

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

# **Chapitre 1**

## **Introduction**

### **❖ T.P. PIC32MX**

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



## CONTENU DU COURS T.P.

1. Introduction (ce chapitre)
2. Prise en main MPLABX, XC32 et Harmony
3. Prise en main MPLABX et ICD, programmation et debug
4. Gestion des E/S
5. Timers, interruptions et PWM
6. Utilisation des entrées de capture
7. Utilisation de l'USART
8. Gestion du bus I2C (machine d'état)
9. Gestion du bus USB
10. Gestion de l'Ethernet et de TCP/IP



## CONTENU DE L'INTRODUCTION

<i>Remarque préliminaire</i>	<b>1</b>
<b>1. Introduction</b>	<b>2</b>
<b>1.1. Pourquoi le PIC32MX</b>	<b>2</b>
<b>1.2. Apprentissage d'une nouvelle famille de microcontrôleurs</b>	<b>3</b>
<b>1.3. Développement et mise au point des programmes</b>	<b>4</b>
<b>1.4. Harmony : apports et contraintes</b>	<b>4</b>
<b>1.5. Le PIC32MX795F512L en bref</b>	<b>5</b>
1.5.1. Schéma bloc	5
1.5.2. Boîtier et brochage	6
1.5.3. Boîtier et brochage de la version H	7
1.5.4. Quelques caractéristiques	8
1.5.5. USB, Ethernet et CAN selon modèle	9
1.5.6. Organisation de la mémoire	10
<b>1.6. Documentations à disposition</b>	<b>11</b>
1.6.1. Documentation PIC32MX	11
1.6.2. Documentation du Kit PIC32MX	11
1.6.3. Documentation compilateur et environnement	11
1.6.4. Documentation Harmony	12
1.6.5. Documentation dans l'environnement MPLABX	12
<b>1.7. Historique des versions</b>	<b>12</b>
1.7.1. V1.0 2013	12
1.7.2. V1.5	12
1.7.3. V1.6	12
1.7.4. V1.7 octobre 2016	12
1.7.5. V1.8 novembre 2017	12
1.7.6. V1.81 novembre 2021	12



## REMARQUE PRÉLIMINAIRE

Le présent cours est illustré à travers l'utilisation des outils de Microchip, dont notamment le framework Harmony. Au fil de l'évolution des versions, il est possible que quelques différences existent entre la version utilisée lors de l'élaboration et la version actuelle lors de la lecture. Les principes présentés restent applicables.

Dans la mesure du possible, la version utilisée pour les exemples sera indiquée.

Le lecteur est invité à se reporter à la documentation de sa version disponible sous :  
<Répertoire Harmony>\v<n>\doc

## 1. INTRODUCTION

Ce cours a pour objectif d'offrir un support pour les manipulations de laboratoire et les projets utilisant les microcontrôleurs PIC32MX et en particulier le PIC32MX795F512L équipant le starter-kit développé dans le cadre de l'Ecole Supérieure.

### 1.1. POURQUOI LE PIC32MX

L'introduction de l'utilisation d'un microcontrôleur 32 bits moderne dans le cours MINF est indispensable pour permettre aux étudiants d'étendre leurs compétences dans ce domaine. Le choix des familles de microcontrôleurs est vaste.

Dans le domaine des microcontrôleurs 32 bits à usage général, une architecture domine actuellement le marché : les microcontrôleurs à cœurs ARM. Le modèle économique d'ARM est particulier dans le sens où il n'est pas basé sur la fabrication de microcontrôleurs, mais sur la conception de propriétés intellectuelles (IP, Intellectual Properties). Ainsi, de nombreux fabricants intègrent des cœurs ARM dans leurs familles de microcontrôleurs et ajoutent leurs périphériques propres.

Il existe un concurrent, moins connus que ARM : les cœurs MIPS. Bien évidemment, s'agissant du fonctionnement du cœur du microcontrôleur, les concepts généraux sont similaires. Remarquons que le modèle économique de MIPS est également basé sur la revente d'IP.

La famille de microcontrôleurs choisie pour illustrer ce cours est la famille des PIC32MX de Microchip, basée sur un cœur MIPS. Les raisons de ce choix sont multiples :

- Famille 32 bits à usage général.
- Support de l'USB et d'Ethernet.
- Disponibilité d'outils de développement hardware (sonde de debug) et software (compilateur, IDE) gratuits ou peu coûteux.
- Migration depuis la famille précédemment utilisée (PIC18). Quand bien même il s'agissait d'une famille du même fabricant, le cœur était différent (8 bits), et l'IDE utilisé était également différent. Toutefois, la réutilisation des outils de debug et la connaissance de certains périphériques, qu'on retrouve identiques dans le PIC32, facilite la transition.

Voici les spécificités et champs d'applications du PIC32MX (en particulier le PIC32MX795F512L) par rapport aux autres membres de la famille des PIC32 basés sur cœur MIPS :

	Memory Flash/SRAM	Automotive	Connectivity	Functional Safety	Graphics	Motor Control	Security	Ultra-Low Power
<b>PIC32MZ EF FPU</b> MIPS32® M-Class, 252 MHz	512-2048 KB/ 128-512 KB	●	●	●	●		●	
<b>PIC32MZ DA</b> MIPS32 microAptiv™, 200 MHz	1024-2048 KB/ 256-640 KB		●	●	●		●	
<b>PIC32MK</b> MIPS32 microAptiv, 120 MHz	256-1024 KB/ 128-256 KB	●	●	●	●	●		
<b>PIC32MX 3/4</b> MIPS32 M4K®, 80-120 MHz	32-512 KB/ 8-128 KB		●	●				
<b>PIC32MX 5/6/7</b> MIPS32 M4K, 80 MHz	64-512 KB/ 16-128 KB		●	●				
<b>PIC32MX 1/2 XLP</b> MIPS32 M4K, 72 MHz	128-256 KB/ 32-64 KB		●	●			●	
<b>PIC32MX 1/2/5</b> MIPS32 M4K, 50 MHz	16-512 KB/ 4-64 KB		●	●				
<b>PIC32CM MC</b> Arm® Cortex®-M0+, 48 MHz	64-128 KB/ 8-16 KB			●		●		
<b>PIC32MM</b> MIPS32 microAptiv UC, 25 MHz	16-256 KB/ 4-32 KB	●					●	

Tiré de [www.microchip.com](http://www.microchip.com), " 32-bit PIC® Microcontrollers (MCUs) », consulté le 9.11.2021

Notons que Microchip fabrique également des microcontrôleurs à cœur ARM, et que les différentes familles peuvent être programmées à l'aide de l'IDE MPLAB X.

## 1.2. APPRENTISSAGE D'UNE NOUVELLE FAMILLE DE MICROCONTROLEURS

Finalement, le choix d'une famille de microcontrôleurs utilisée pour l'apprentissage et une affaire d'habitudes et de « religion ». Par rapport à une autre famille, les outils changeront, mais les concepts resteront applicables.

Tout changement de famille de microcontrôleurs nécessite forcément une certaine quantité de mises à jour et apprentissage par rapport à n'importe quelle autre famille de microcontrôleurs ou fabricant.

L'utilisation de l'architecture MIPS 32 bits et du compilateur Microchip ainsi que du framework Harmony vont se traduire par un apprentissage d'un nouvel environnement au niveau des détails pour gérer des éléments connus tels que les entrées-sorties, timers, PWM, interruptions et autres.

### **1.3. DÉVELOPPEMENT ET MISE AU POINT DES PROGRAMMES**

Au niveau du compilateur, le choix s'est porté sur le XC32 de Microchip, génération basée sur le compilateur universel gnu et supportant C et C++.

Pour faire le lien entre l'environnement MPLABX IDE et le microcontrôleur cible, une sonde de debug (ICD : In-Circuit Debugger) de Microchip sera utilisée. Cet outil s'intègre naturellement à MPLABX.

### **1.4. HARMONY : APPORTS ET CONTRAINTES**

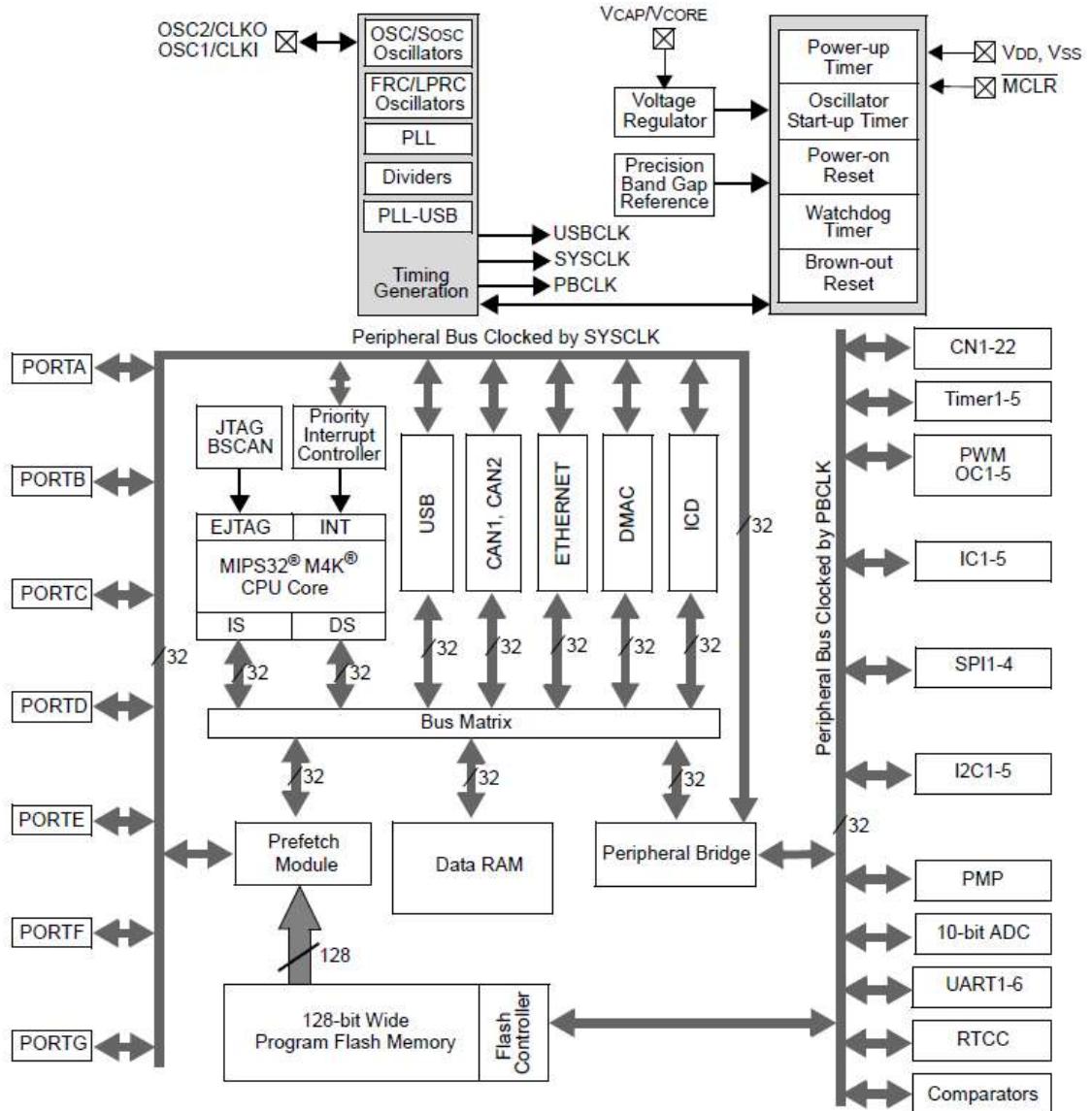
Microchip présente Harmony comme : MPLAB Harmony Integrated Software Framework. Il s'agit d'un ensemble de logiciels comportant principalement les librairies de périphériques, mais aussi des exemples complets permettant par exemple de gérer une pile TCP/IP. Les contraintes sont une structure lourde pour mettre en place un projet ainsi qu'une évolution des librairies périphériques incompatible avec les anciennes librairies de Microchip (MLA).

Un grand avantage est le MHC (Microchip Harmony Configurator). Il s'agit d'un assistant permettant de générer un projet complet avec la mise en œuvre des périphériques choisis.

- ☺ L'utilisation du MHC allège passablement le travail et réduit l'étude des nombreuses fonctions de la nouvelle PLIB.
- ☺ L'intégration d'un BSP spécifique est possible depuis Harmony 1.06 et MPLAB X 3.x, ce qui simplifie encore la génération de projet.

