<Répertoire Harmony>\v<n>\bsp
- Patcher le fichier de configuration des BSP :  
 Attention : C'est là le point délicat. Il faut inclure dans le fichier .hconfig du poste la référence au BSP. Il ne faut pas copier le fichier (par exemple DS60001xxx.hconfig), mais il plutôt ajouter les infos spécifiques au BSP dans le fichier original. Ceci dans le but de ne pas écraser les infos d'un autre BSP ajouté.

### **2.7.2.3. RESTAURATION D'UN EXERCICE OU TP**

Pour les besoins des étudiants, lorsqu'il s'agit de restaurer un projet simple réalisé avec un BSP standard (par exemple sur le starter-kit ES), tels qu'un exercice ou un TP, il suffit de recopier le répertoire racine du projet sauvegardé au bon endroit dans l'arborescence de la bonne version d'Harmony.

### 2.7.3. BSP PIC32MX\_SKES

Le BSP spécifique au kit PIC32MX795F512L (**pic32mx\_skes**) se trouve sous :  
 ...\\Maitres-Eleves\\SLO\\Modules\\SL229\_MINF\\Harmony\_ES\\v<n>

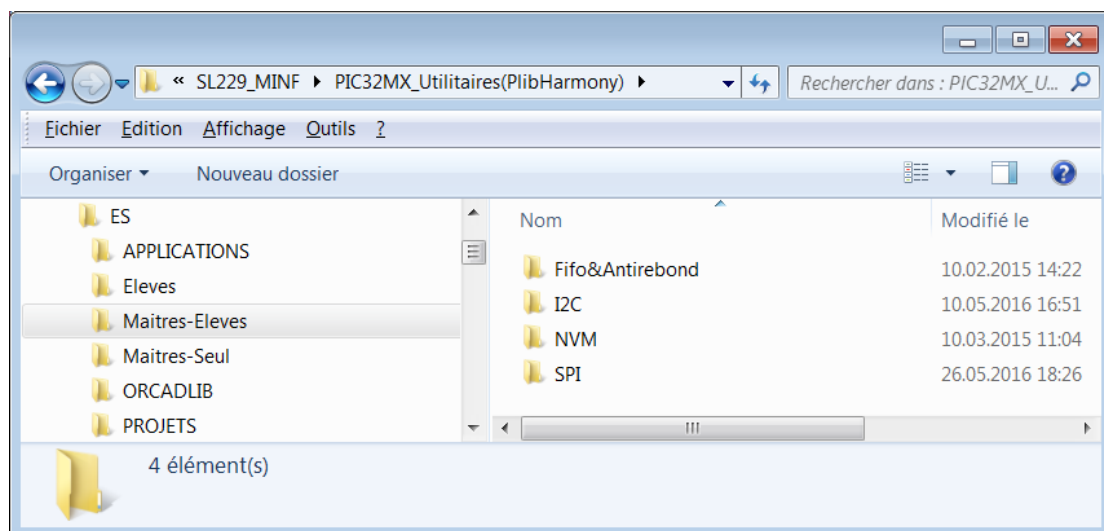
Mise à jour de la liste des BSP :

Il est nécessaire de modifier un fichier Harmony, cela s'effectue en copiant le fichier déjà modifié **DS60001156.hconfig** de  
 ...\\Maitres-Eleves\\SLO\\Modules\\SL229\_MINF\\Harmony\_ES\\v<n>\\config sous  
 <Répertoire Harmony>\\v<n>\\bsp\\config

☛ On modifie un fichier unique qui provient de l'installation de Harmony ! Avant de copier, s'assurer que l'on n'a pas ajouté un autre BSP de la même famille.

### 2.7.4. UTILITAIRES APPLICATION

Les utilitaires pour l'application se trouvent sous  
 ...\\Maitres-Eleves\\SLO\\Modules\\SL229\_MINF\\PIC32MX\_Utilitaires(PlibHarmony)



☞ Ce répertoire sera mis à jour si nécessaire en fonction de la version de Harmony et de l'évolution des versions des librairies.

## 2.8. CONCLUSION

Ce chapitre, appelé à évoluer au fur et à mesure de l'expérience acquise et de l'évolution des logiciels fournis par Microchip, a proposé deux approches : l'une permettant de réaliser des projets avec le MHC, l'autre d'être à même de modifier des projets exemples en vue de mettre en œuvre l'USB ou de réaliser un Web Server.



## **2.9. HISTORIQUE DES VERSIONS**

### **2.9.1. V1.0 JANVIER 2013**

Création du document.

### **2.9.2. V1.5 SEPTEMBRE 2014**

Refonte complète du document suite à l'introduction de Harmony V1.00

### **2.9.3. V1.6 SEPTEMBRE 2015**

Adaptation du document suite à l'introduction de Harmony V1.06 et du MPLAB X 3.06.  
Introduction de 2 approches : génération de l'application avec le MHC et adaptation d'un exemple Microchip.

### **2.9.4. V1.7 NOVEMBRE 2016**

Adaptation du document suite à l'introduction de Harmony V1.08 et du MPLAB X 3.40.  
Ajout de la méthode de sauvegarde et restauration des projets avec BSP spécifique.

### **2.9.5. V1.8 NOVEMBRE 2017**

Reprise et relecture par SCA.  
Adaptation de la partie sauvegarde/restauration d'un projet.

### **2.9.6. V1.81 NOVEMBRE 2018**

Précisions installation environnement et relecture en relation avec les versions suivantes : MPLAB X 4.15, XC32 2.05 et Harmony 2.05.01.