

MINF
Mise en œuvre
des microcontrôleurs PIC32MX

Chapitre 3

**Prise en main MPLAB X et ICD,
programmation et debug**

✕ T.P. PIC32MX

Christian HUBER (CHR)
Serge CASTOLDI (SCA)
Version 1.81 novembre 2021

CONTENU DU CHAPITRE 3

3. Prise en main MPLAB X et ICD	3-1
3.1. Réglages et première utilisation	3-1
3.1.1. Mise à jour du firmware de l'ICD	3-2
3.1.2. Utilisation d'un autre ICD	3-3
3.1.3. Affichage lors du chargement du programme	3-3
3.1.4. Configuration du projet	3-4
3.1.4.1. Niveau d'optimisation	3-4
3.1.4.2. Avertissements à la compilation	3-5
3.2. Exécution du projet avec ICD	3-6
3.2.1. Run main project	3-6
3.2.2. Debug Main Project	3-6
3.2.2.1. Pause/Continue	3-7
3.2.2.2. Vue d'ensemble des actions de Debug	3-7
3.2.2.3. Observations durant la pause	3-8
3.2.2.4. Reprise de l'exécution	3-8
3.2.2.5. Exécution pas à pas	3-8
3.2.2.6. Run To Cursor	3-10
3.2.2.7. Utilisation des points d'arrêts (breakpoints)	3-11
3.2.2.8. Obtention d'un listing assembleur	3-12
3.2.2.9. Fin de la session de debug	3-14
3.3. Utilisation ICD avec IPE	3-15
3.3.1. Lancement de l'IPE	3-15
3.3.2. Configuration	3-16
3.3.3. Sélection du fichier Hex	3-17
3.3.4. Action program	3-17
3.4. Conclusion	3-18
3.5. Historique des versions	3-18
3.5.1. V1.0 février 2013	3-18
3.5.2. V1.1 février 2014	3-18
3.5.3. V1.2 mars 2014	3-18
3.5.4. V1.5 octobre 2014	3-18
3.5.5. V1.6 septembre 2015	3-18
3.5.6. V1.7 novembre 2016	3-18
3.5.7. V1.8 novembre 2017	3-18
3.5.8. V1.81 novembre 2021	3-18

3. PRISE EN MAIN MPLAB X ET ICD

Dans ce chapitre, nous allons étudier comment installer et configurer une sonde de debug (ICD – In-Circuit Debugger) pour exécuter et debugger les programmes depuis MPLAB X.

Ce document a été réalisé sur la base de l'utilisation d'une sonde de Microchip ICD3. Cette sonde n'est plus disponible à la vente. Sa remplaçante, l'ICD4 ou un autre modèle moins coûteux, le SNAP, peuvent être utilisés à la place et sont suffisants pour les besoins de ce cours. Du point de vue de l'intégration de la sonde dans l'IDE MPLAB X, une fois les réglages effectués, l'utilisation est la même.

Le présent document reprend les principes utilisés dans le document :

...\PROJETS\SLO\1102x_SK32MX775F512L\Hardware\11020_SK32MX775F512L\B\Software_local\SK32MX795F512L\CahierLabo\SK32MX775F512L_MPLAB.pdf

Ce document avait été établi sur la base de l'environnement précédemment disponible à MPLAB X, soit MPLAB avec le compilateur C32.

Remarque : Les exemples de programmes utilisés dans le présent document sont antérieurs à l'introduction de Harmony. Les explications sur l'utilisation des actions de debugging correspondent à MPLABX version 2.X.

3.1. REGLAGES ET PREMIERE UTILISATION

L'ICD (In-Circuit Debugger) est le périphérique qui fait le lien entre le PC avec MPLABX et le PIC32. Cette sonde peut être utilisée pour programmer et débbugger le PIC32.

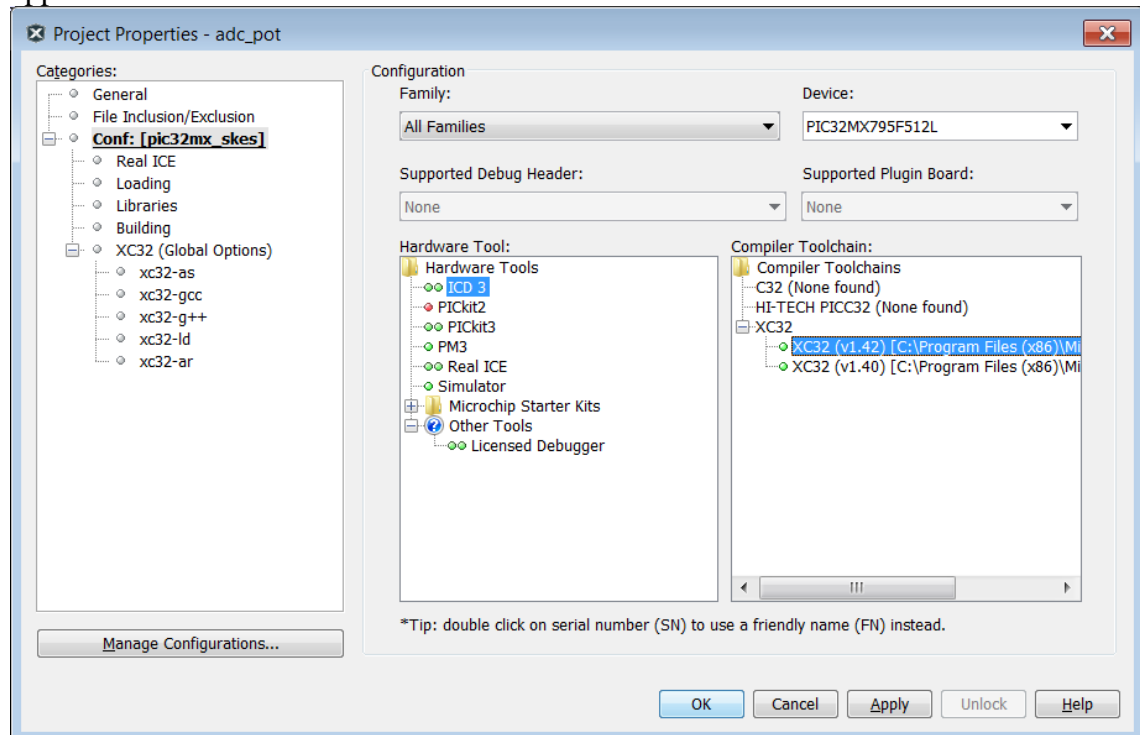


Sonde ICD3

Les signaux nécessaires à la communication avec le uC sont :