## 1.5. LE PIC32MX795F512L EN BREF

### 1.5.1. SCHÉMA BLOC

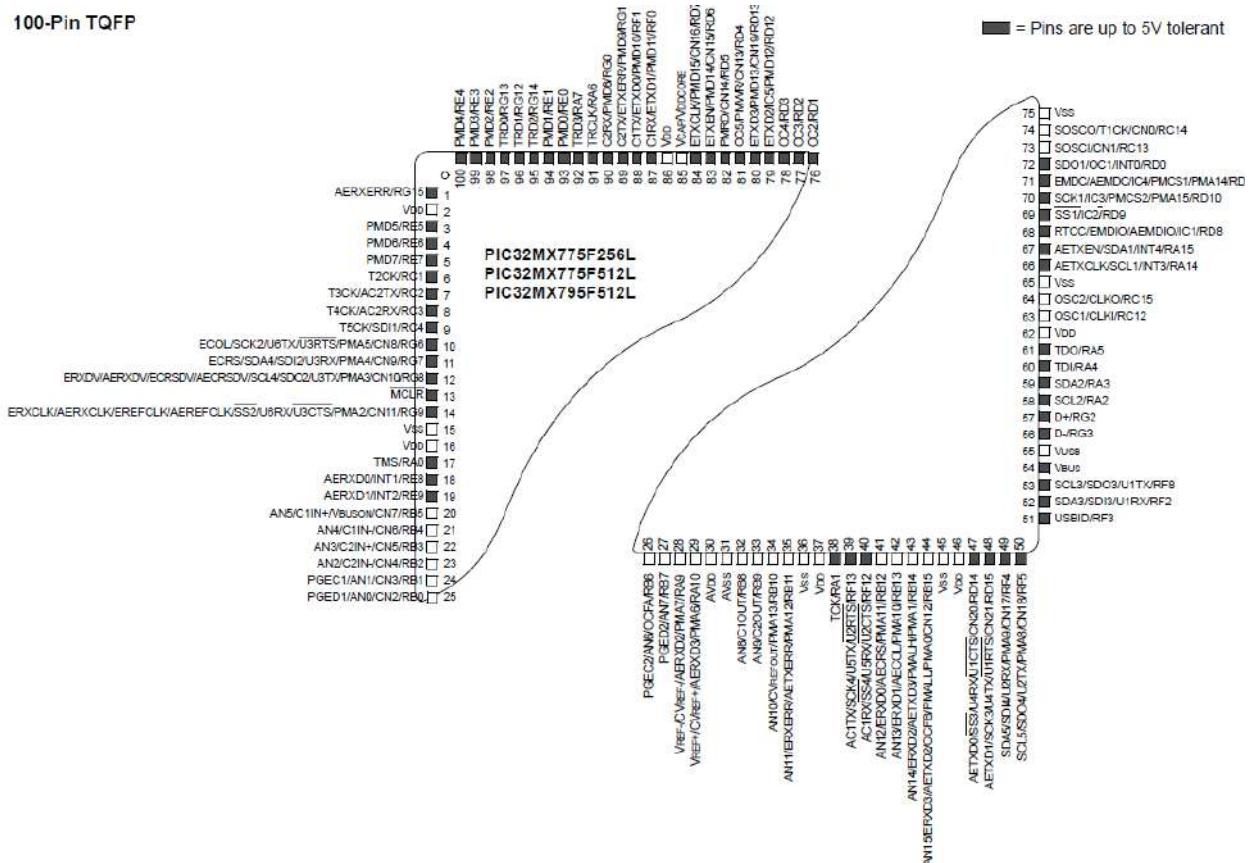


### 1.5.2. BOÎTIER ET BROCHAGE

Le boîtier choisi pour le kit est le TQFP de 100 broches. Le L signifie large.

100-Pin TQFP

■ = Pins are up to 5V tolerant

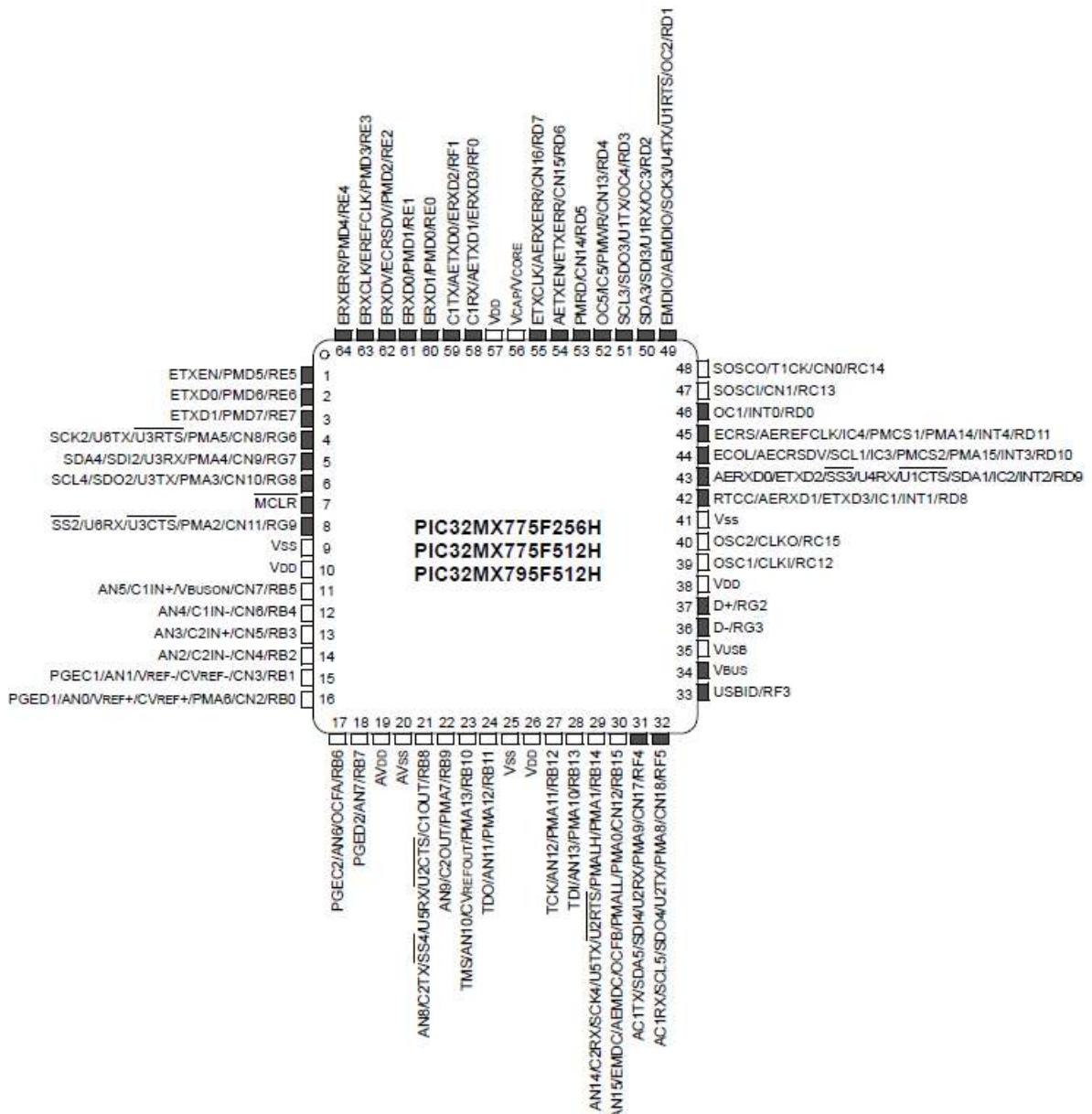


### 1.5.3. BOÎTIER ET BROCHAGE DE LA VERSION H

Le PIC32MX795F512H (H = half) dispose des mêmes fonctionnalités avec une réduction du nombre d'entrées-sorties à cause du boîtier TQFP de 64 broches.

64-Pin TQFP

■ = Pins are up to 5V tolerant



### 1.5.4. QUELQUES CARACTÉRISTIQUES



## PIC32MX5XX/6XX/7XX

### High-Performance, USB, CAN and Ethernet 32-bit Flash Microcontrollers

#### High-Performance 32-bit RISC CPU:

- MIPS32® M4K® 32-bit core with 5-stage pipeline
- 80 MHz maximum frequency
- 1.56 DMIPS/MHz (Dhrystone 2.1) performance at zero Wait state Flash access
- Single-cycle multiply and high-performance divide unit
- MIPS16e® mode for up to 40% smaller code size
- Two sets of 32 core register files (32-bit) to reduce interrupt latency
- Prefetch Cache module to speed execution from Flash

#### Microcontroller Features:

- Operating voltage range of 2.3V to 3.6V
- 64K to 512K Flash memory (plus an additional 12 KB of Boot Flash)
- 16K to 128K SRAM memory
- Pin-compatible with most PIC24/dsPIC® DSC devices
- Multiple power management modes
- Multiple interrupt vectors with individually programmable priority
- Fail-Safe Clock Monitor mode
- Configurable Watchdog Timer with on-chip Low-Power RC oscillator for reliable operation

#### Peripheral Features:

- Atomic SET, CLEAR and INVERT operation on select peripheral registers
- Up to 8-channels of hardware DMA with automatic data size detection
- USB 2.0-compliant full-speed device and On-The-Go (OTG) controller:
  - Dedicated DMA channels
- 10/100 Mbps Ethernet MAC with MII and RMII interface:
  - Dedicated DMA channels
- CAN module:
  - 2.0B Active with DeviceNet™ addressing support
  - Dedicated DMA channels
- 3 MHz to 25 MHz crystal oscillator

#### Peripheral Features (Continued):

- Internal 8 MHz and 32 kHz oscillators
- Six UART modules with:
  - RS-232, RS-485 and LIN support
  - IrDA® with on-chip hardware encoder and decoder
- Up to four SPI modules
- Up to five I<sup>2</sup>C™ modules
- Separate PLLs for CPU and USB clocks
- Parallel Master and Slave Port (PMP/PSP) with 8-bit and 16-bit data, and up to 16 address lines
- Hardware Real-Time Clock and Calendar (RTCC)
- Five 16-bit Timers/Counters (two 16-bit pairs combine to create two 32-bit timers)
- Five Capture inputs
- Five Compare/PWM outputs
- Five external interrupt pins
- High-speed I/O pins capable of toggling at up to 80 MHz
- High-current sink/source (18 mA/18 mA) on all I/O pins
- Configurable open-drain output on digital I/O pins

#### Debug Features:

- Two programming and debugging Interfaces:
  - 2-wire interface with unintrusive access and real-time data exchange with application
  - 4-wire MIPS® standard enhanced Joint Test Action Group (JTAG) interface
- Unintrusive hardware-based instruction trace
- IEEE Standard 1149.2 compatible (JTAG) boundary scan

#### Analog Features:

- Up to 16-channel, 10-bit Analog-to-Digital Converter:
  - 1 Msps conversion rate
  - Conversion available during Sleep and Idle
- Two Analog Comparators

### 1.5.5. USB, ETHERNET ET CAN SELON MODÈLE

TABLE 3: PIC32 USB, ETHERNET AND CAN – FEATURES

USB, Ethernet and CAN																		
Device	Pins	Program Memory (KB)		Data Memory (KB)	USB	Ethernet	CAN	Timers/Capture/Compare	DMA Channels (Programmable/Dedicated)	UART <sup>(2,3)</sup>	SPI <sup>(3)</sup>	I <sup>2</sup> C <sup>TM(3)</sup>	10-bit 1 Msps ADC (Channels)	Comparators	PMP/PSP	JTAG	Trace	Packages <sup>(4)</sup>
PIC32MX764F128H	64	128 + 12 <sup>(1)</sup>	32	1	1	1	5/5/5	4/6	6	3	4	16	2	Yes	Yes	No	PT, MR	
PIC32MX775F256H	64	256 + 12 <sup>(1)</sup>	64	1	1	2	5/5/5	8/8	6	3	4	16	2	Yes	Yes	No	PT, MR	
PIC32MX775F512H	64	512 + 12 <sup>(1)</sup>	64	1	1	2	5/5/5	8/8	6	3	4	16	2	Yes	Yes	No	PT, MR	
PIC32MX795F512H	64	512 + 12 <sup>(1)</sup>	128	1	1	2	5/5/5	8/8	6	3	4	16	2	Yes	Yes	No	PT, MR	
PIC32MX764F128L	100	128 + 12 <sup>(1)</sup>	32	1	1	1	5/5/5	4/6	6	4	5	16	2	Yes	Yes	Yes	PT, PF, BG	
PIC32MX775F256L	100	256 + 12 <sup>(1)</sup>	64	1	1	2	5/5/5	8/8	6	4	5	16	2	Yes	Yes	Yes	PT, PF, BG	
PIC32MX775F512L	100	512 + 12 <sup>(1)</sup>	64	1	1	2	5/5/5	8/8	6	4	5	16	2	Yes	Yes	Yes	PT, PF, BG	
PIC32MX795F512L	100	512 + 12 <sup>(1)</sup>	128	1	1	2	5/5/5	8/8	6	4	5	16	2	Yes	Yes	Yes	PT, PF, BG	

Legend: PF, PT = TQFP      MR = QFN      BG = XBGA

Note 1: This device features 12 KB boot Flash memory.

2: CTS and RTS pins may not be available for all UART modules. Refer to the “Pin Diagrams” section for more information.