- La masse GND
- Le reset /MCLR
- Les 2 signaux de communication PGECn et PGEDn. Suivant le modèle de uC utilisé, plusieurs paires de ce signaux peuvent être à disposition.
- L'alimentation (optionnelle)

Il existe différents types de sondes de programmation/debug pour les PIC32. Lors de la création du projet, il faut aller dans les options du projet afin de sélectionner le bon appareil :



3.1.1. MISE A JOUR DU FIRMWARE DE L'ICD

Cette situation se produit par exemple lorsque l'on utilise un ICD3 avec une nouvelle version de MPLABX. La programmation du uC prend alors plus de temps qu'à l'habitude, car la mise à jour de l'ICD3 est réalisée tout d'abord. Cela est normal et n'a lieu que la première fois.

```

Output
ExampleIO (Build, Load, ...) ICD 3

Connecting to MPLAB ICD 3...

Currently loaded firmware on ICD 3
Firmware Suite Version.....01.35.16 *
Firmware type.....PIC32MX

Now Downloading new Firmware for target device: PIC32MX795F512L
Downloading bootloader
Bootloader download complete
Programming download...
Downloading RS...
RS download complete
Programming download...
Downloading AP...
AP download complete
Programming download...

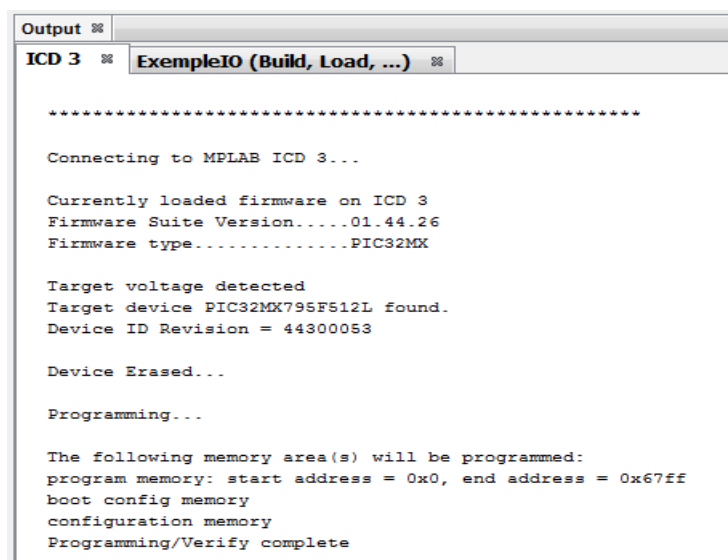
```

3.1.2. UTILISATION D'UN AUTRE ICD

Le projet mémorise le no de série de l'ICD utilisé. Si on change d'ICD (lors de la reprise d'un projet par exemple), on obtient un message d'avertissement qui demande si on veut utiliser une autre sonde.

3.1.3. AFFICHAGE LORS DU CHARGEMENT DU PROGRAMME

Lors des chargements suivants du programme, l'affichage suivant ("Programming/Verify complete") indique que le uC est prêt :



```
Output %
ICD 3 % ExempleIO (Build, Load, ...) %

*****

Connecting to MPLAB ICD 3...

Currently loaded firmware on ICD 3
Firmware Suite Version.....01.44.26
Firmware type.....PIC32MX

Target voltage detected
Target device PIC32MX795F512L found.
Device ID Revision = 44300053

Device Erased...

Programming...

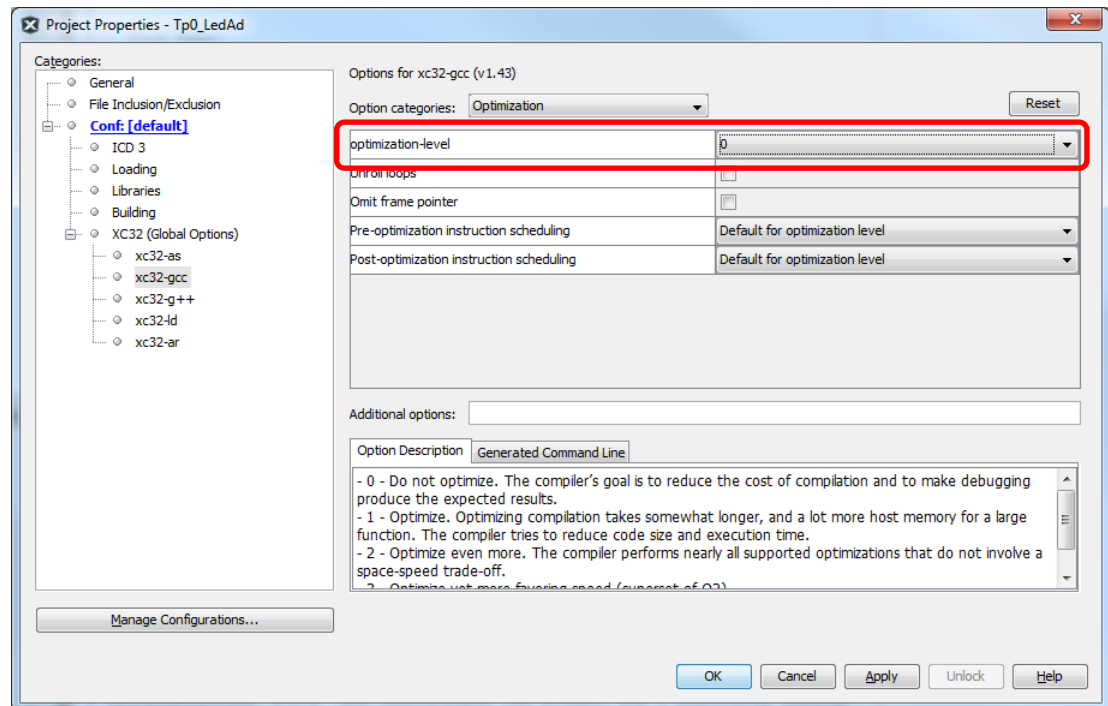
The following memory area(s) will be programmed:
program memory: start address = 0x0, end address = 0x67ff
boot config memory
configuration memory
Programming/Verify complete
```

3.1.4. CONFIGURATION DU PROJET

3.1.4.1. NIVEAU D'OPTIMISATION

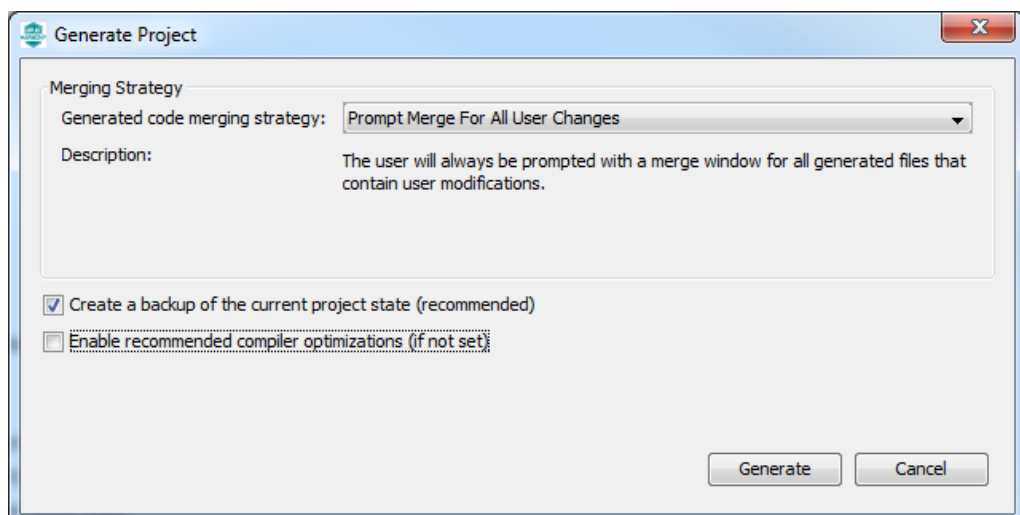
Dans les propriétés, sous xc32-gcc > optimization, par défaut le niveau d'optimisation est 1.

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



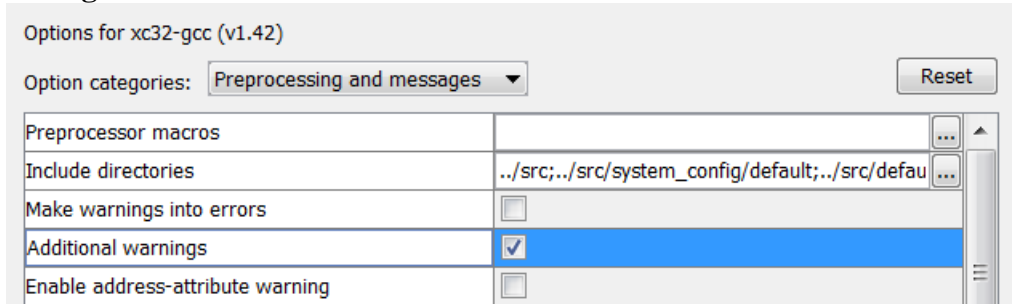
Un niveau d'optimisation à zéro ne sera pas toujours possible avec les exemples plus volumineux, comme ceux démontrant l'ethernet ou l'USB.

👉 Lors de la génération du code avec le MHC, il faut enlever la coche 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.



3.1.4.2. AVERTISSEMENTS A LA COMPILATION

Sous xc32-gcc > preprocessing and messages, **ajouter la coche "Additional warnings"** :



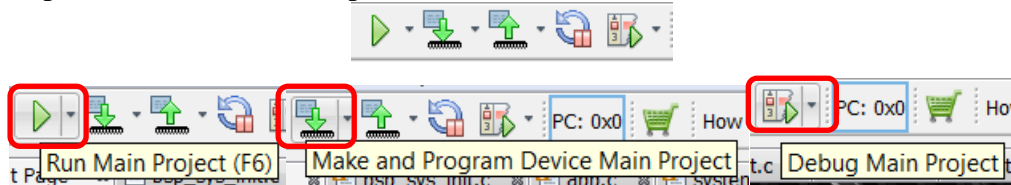
Cela permet d'obtenir un meilleur contrôle du code, comme les variables non utilisées et les références indirectes, et donc aide à rendre le code plus sûr.

3.2. EXECUTION DU PROJET AVEC ICD

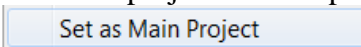
Il y a 2 possibilités :

- Exécution sans debugging (**Menu Run**, Run Main Project ou F6),
- Exécution avec debugging (**Menu Debug**, Debug main project).

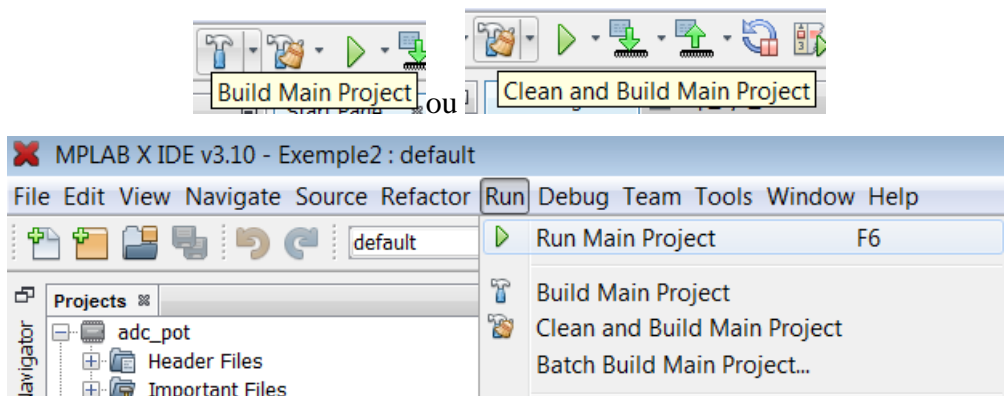
En plus des 2 menus, il est possible de déclencher directement les actions avec :



3.2.1. RUN MAIN PROJECT

Remarque : si on dispose de plusieurs projets, sélectionnez le projet voulu et par un clic droit et obtenez le menu déroulant. Activez : 

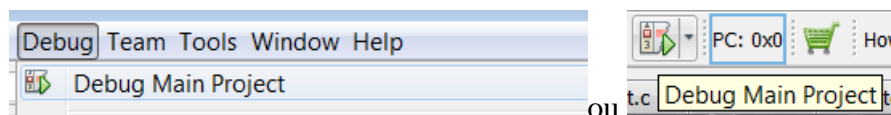
Exécution du projet sans debugging : Il suffit à partir du menu Run d'activer **Run Main Project**. Action directe avec F6, ceci après avoir effectué :



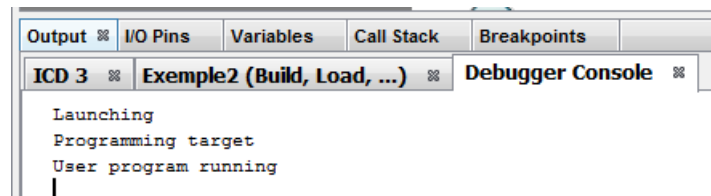
😊 Le programme est chargé de manière non-volatile. Après coupure d'alimentation et ré-enclenchement le programme est toujours présent en mémoire et s'exécute.