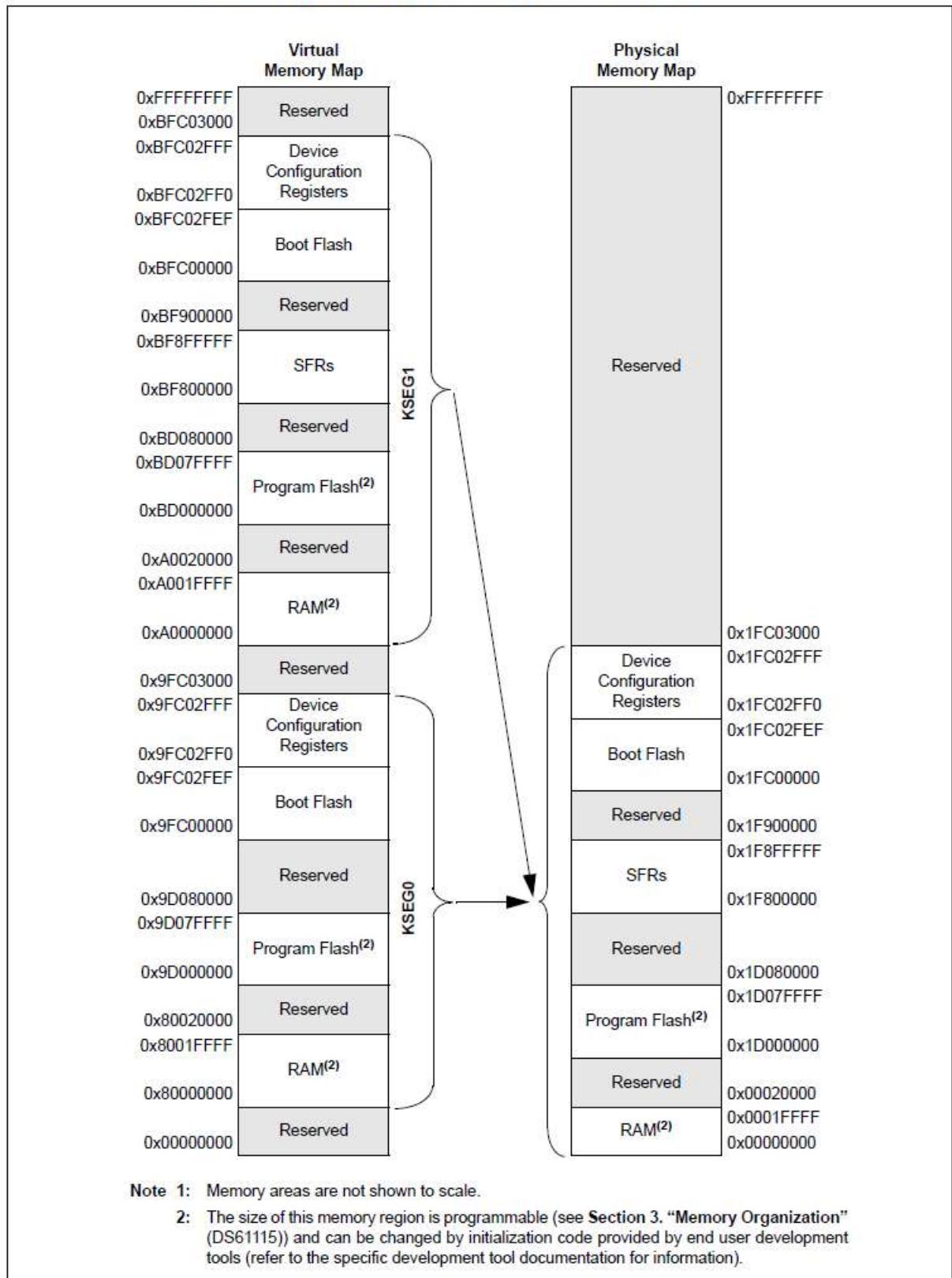
 3: Some pins between the UART, SPI and I<sup>2</sup>C modules may be shared. Refer to the “Pin Diagrams” section for more information.

4: Refer to 32.0 “Packaging Information” for more information.

☝ Le PIC32MX795F512H, le petit frère avec 64 broches, est un bon choix pour certains projets nécessitant l'Ethernet et/ou l'USB. Si ces fonctionnalités ne sont pas nécessaires, on pourra s'orienter vers une autre série aux fonctionnalités plus réduites.

### 1.5.6. ORGANISATION DE LA MÉMOIRE

FIGURE 4-6: MEMORY MAP ON RESET FOR PIC32MX695F512H, PIC32MX695F512L,  
PIC32MX795F512H AND PIC32MX795F512L DEVICES



## 1.6. DOCUMENTATIONS À DISPOSITION

### 1.6.1. DOCUMENTATION PIC32MX

Vous trouverez l'ensemble de la documentation des PIC32MX sur le réseau sous ...\\PROJETS\\SLO\\1102x\_SK32MX775F512L\\Datasheets\\PIC32 Family Reference Manual.

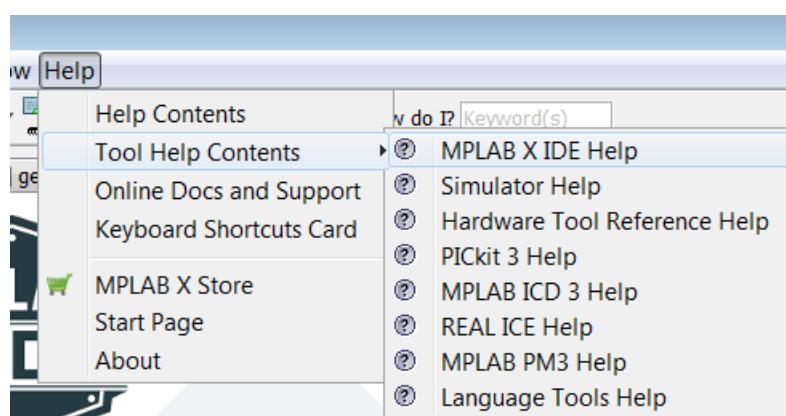
Remarque le projet 1102x est la première version du kit PIC32MX qui n'est plus utilisé.

### 1.6.2. DOCUMENTATION DU KIT PIC32MX

Vous trouverez le schéma du kit sur le réseau sous ...\\PROJETS\\SLO\\1102x\_SK32MX775F512L\\Hardware\\11020\_SK32MX775F512L

### 1.6.3. DOCUMENTATION COMPILATEUR ET ENVIRONNEMENT

Dans MPLAB X, l'aide est accessible à partir du menu :



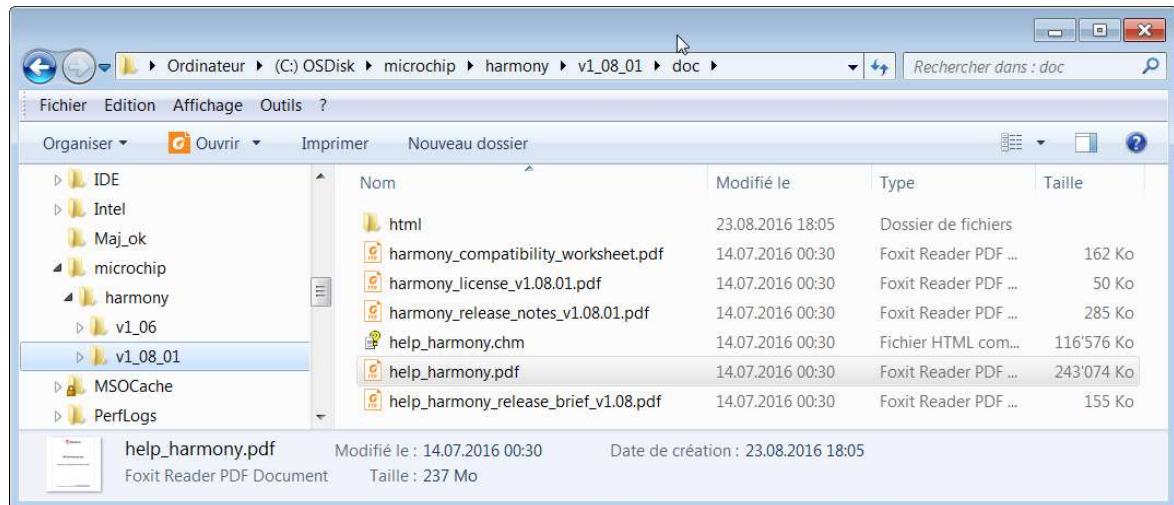
Pour le compilateur XC32, sa documentation se trouve sous :

<Répertoire xc32>\\v<n>\\docs.

En particulier dans le fichier MPLAB-XC32-Users-Guide.pdf.

### 1.6.4. DOCUMENTATION HARMONY

La documentation Harmony est présente dans le sous répertoire \doc de l'installation de Harmony.



### 1.6.5. DOCUMENTATION DANS L'ENVIRONNEMENT MPLABX

Dans l'ensemble de documentations accessible par le menu Help du MPLABX, on trouve la section **XC32 ToolChain** et la sous-section XC32 Peripheral Librairies qui correspond à l'ancienne PLIB.

Lorsque l'on utilise le MHC, une aide contextuelle est fournie automatiquement.

## 1.7. HISTORIQUE DES VERSIONS

### 1.7.1. V1.0 2013

Traite de MPLABX et XC32 sans Harmony (ancienne plib)

### 1.7.2. V1.5

Traite de MPLABX et XC32 avec Harmony (nouvelle plib)

### 1.7.3. V1.6

Traite de MPLABX 3.10 et XC32 avec Harmony 1.06

### 1.7.4. V1.7 OCTOBRE 2016

Traite de MPLABX 3.40 et XC32 avec Harmony 1.08

### 1.7.5. V1.8 NOVEMBRE 2017

Traite de MPLABX 3.61 et XC32 avec Harmony 1.11 (sauf indication contraire dans le document). Reprise et relecture par SCA

### 1.7.6. V1.81 NOVEMBRE 2021

Compléments choix du PIC32MX.

**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_skes	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 :

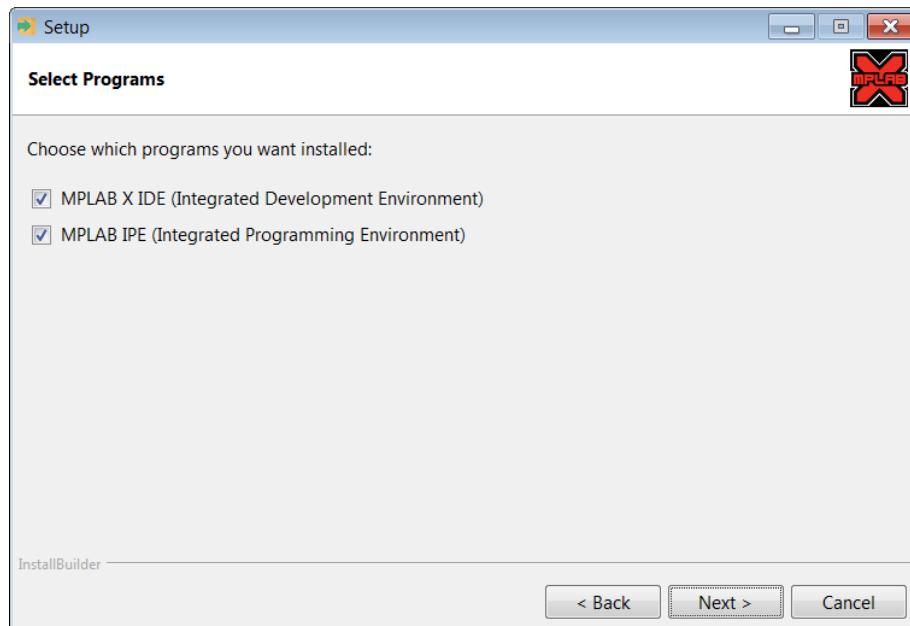
- MPLAB X 3.40
- XC32 1.42
- Harmony 1.08\_01

### 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 !

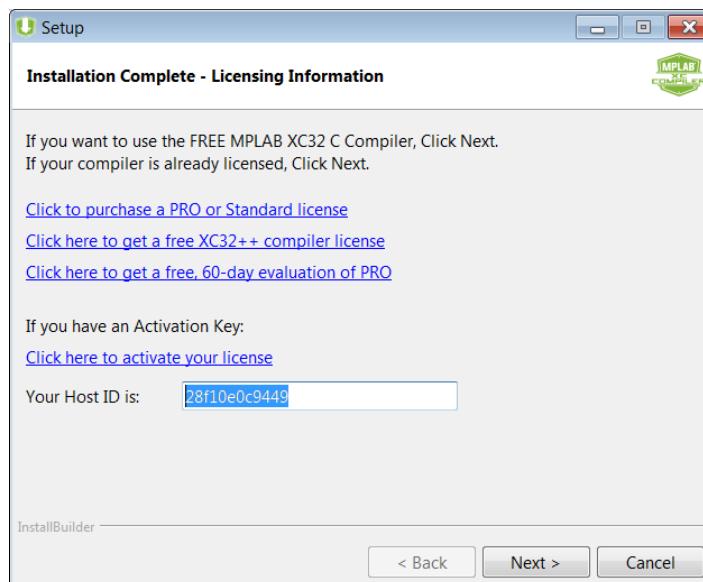


### 2.2.2. INSTALLATION DU COMPILATEUR XC32

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.

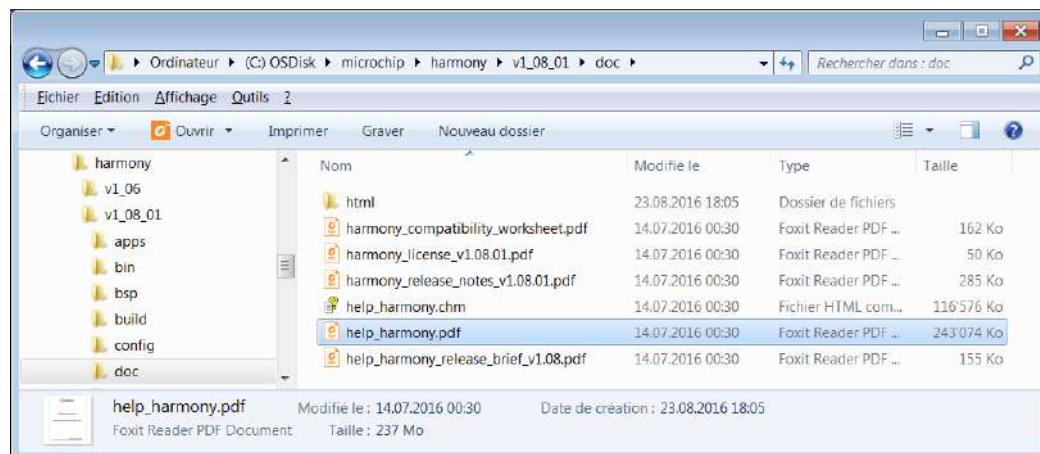


### 2.2.3. INSTALLATION DE HARMONY

Le fichier d'installation `harmony_v1_08_01_windows_installer.exe`, se trouve sous `...\\Maitres-Eleves\\Install\\PICS\\MPLAB\\Harmony\\v1_08_01`.