3.2.2. DEBUG MAIN PROJECT

Exécution du projet avec debugging. Il suffit à partir du menu **Debug** d'activer : **Debug Main Project**.



On obtient dans la fenêtre de sortie avec l'onglet Debugger Console, les informations suivantes :



Au niveau de la barre d'outils il y a davantage d'actions :



3.2.2.1. PAUSE/CONTINUE

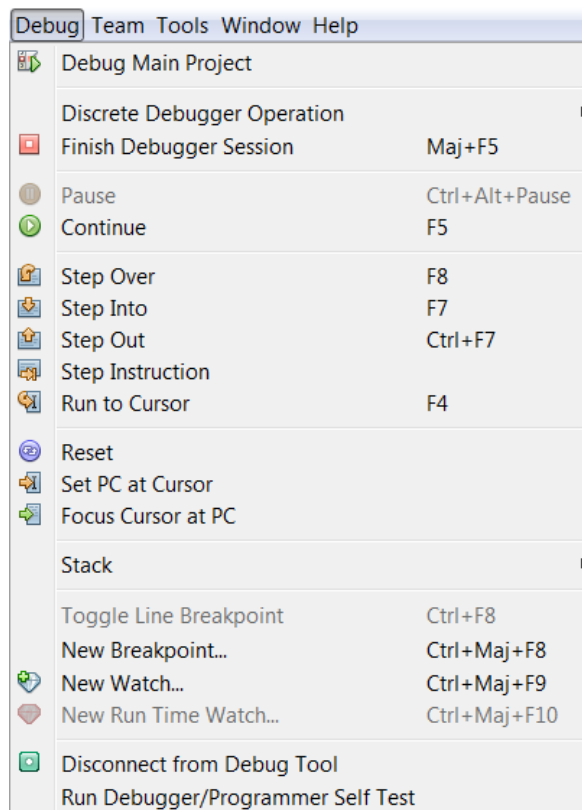
Pour stopper momentanément l'exécution du programme il faut activer



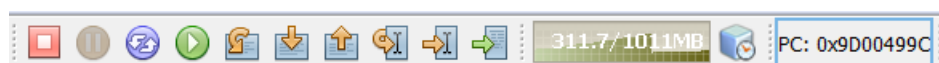
au niveau du menu Debug ou dans la barre d'outils.

3.2.2.2. VUE D'ENSEMBLE DES ACTIONS DE DEBUG

Dans cette situation de pause, la liste des actions de debugging disponibles est plus importante :



Et au niveau de la barre d'outils :



3.2.2.3. OBSERVATIONS DURANT LA PAUSE

En plaçant le curseur sur une variable, on obtient son adresse et sa valeur.

```



177
178     while(1) {
179         // Ne r
Address = 0xA0000204,  compteurMain = 0x002608E2
compteurMain++;
181     }

```

Dans la fenêtre Variables, il est possible d'introduire un nom de variable (Enter New Watch) et d'obtenir les informations.






Variables		Call Stack	
Name	Type	Address	Value
ChenillardPhase	int	0xA0000200	0x00000002
<Enter new watch>			
compteurMain	int	0xA0000204	0x002608E2

3.2.2.4. REPRISE DE L'EXECUTION

Pour reprendre l'exécution on utilise  Continue F5 au niveau du menu Debug ou  au niveau de la barre d'outils.

3.2.2.5. EXECUTION PAS A PAS

Lorsque l'on est en pause, il est possible d'avancer en pas à pas. Le menu Debug nous indique :

	Continue	F5
	Step Over	F8
	Step Into	F7
	Step Instruction	
	Run to Cursor	F4

3.2.2.5.1. Step Into



Il s'agit d'un pas à pas qui entre dans les fonctions et exécute pas à pas le contenu de la fonction.

3.2.2.5.2. Step Over



Il s'agit d'un pas à pas qui exécute les fonctions comme une action unique.

Dans l'extrait ci-dessous, le Step Over exécute pas à pas chaque ligne. La ligne 161 qui correspond à la fonction compte() est exécutée comme un pas.

```

158     compte ();
159     LED_D9_W = 1; //éteint
160     LED_D10_W = 0; //allumé
161     compte ();
162     LED_D10_W = 1; //éteint
163     LED_D11_W = 0; //allumé

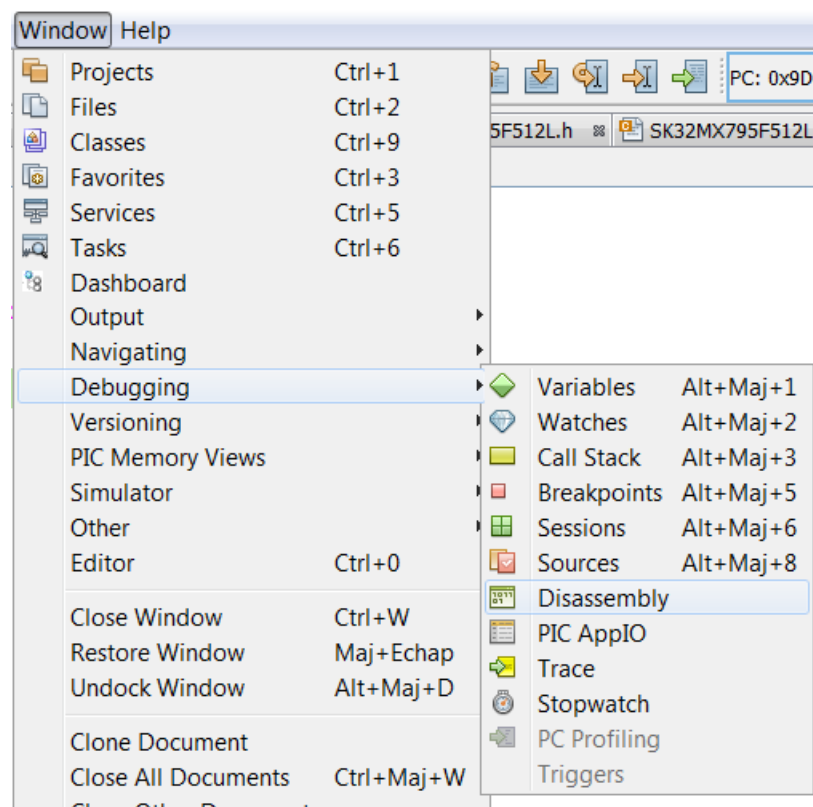
```

3.2.2.5.3. Step Instruction

Il s'agit d'un pas à pas qui exécute chaque instruction (assembleur) pas à pas. Il se déclenche par le menu Debug

	Step Over	F8
	Step Into	F7
	Step Instruction	
	Run to Cursor	F4

Il faut utiliser le menu Window, Debugging, Disassembly.



On obtient (situation avant le Step):

```

27 !
28 ! // Chenillard sous interruption
29 ! // De Led0 à Led5
30 ! switch (ChenillardPhase) {
    ⇨ 0x9D0013E8: LW V0, -32752(GP)
32 0x9D0013EC: SLTIU V1, V0, 6
33 0x9D0013F0: BEQ V1, ZERO, 0x9D001570
34 0x9D0013F4: NOP
35 0x9D0013F8: SLL V1, V0, 2
36 0x9D0013FC: LUI V0, -25344
37 0x9D001400: ADDIU V0, V0, 5140
38 0x9D001404: ADDU V0, V1, V0

```

Après 1 Step Instruction :

```

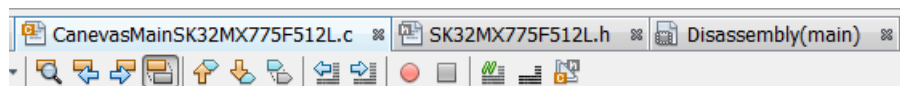
27 !
28 ! // Chenillard sous interruption
29 ! // De Led0 à Led5
30 ! switch (ChenillardPhase) {
31 0x9D0013E8: LW V0, -32752(GP)
    ⇨ 0x9D0013EC: SLTIU V1, V0, 6
33 0x9D0013F0: BEQ V1, ZERO, 0x9D001570
34 0x9D0013F4: NOP

```


Et le Step Instruction passe d'une instruction à l'autre.

Remarque : Une instruction C est la plupart du temps découpée en n instructions assembleur.

Pour revenir au .c il faut sélectionner le fichier.



3.2.2.6. RUN TO CURSOR

Dans une situation de pause, il suffit de placer le curseur à l'endroit où l'on veut s'arrêter et d'activer le 

```

194 // De Led0 à Led5
    ⇨ switch (ChenillardPhase) {
196     case 0:
197         LED5_W = 1; //éteint
198         LED0_W = 0; //allumé
199         ChenillardPhase++;
200     break;
201     case 1:
202         LED0_W = 1; //éteint

```

Le programme s'exécute et s'arrête à l'emplacement du curseur.


```
194 // De Led0 à Led5
    switch (ChenillardPhase) {
196     case 0:
197         LED5_W = 1; //éteint
198         LED0_W = 0; //allumé
    ChenillardPhase++;
200     break;
201     case 1:
202         LED0_W = 1; //éteint
```

3.2.2.7. UTILISATION DES POINTS D'ARRETS (BREAKPOINTS)

Pour stopper l'exécution à un endroit précis du programme, le point d'arrêt est très utile (en particulier avec les interruptions).

Il suffit de cliquer dans la colonne grise portant les numéros de lignes.

```
198         LED0_W = 0; //allumé
199         ChenillardPhase++;
200     break;
201     case 1:
202         LED0_W = 1; //éteint
203         LED1_W = 0; //allumé
    ChenillardPhase++;
205     break;
206     case 2:
207         LED1_W = 1; //éteint
208         LED2_W = 0; //allumé
209         ChenillardPhase++;
210     break;
```

Si on exécute le , le programme s'exécute et s'arrête sur le nouveau point d'arrêt.

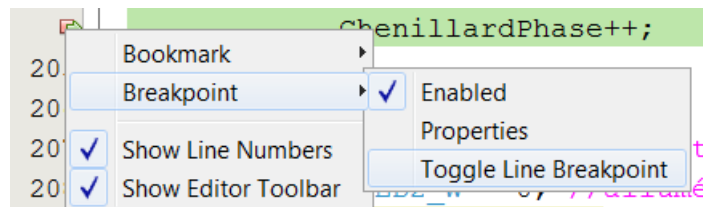
```

194 // De Led0 à Led5
195 switch (ChenillardPhase) {
196     case 0:
197         LED5_W = 1; //éteint
198         LED0_W = 0; //allumé
199         ChenillardPhase++;
200         break;
201     case 1:
202         LED0_W = 1; //éteint
203         LED1_W = 0; //allumé
204         ChenillardPhase++;
205         break;
206     case 2:
207         LED1_W = 1; //éteint

```

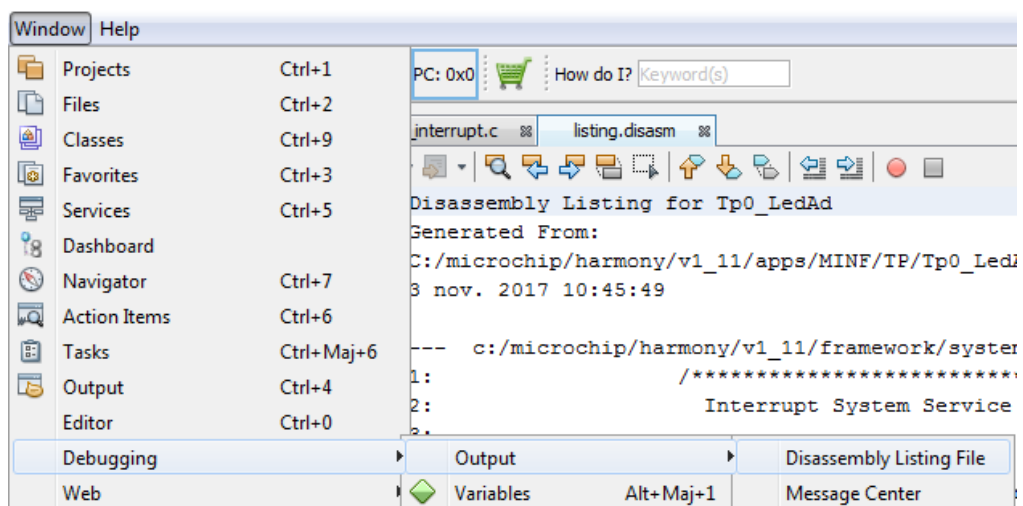
La ligne sur laquelle le programme est arrêté est colorée en vert.

Pour supprimer le break point il suffit de cliquer à nouveau sur le carré marquant le break point. Il est aussi possible à partir d'un clic droit d'activer **Toggle Line Breakpoint**.