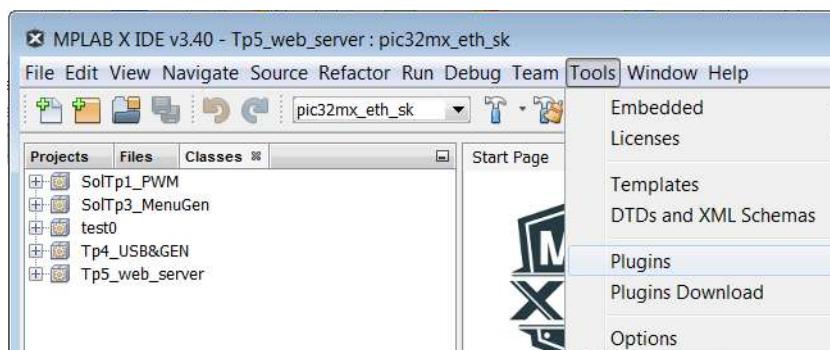
Réalisez l'installation avec les settings par défaut.

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

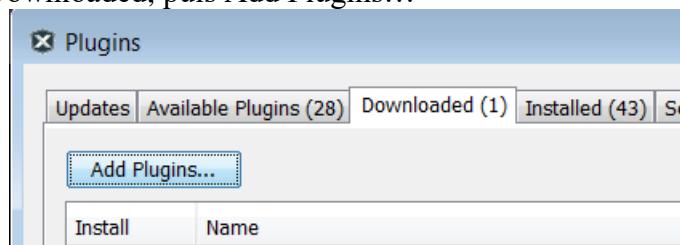


### 2.2.4. INTEGRATION DE HARMONY DANS MPLAB X

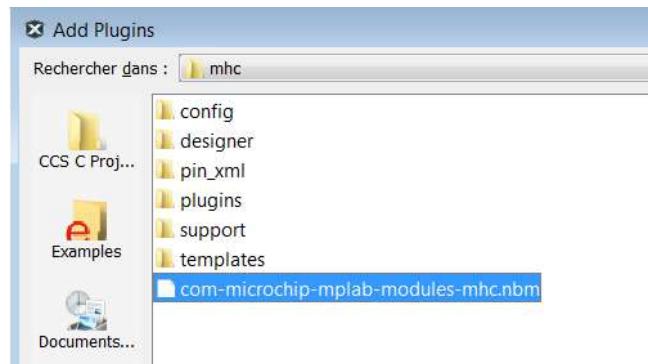
On lance MPLAB X, puis on ouvre l'onglet Tools > Plugins



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

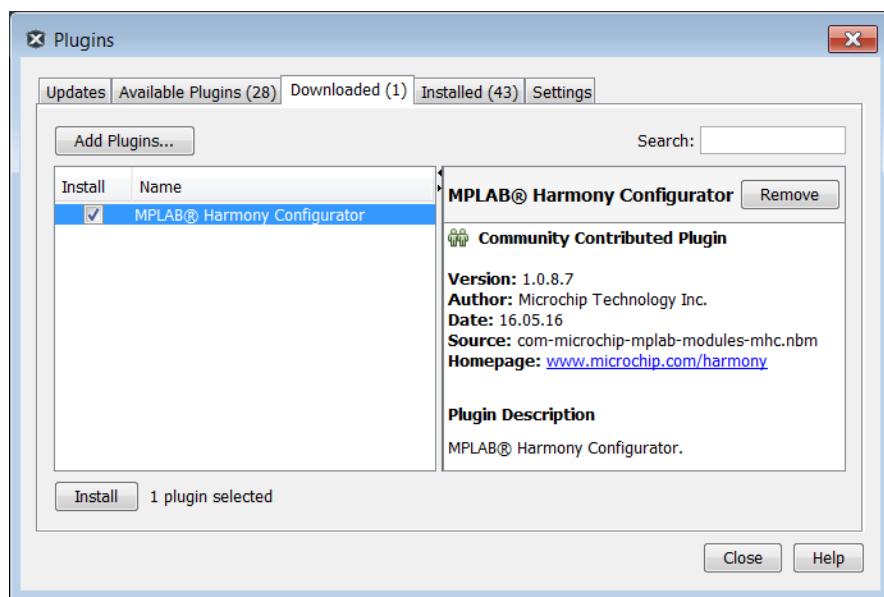


Et aller chercher le fichier ci-dessous :



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.

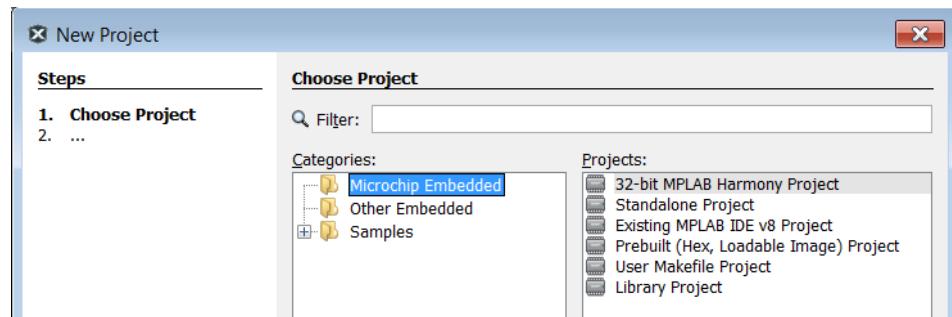


Il faut installer le plugin avec le bouton Install.

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

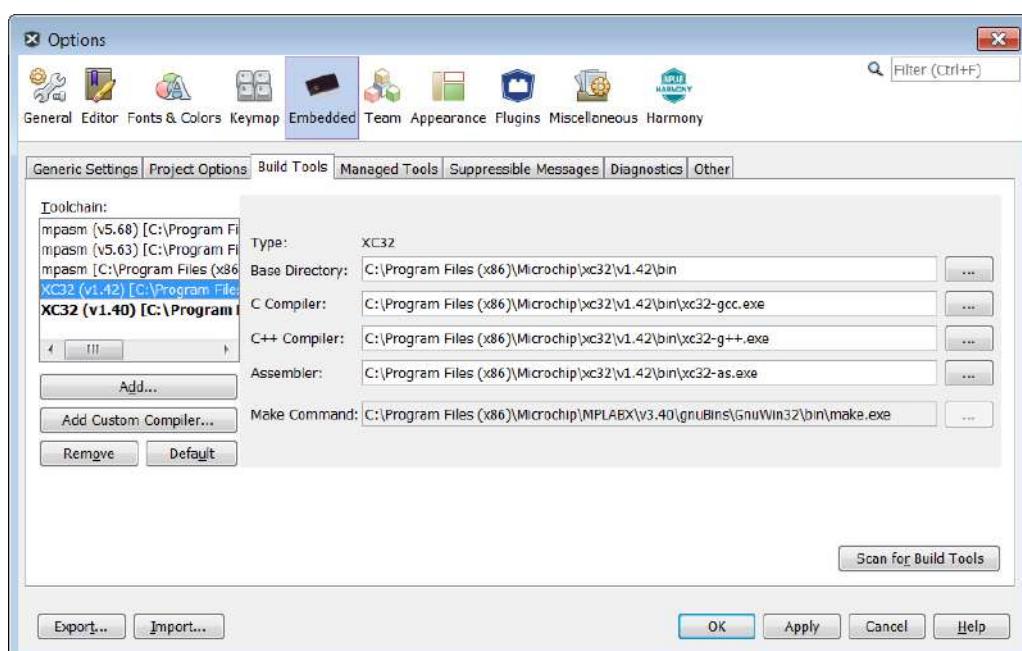


### 2.2.5. SELECTION VERSION DU XC32 DANS MPLAB X

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

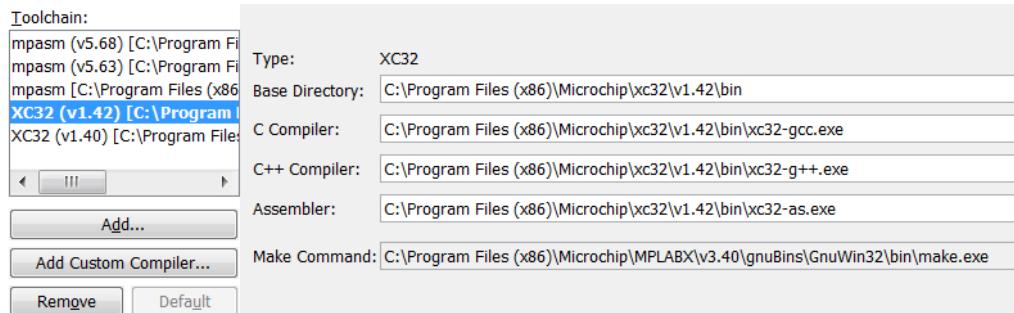
Ouvrir l'onglet Tools, sélectionnez Options. Dans la nouvelle fenêtre qui s'est ouverte sélectionnez Embedded.

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



### 2.2.5.1. ETABLISSEMENT V1.42 PAR DEFAUT

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



### 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.

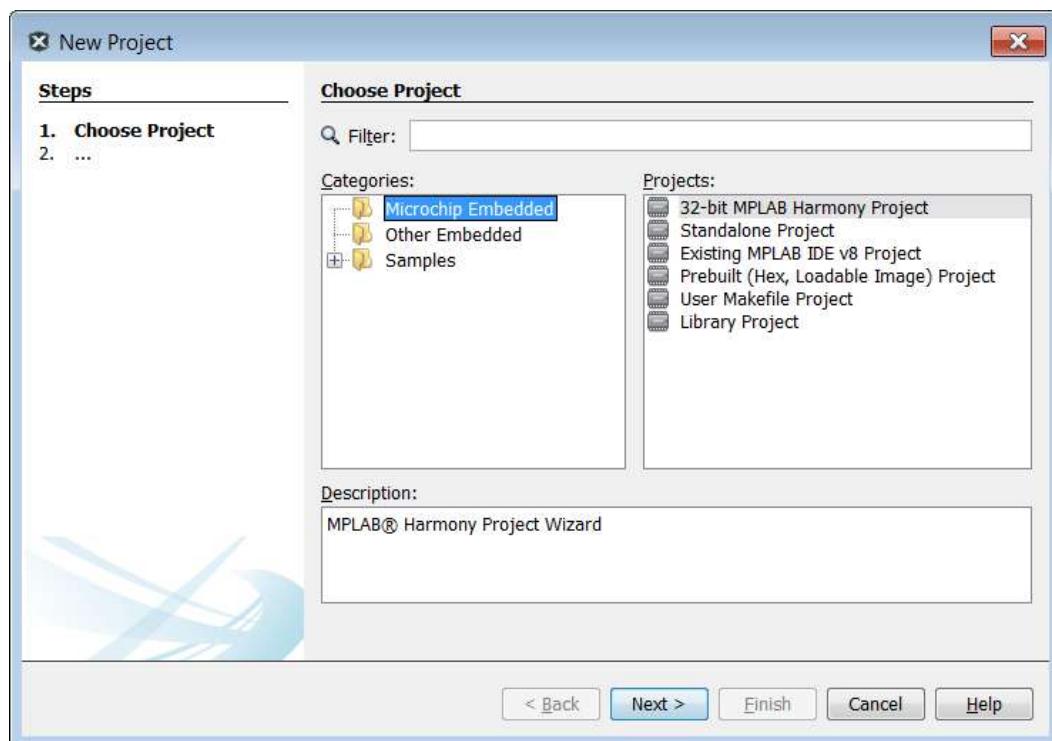
En plus de la création, le MHC permet la configuration graphique des fusibles, le choix d'un BSP et l'ajout de pilotes de périphériques.

Voici les étapes de la réalisation.

### 2.3.1. CRÉATION D'UN PROJET HARMONY

#### 2.3.1.1. CHOOSE PROJECT

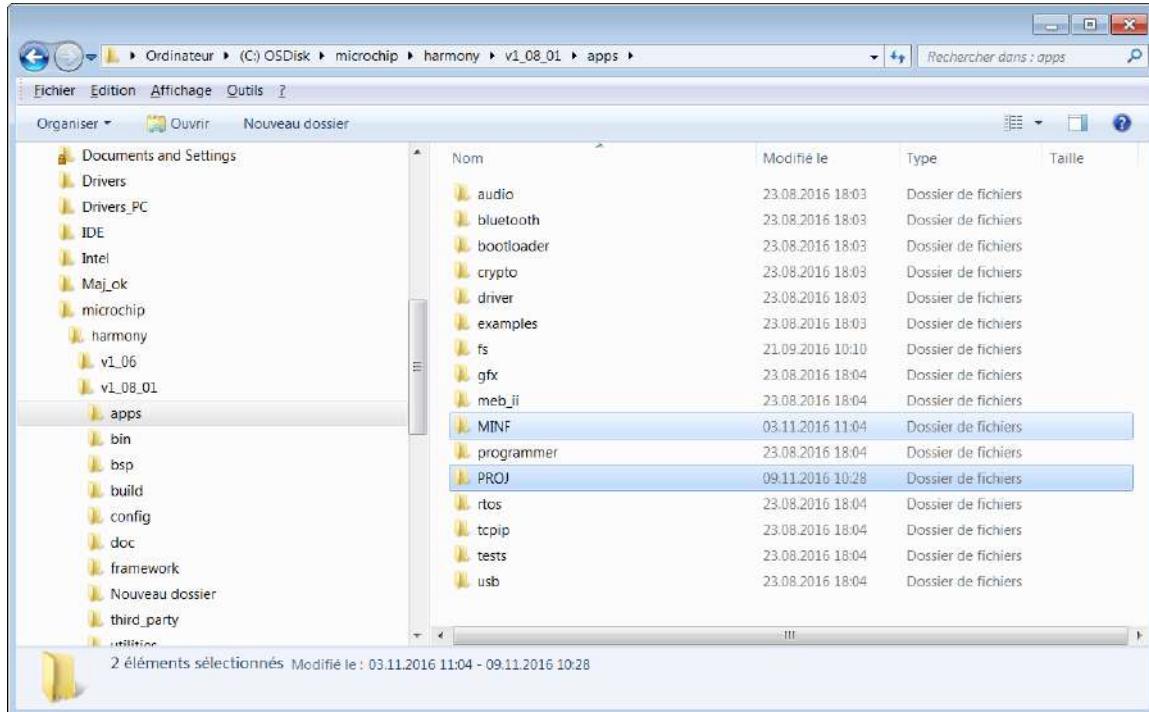
Il s'agit de créer un nouveau projet, mais au lieu de sélectionner un "Standalone Project", on va sélectionner 32 bit MPLAB Harmony Project.



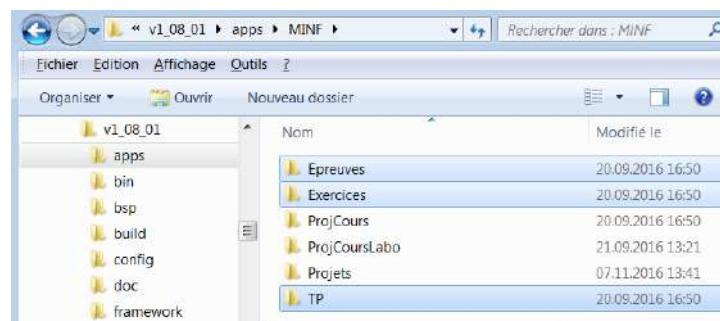
### 2.3.1.2. NAME AND LOCATION

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.

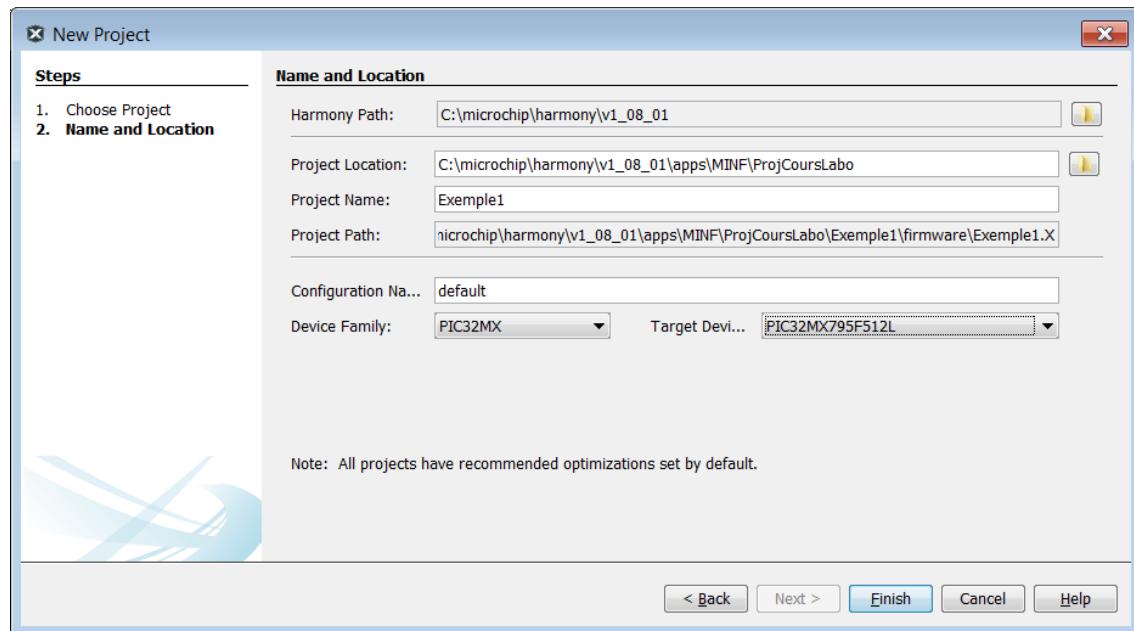
Pour éviter d'avoir trop de fichiers éparses sous apps, il est demandé de travailler avec les sous-répertoires MINF et PROJ, comme ci-dessous :



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



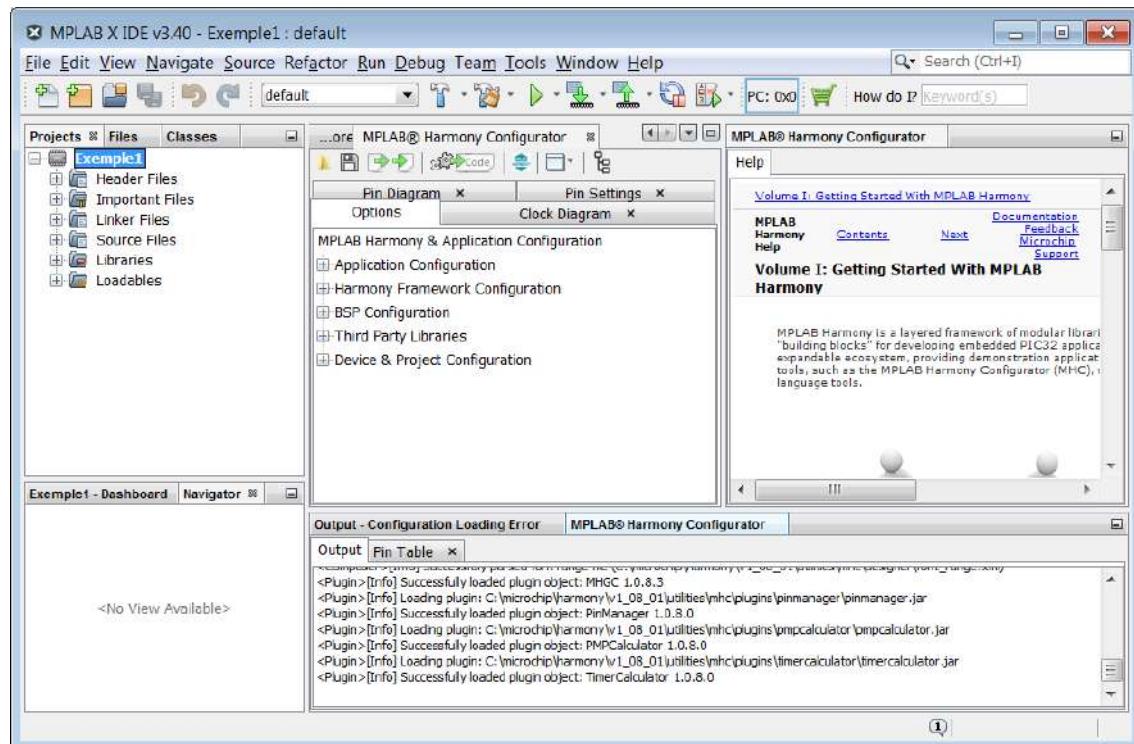
Pour les besoins de la réalisation des chapitres de cours pour le laboratoire, utilisation du répertoire apps\MINF\ProjCoursLabo. D'où :



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

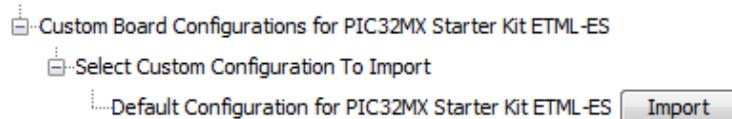
Il est important de commencer par la sélection du BSP, car cela impacte la section "Device & Project Configuration".

#### 2.3.2.1. BSP CONFIGURATION

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



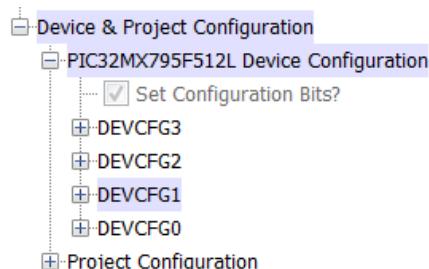
Au-dessous de la sélection du BSP, effectuer l'importation des réglages par défaut pour le kit :



#### 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 :

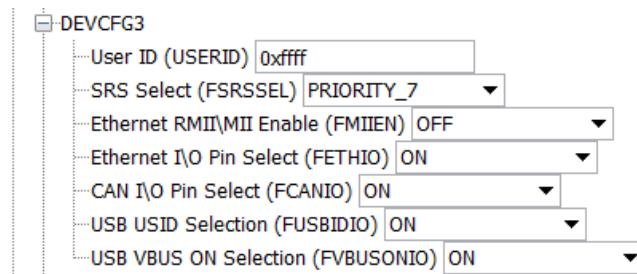


De par le choix du BSP du starter-kit ES, la partie "Device configuration" sera automatiquement configurée avec les valeurs par défaut pour le kit.

### 2.3.2.2.1. Section DEVCFG3

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

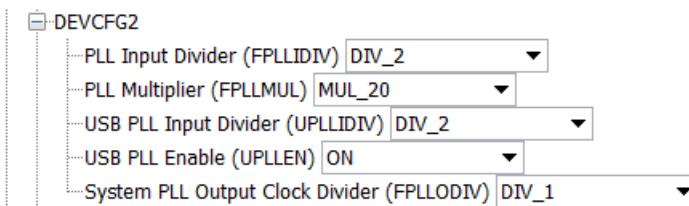
Il est possible de conserver la proposition par défaut. Si on le souhaite, on peut modifier les 2 config USB que l'on met à OFF.



### 2.3.2.2.2. Section DEVCFG2

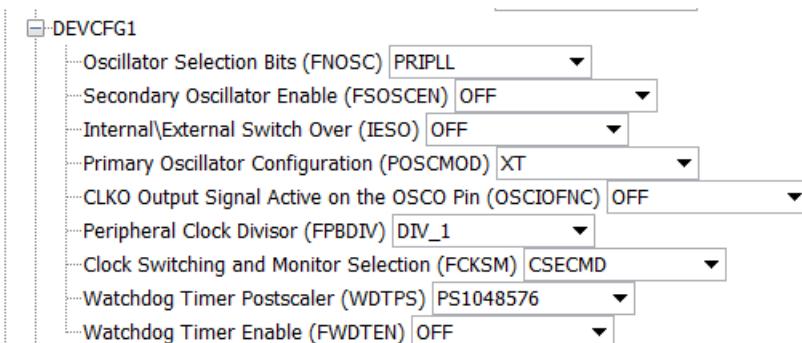
Cette section concerne la configuration des PLL.

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



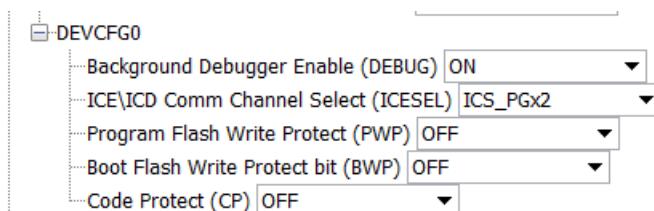
### 2.3.2.2.3. Section DEVCFG1

Cette section concerne la configuration des oscillateurs.



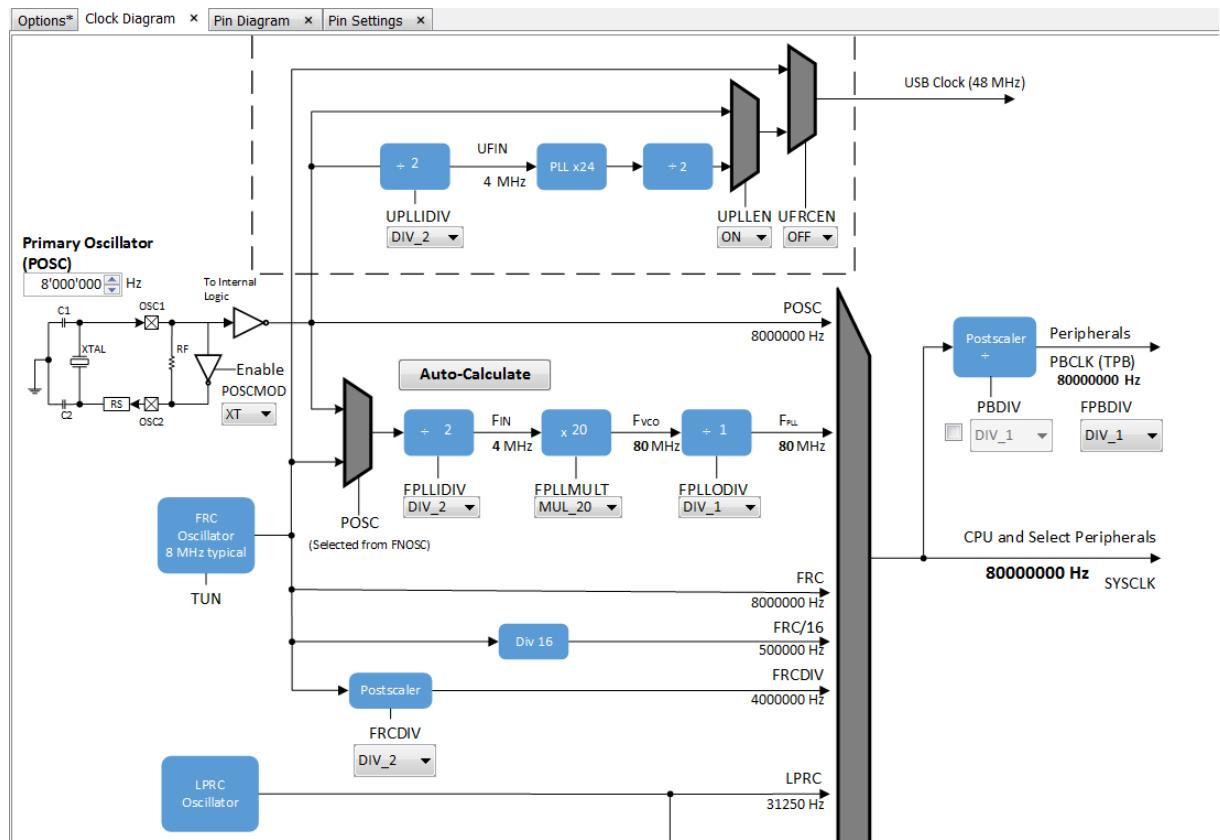
### 2.3.2.2.4. Section DEVCFG0

Cette section concerne le debugger et la mémoire programme.