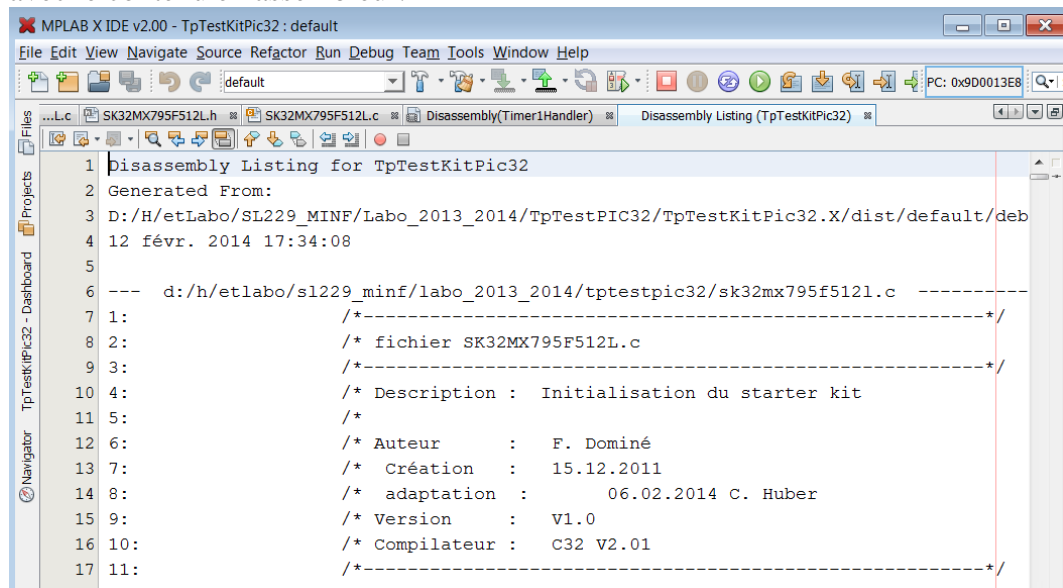


3.2.2.8. OBTENTION D'UN LISTING ASSEMBLEUR

Avec le menu Window > Debugging > Output > Disassembly Listing File:



On obtient un nouvel onglet avec un fichier comportant l'ensemble des fichiers avec le contenu en assembleur.

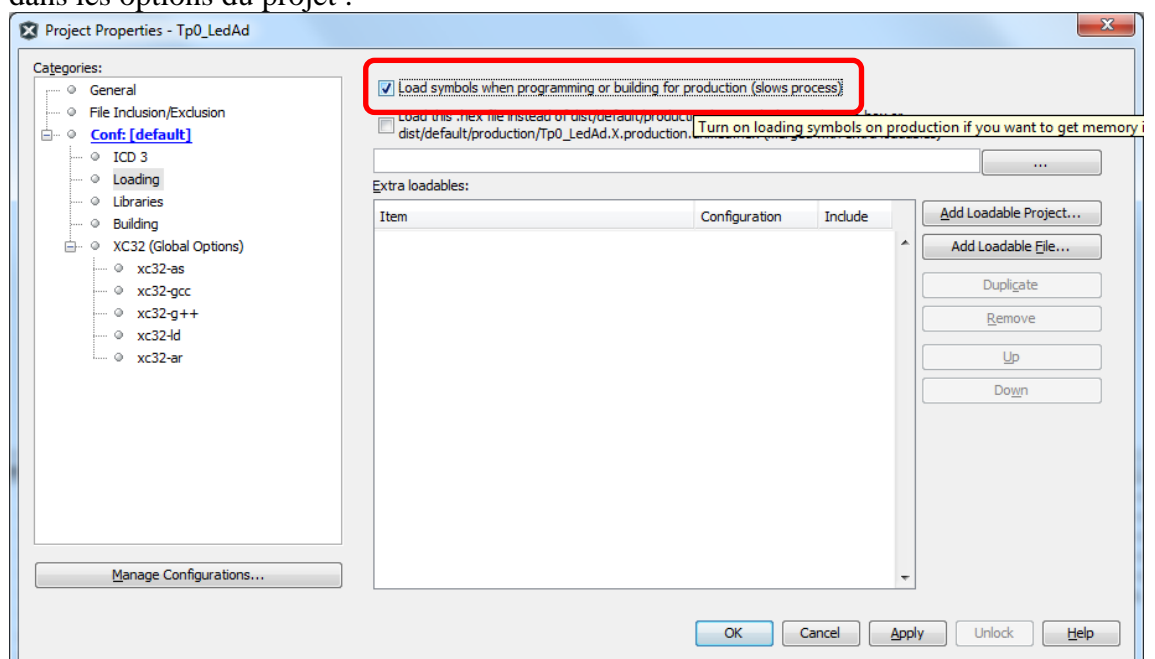


Avec par exemple la boucle du programme principal.

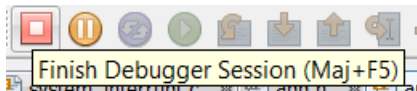
```


1858 178: while(1){
1859 179: // Ne rien faire (juste un comptage
1860 180: compteurMain++;
1861 9D00137C 8F828014 LW V0, -32748(GP)
1862 9D001380 24420001 ADDIU V0, V0, 1
1863 9D001384 AF828014 SW V0, -32748(GP)
1864 181: }
1865 9D001388 0B4004DF J 0x9D00137C
1866 9D00138C 00000000 NOP
1867 182:
1868 183: } // End main
  
```

✎ Pour que listing désassemblé soit disponible, il faut mettre la coche suivante dans les options du projet :

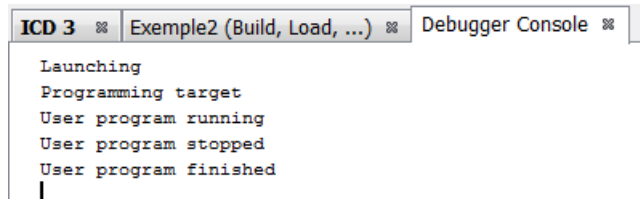


3.2.2.9. FIN DE LA SESSION DE DEBUG



Il faut utiliser :  ou avec le menu Debug.

Au niveau du Debugger Console on obtient les 2 dernières lignes :



👉 Il est recommandé de terminer la session de debug afin d'introduire une modification au programme.