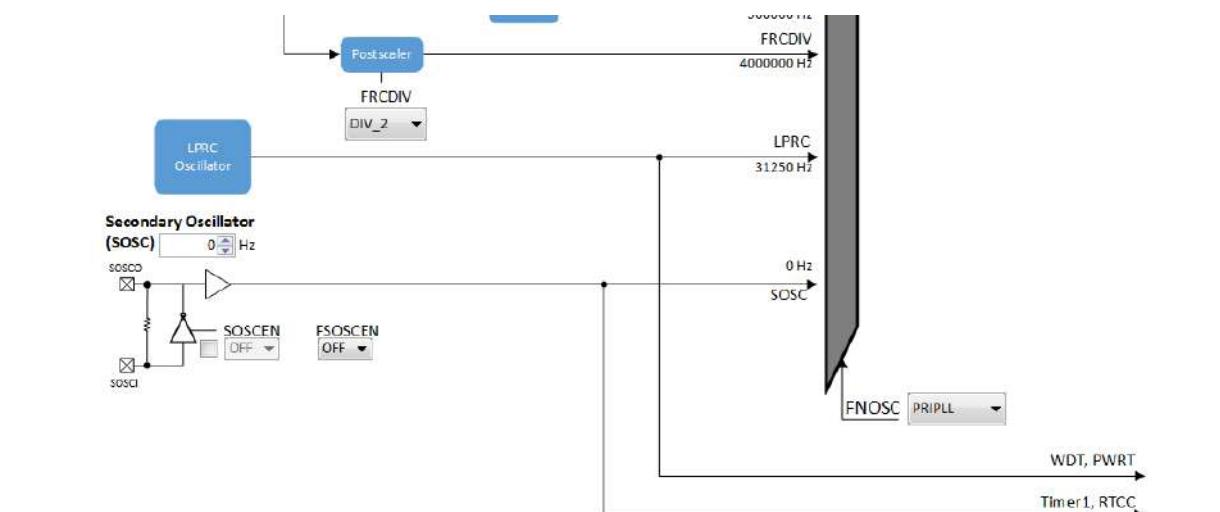
### 2.3.2.3. L'AIDE DU CLOCK DIAGRAM

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 !

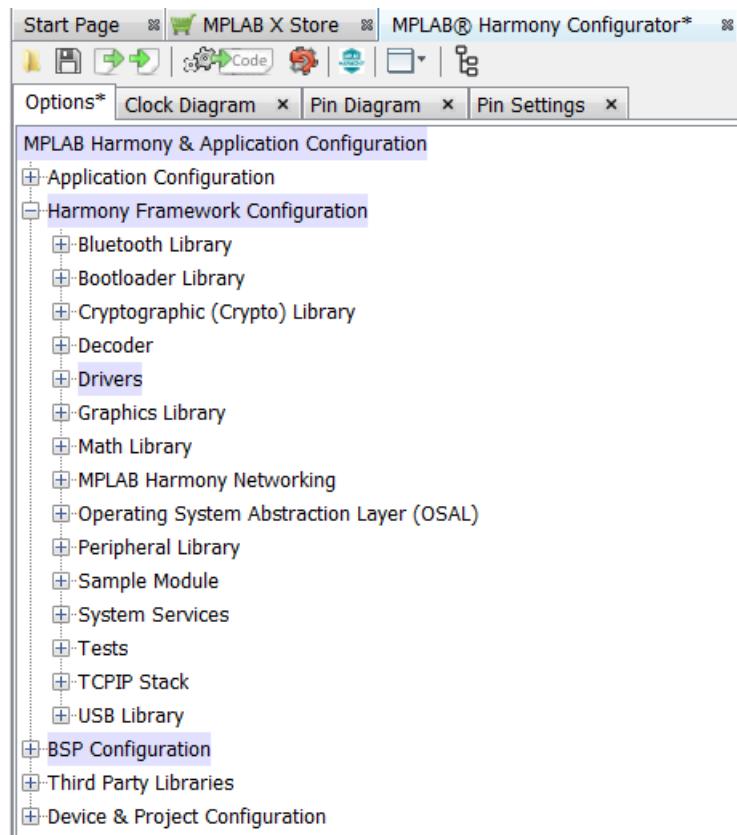
Avec un quartz à 8 MHz comme c'est le cas sur le starter-kit ES, il faut **POSCMOD = XT**. Avec le **DIV\_2** puis le **MUL\_20** on obtient au final 80 MHz pour **SYSCLK**, ce qui est le maximum pour le modèle de PIC32 du kit. Il faut encore **FNOSC en PRIPLL**.



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

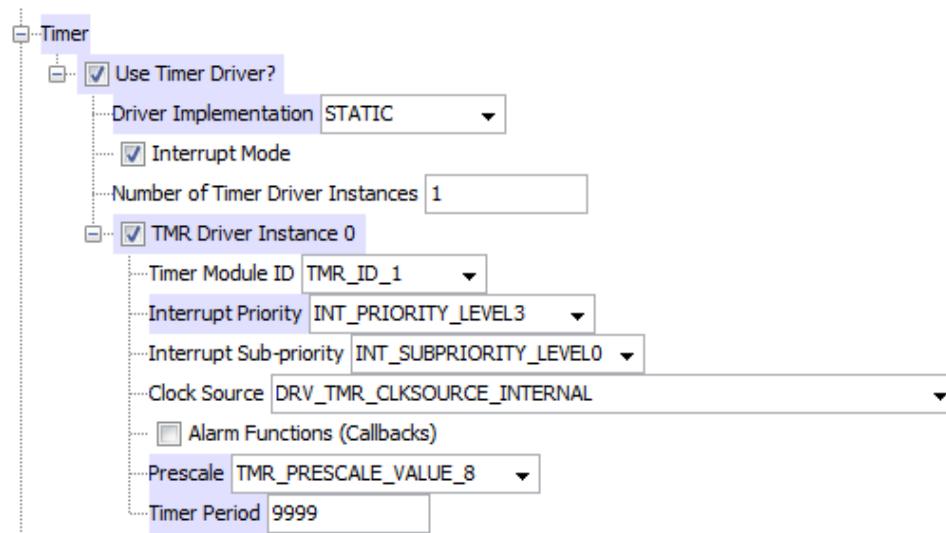
### 2.3.2.4. HARMONY FRAMEWORK CONFIGURATION

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

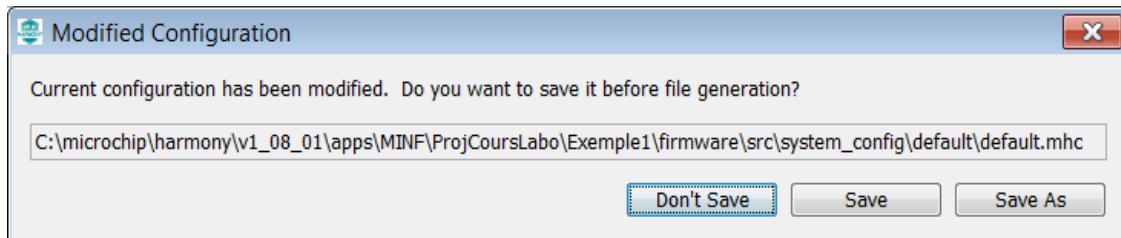
Elle est imposée par le choix du PIC.



### 2.3.2.5. GENERATION ET SAUVEGARDE DE LA CONFIGURATION

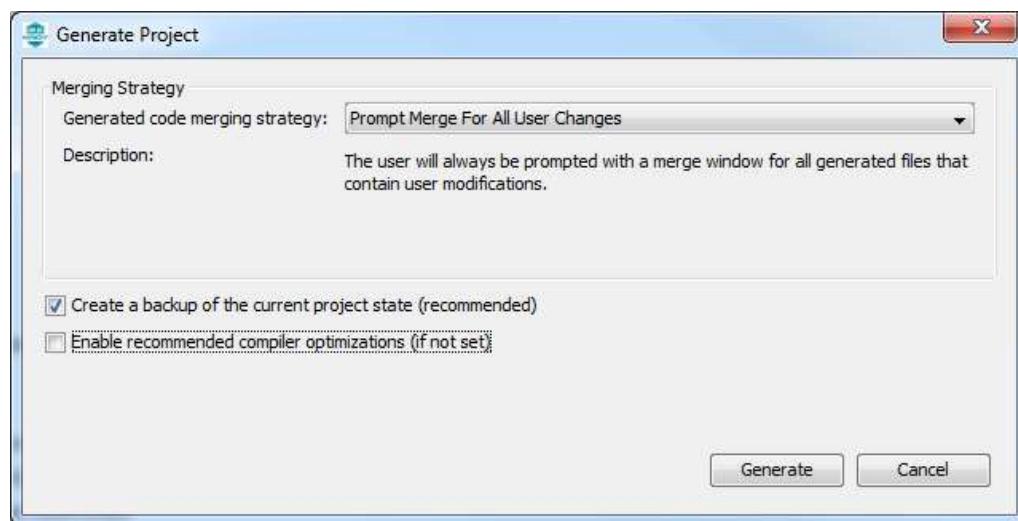
Génération avec le bouton Generate...

La demande de sauvegarde s'affiche, répondre Save.



Au menu suivant :

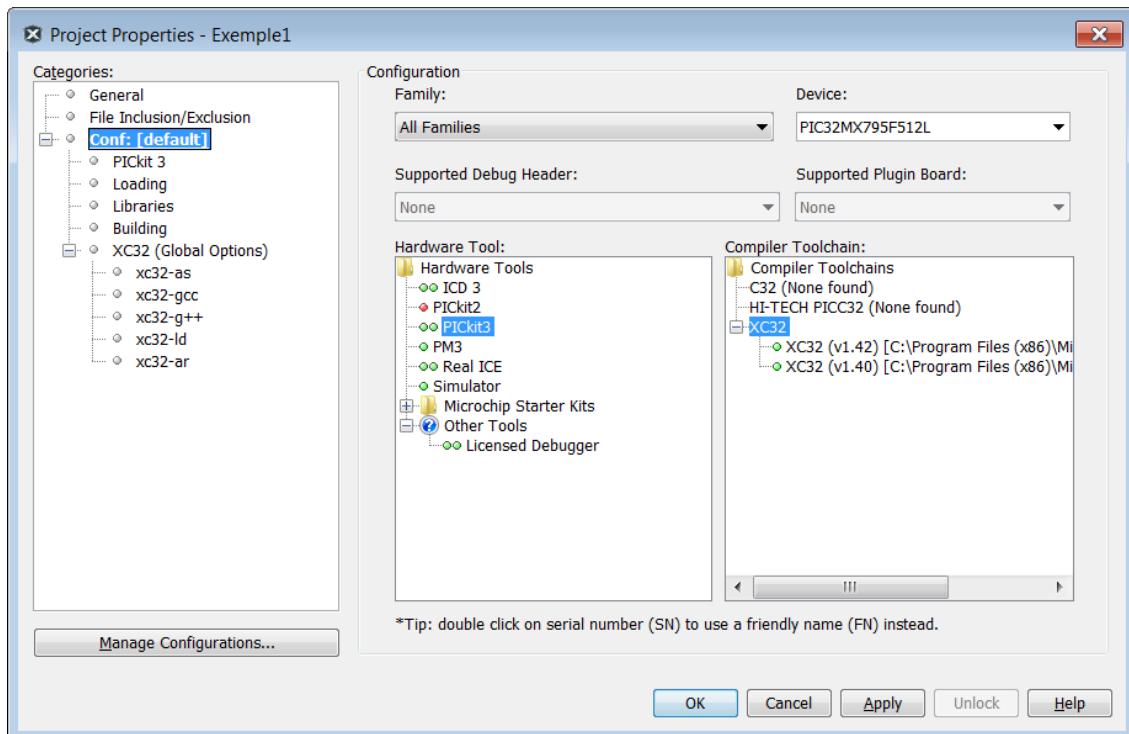
- Enlever la coche "Enable recommended compiler optimizations (if not set)", sans quoi l'optimisation de la compilation sera remise à 1 à chaque génération de code par le MHC.
- Laisser le reste par défaut