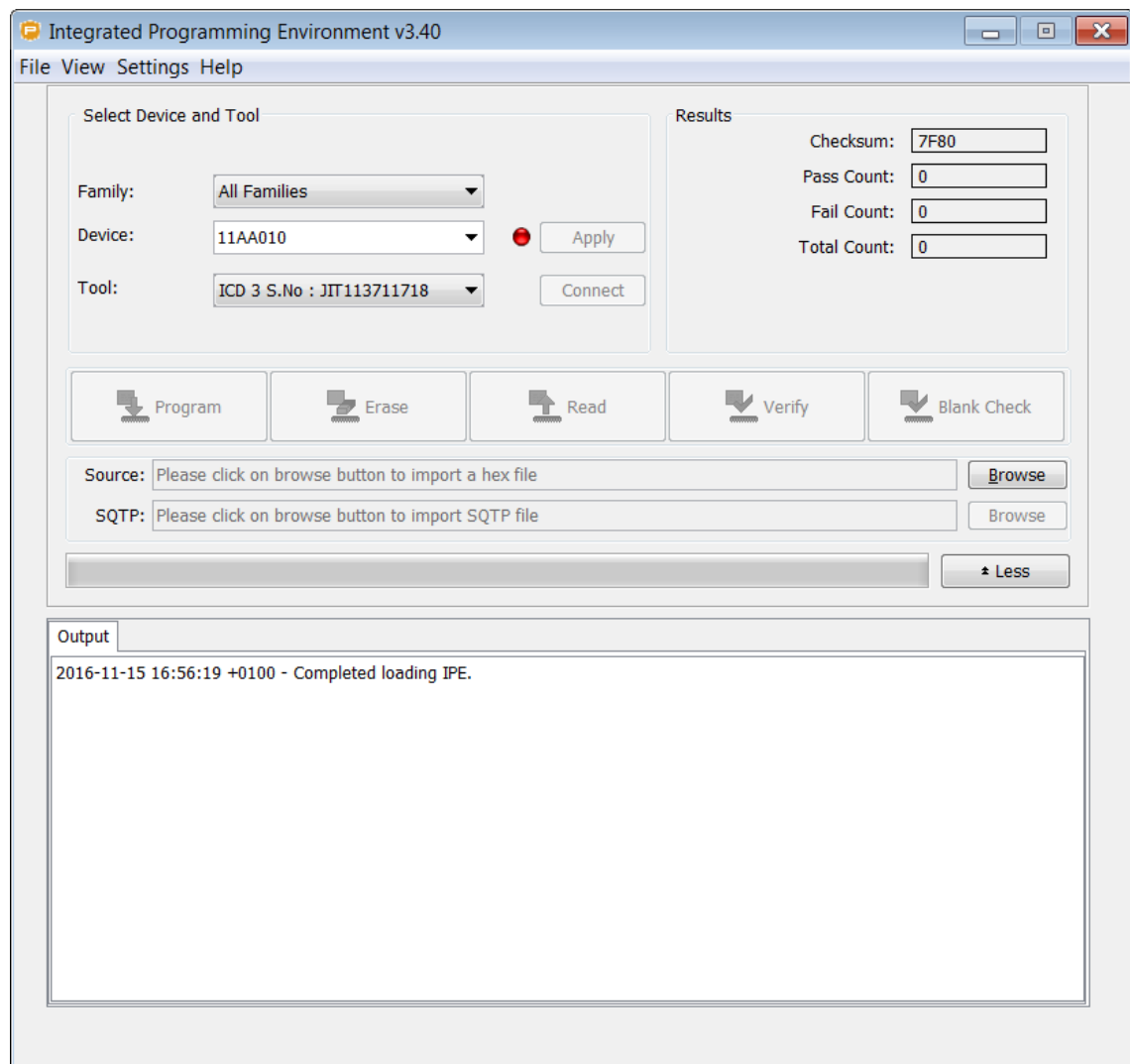
3.3. UTILISATION ICD AVEC IPE

Le nom complet de MPLAB X est "MPLAB X IDE", pour Integrated Development Environment. Cet environnement permet le développement d'applications à partir de leur code source.

L'outil IPE (Integrated Programming Environment) permet de charger un programme sur la base d'un fichier .hex. Ce type de fichier est une norme de représentation du contenu d'une mémoire. Il est généré à la compilation.

3.3.1. LANCEMENT DE L'IPE

Il faut lancer le MPLAB IPE :






3.3.2. CONFIGURATION

Il est nécessaire de sélectionner le modèle de uC, ainsi que l'outil utilisé :