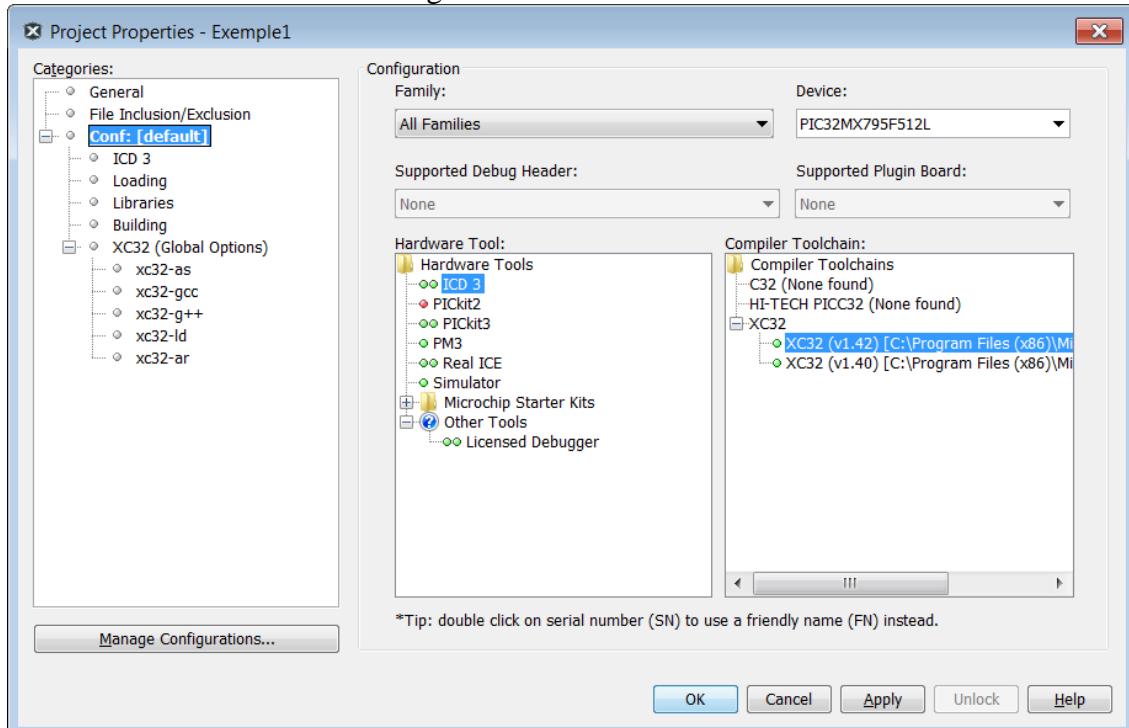
Avec ce choix, une demande de fusion apparaitra pour chaque modification.

### 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.



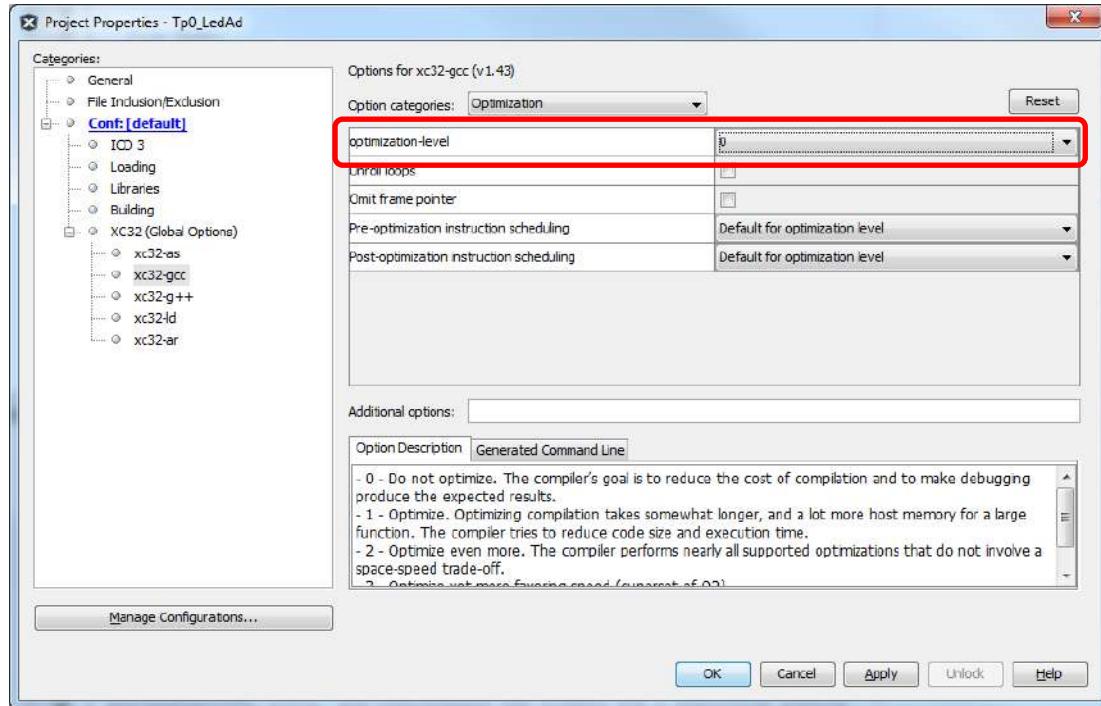
Sélectionner le bon outil de debug :



### 2.3.3.1. NIVEAU D'OPTIMISATION

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

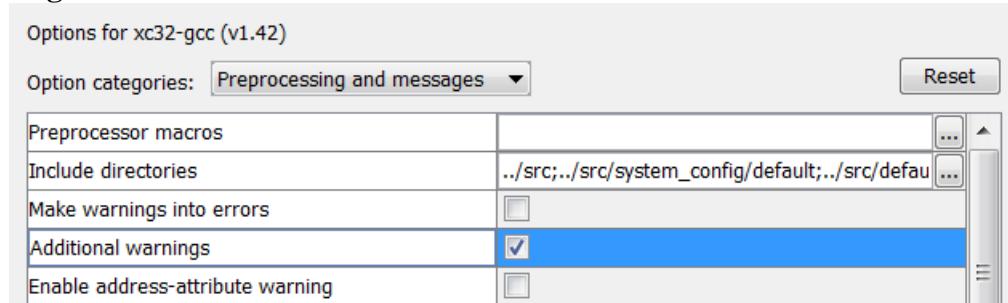
Pour faciliter le debugging il faut **régler le niveau d'optimisation à "0 - Do not optimize"** :



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

### 2.3.3.2. AVERTISSEMENTS A LA COMPILEATION

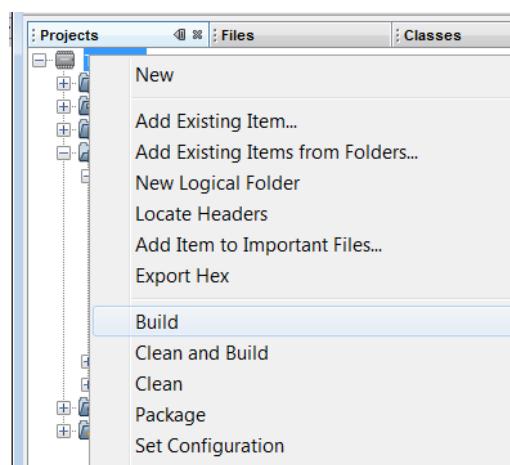
Sous xc32-gcc > preprocessing and messages, **ajouter la coche "Additional warnings"** :



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.

### 2.3.4. BUILD DU PROJET

Le **Build** ou le **Clean and Build** peuvent être exécutés à partir des icônes ou par le menu déroulant obtenu à partir d'un clic droit sur le projet.



On observe le résultat du Build dans la fenêtre de sortie (Output) :

```

Configuration Loading Error Exemple1 (Clean, Build, ...)

"C:\Program Files (x86)\Microchip\xc32\v1.42\bin\xc32-gcc.exe" -g -x c -c -mprocessor=32MX795F512L -ffunction-sect
"C:\Program Files (x86)\Microchip\xc32\v1.42\bin\xc32-gcc.exe" -g -x c -c -mprocessor=32MX795F512L -ffunction-sect
"C:\Program Files (x86)\Microchip\xc32\v1.42\bin\xc32-gcc.exe" -g -x c -c -mprocessor=32MX795F512L -ffunction-sect
"C:\Program Files (x86)\Microchip\xc32\v1.42\bin\xc32-gcc.exe" -g -x c -c -mprocessor=32MX795F512L -ffunction-sect
"C:\Program Files (x86)\Microchip\xc32\v1.42\bin\xc32-gcc.exe" -g -x c -c -mprocessor=32MX795F512L -dist/default/product
"C:\Program Files (x86)\Microchip\xc32\v1.42\bin\xc32-bin2hex dist/default/production/Exemple1.X.production.elf
make[2]: Leaving directory 'C:/microchip/harmony/v1_08_01/apps/MINF/ProjCoursLabo/Exemple1/firmware/Exemple1.X'
make[1]: Leaving directory 'C:/microchip/harmony/v1_08_01/apps/MINF/ProjCoursLabo/Exemple1/firmware/Exemple1.X'

BUILD SUCCESSFUL (total time: 5s)
Loading code from C:/microchip/harmony/v1_08_01/apps/MINF/ProjCoursLabo/Exemple1/firmware/Exemple1.X/dist/default/
Loading completed

```

## 2.4. ADAPTATION DU CANEVAS

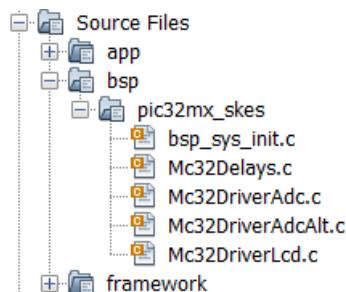
Voici quelques étapes nécessaires avant d'obtenir une application capable d'exécuter une action sur le kit PIC32MX de l'ES. Maintenant avec l'obtention du BSP pic32mx\_skes, il s'agit seulement de vérifier si tout est correct dans l'arborescence du projet.

### 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.

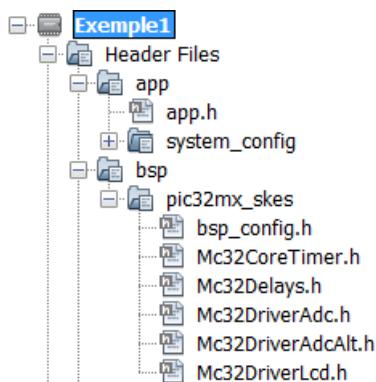
#### 2.4.1.1. LOCALISATION DU BSP DANS LES SOURCE FILES

Il faut vérifier la présence du répertoire BSP et du sous répertoire pic32mx\_skes, qui doit contenir les fichiers comme ci-dessous :



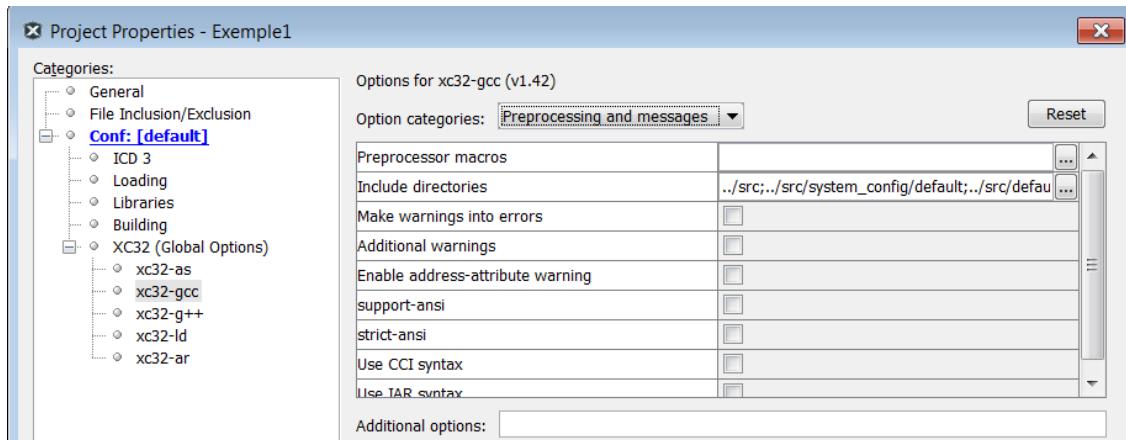
#### 2.4.1.2. LOCALISATION DU BSP DANS HEADER FILES

Il faut vérifier la présence du répertoire BSP et du sous répertoire pic32mx\_skes, qui doit contenir les fichiers comme ci-dessous :