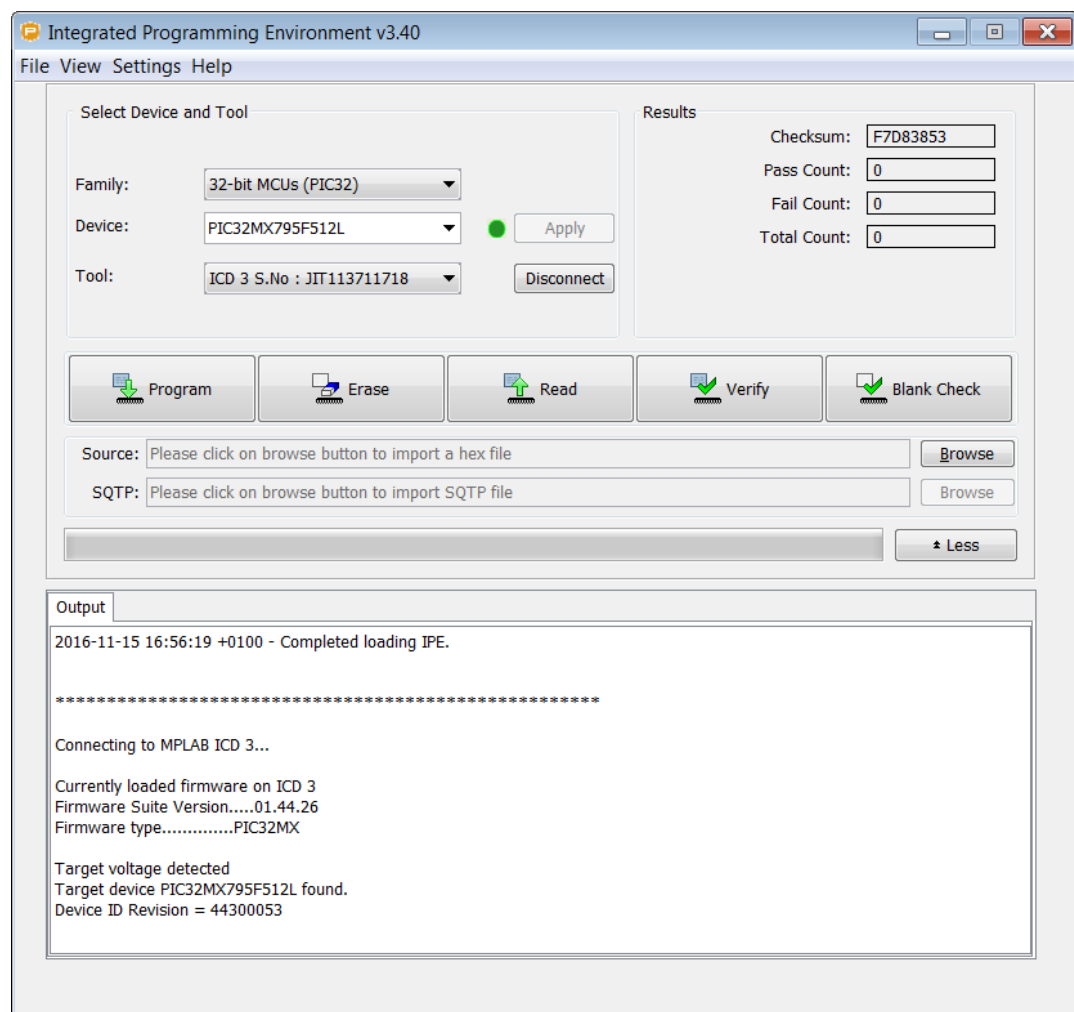
- Sélection Family
- Sélection Device

Family:	32-bit MCUs (PIC32)
Device:	PIC32MX795F512L

Il faut activer   et obtenir le rond vert.

Il faut sélectionner l'ICD et se connecter au uC : .

On obtient :



Différentes actions sont disponibles :

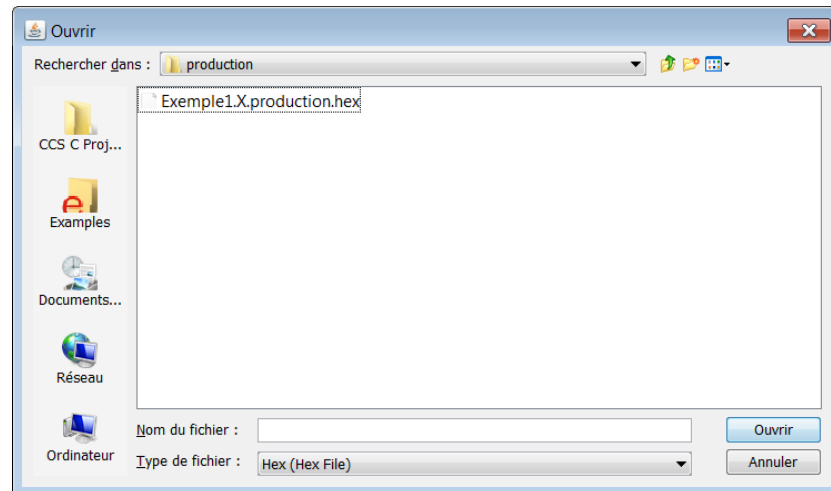
- Programmer la mémoire du uC
- L'effacer
- La lire (par exemple pour copier le programme d'un uC non protégé)
- Vérifier son contenu
- Contrôler qu'il est bien effacé

3.3.3. SELECTION DU FICHIER HEX

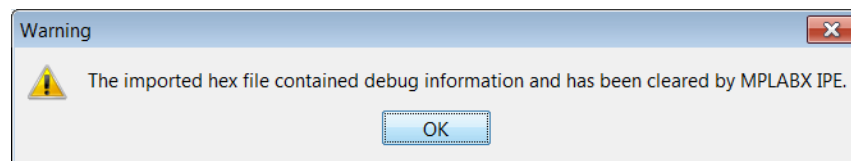
Pour charger un fichier hex, aller sous File > Import > Hex.

Dans un projet Harmony le fichier .hex se trouve dans le répertoire du projet sous dist\default\production

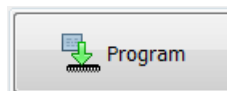
Dans notre exemple :



On peut obtenir :

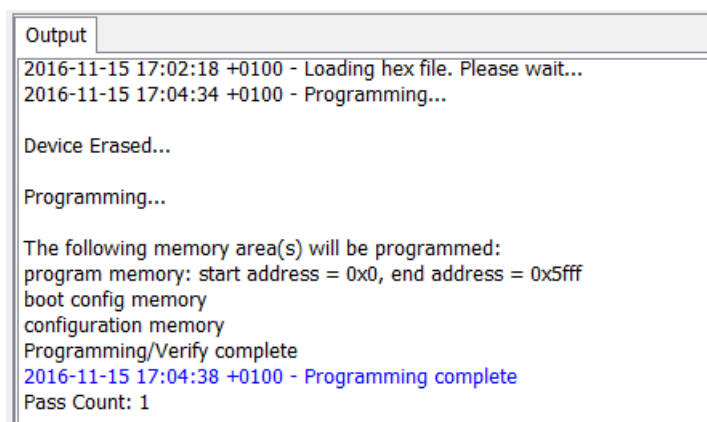


3.3.4. ACTION PROGRAM



Il ne reste plus qu'à utiliser

Avec la quittance dans la fenêtre Output



Le programme s'exécute !

3.4. CONCLUSION

Ce document a présenté les éléments essentiels de l'outil de debugging. Il reste à l'utilisateur à découvrir des détails et astuces complémentaires.

3.5. HISTORIQUE DES VERSIONS

3.5.1. V1.0 FEVRIER 2013

Création du document.

3.5.2. V1.1 FEVRIER 2014

Adaptation à la version 2.0 du MPLABX et 1.31 du XC32. Ajout de compléments et utilisation d'un programme un peu différent.

3.5.3. V1.2 MARS 2014

Ajout procédure installation du driver pour l'ICD3.

3.5.4. V1.5 OCTOBRE 2014

Mise à jour du no de version pour cohérence, mais l'introduction de Harmony n'a pas d'effet sur l'utilisation de l'ICD3.

3.5.5. V1.6 SEPTEMBRE 2015

Mise à jour du no de version pour indiquer l'utilisation de la version 3.X du MPLABX qui a peu d'effet sur l'utilisation de l'ICD3. Conservation d'une partie des exemples avec la version 2.X. Ajout utilisation de l'IPE.

3.5.6. V1.7 NOVEMBRE 2016

Mise à jour du no de version pour indiquer l'utilisation de la version 3.40 du MPLABX qui a peu d'effet sur l'utilisation de l'ICD3. Suppression de la section 1^{ère} installation. Ajout niveau d'optimisation en relation avec facilité de debugging. Mise à jour de la section IPE.

3.5.7. V1.8 NOVEMBRE 2017

Reprise et relecture par SCA.

3.5.8. V1.81 NOVEMBRE 2021

Généralisation ICD3 → ICD