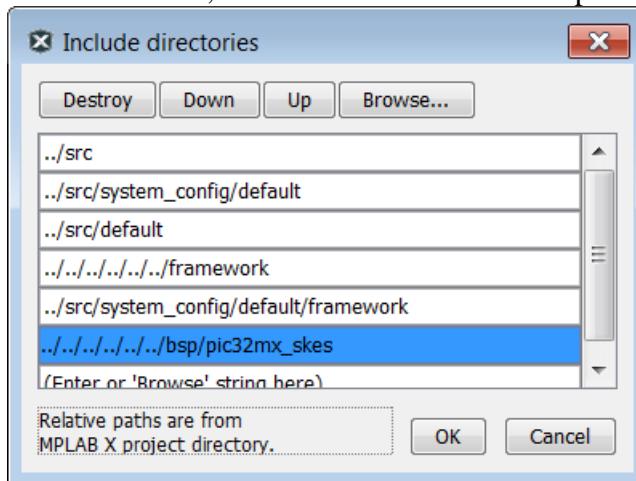


#### 2.4.1.3. VERIFICATION DU CHEMIN INCLUDE

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

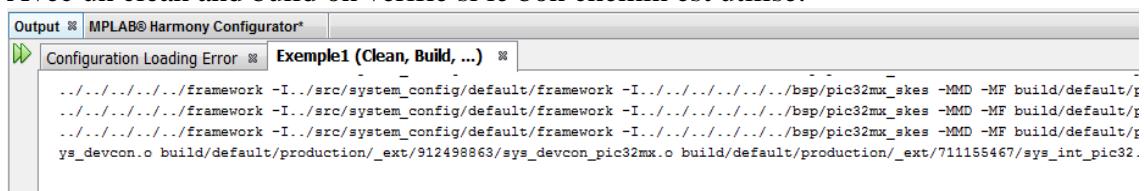


Puis en ouvrant Include directories, on doit avoir le chemin sur pic32mx\_skes :



#### 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 */

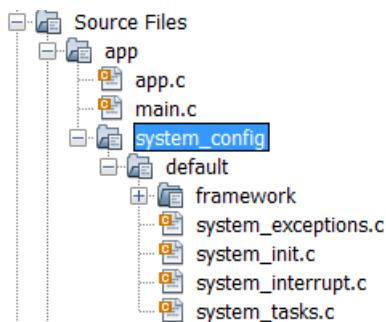
    return ( EXIT_FAILURE );
}

```

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 TMRO */
    DRV_TMRO_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 -ffunctions
"C:\Program Files (x86)\Microchip\xc32\v1.42\bin\xc32-gcc.exe" -g -x c -c -mprocessor=32MX795F512L -ffunctions
"C:\Program Files (x86)\Microchip\xc32\v1.42\bin\xc32-gcc.exe" -g -x c -c -mprocessor=32MX795F512L -ffunctions
../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_b1_on' [-Wimplicit-function-declaration]
    lcd_b1_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. EMPLACEMENT 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

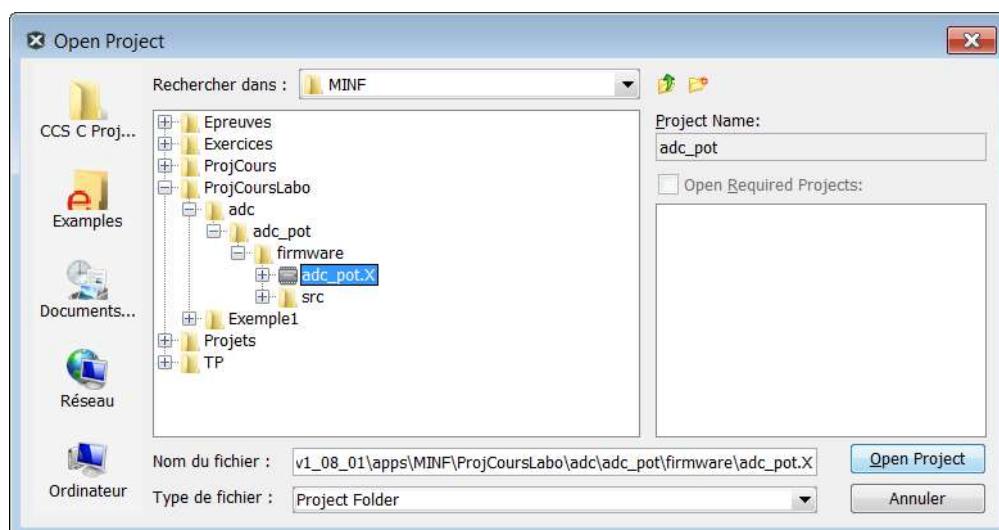
Pour valider la solution du déplacement, on copie l'exemple sous <Répertoire Harmony>\v<n>\apps\MINF\ProjCoursLabo\adc,

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

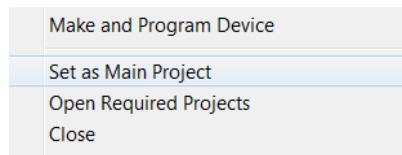
<Répertoire Harmony>\v<n>\apps\examples\peripheral\adc

### 2.6.3. OUVERTURE DU PROJET

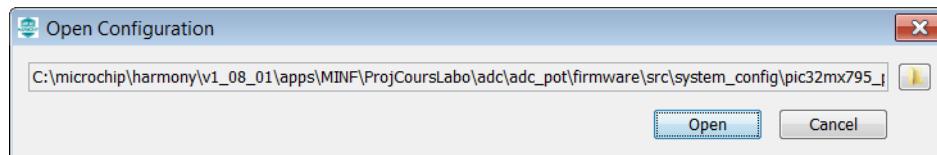
Ouverture du projet avec File, Open Project



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

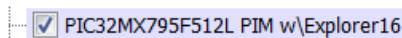


Cela a pour effet de lancer le  MPLAB® Harmony Configurator et de demander l'ouverture de la configuration.

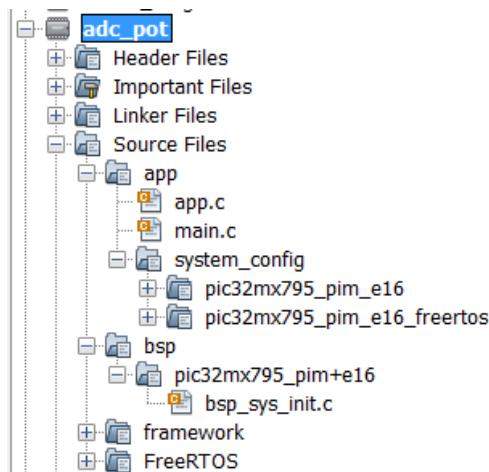


## 2.6.4. SITUATION DE L'EXEMPLE ADC\_POT

Au niveau du BSP celui qui est sélectionné est :



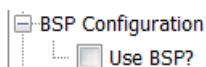
Au niveau du projet, on observe :



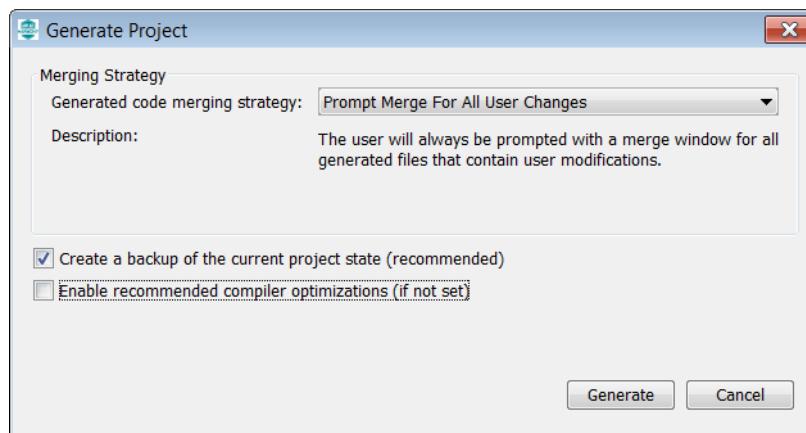
On dispose de 2 configurations, dont une avec FreeRTOS.

## 2.6.5. REMPLACEMENT DU BSP

### 2.6.5.1. SUPPRESSION DU BSP

Pour changer de BSP, il faut décocher Use BSP 

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 fichier (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" :

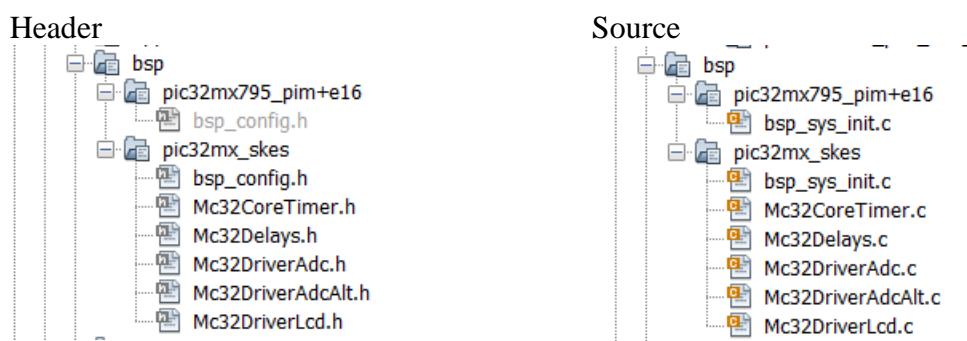


Ensuite on génère à nouveau le code.

### 2.6.6. VERIFICATION DU REMplacement DU BSP

#### 2.6.6.1. SITUATION ARBORESCENCE BSP

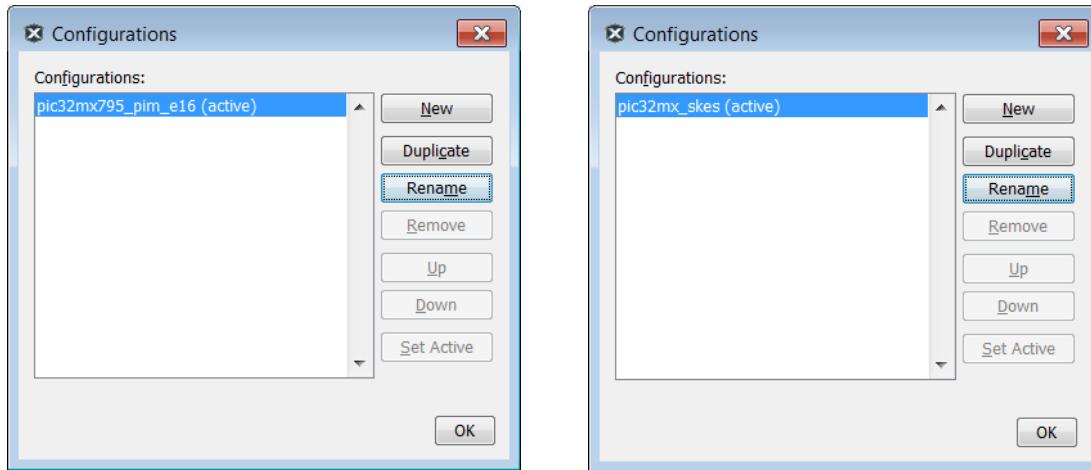
On vérifie que l'on a bien notre BSP et les fichiers associés.



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

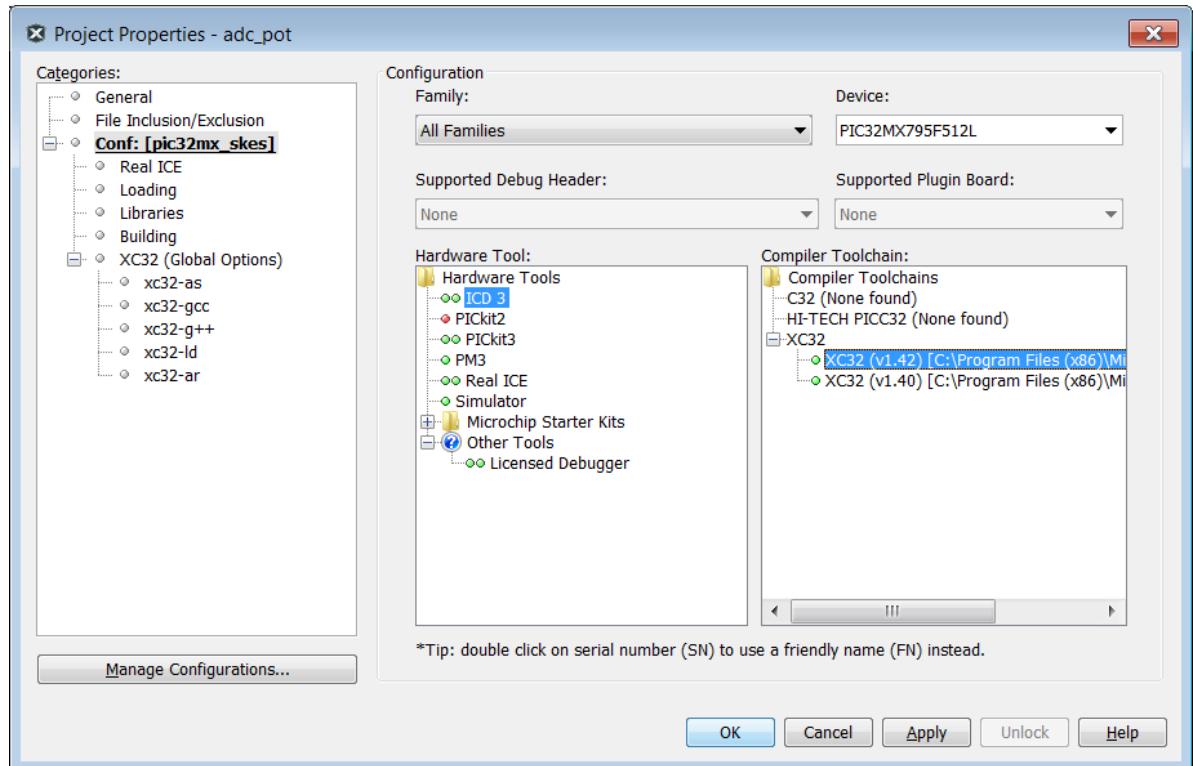
## 2.6.7. MISE EN ORDRE DE LA CONFIGURATION

Il est possible en utilisant [Manage Configurations...](#) 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

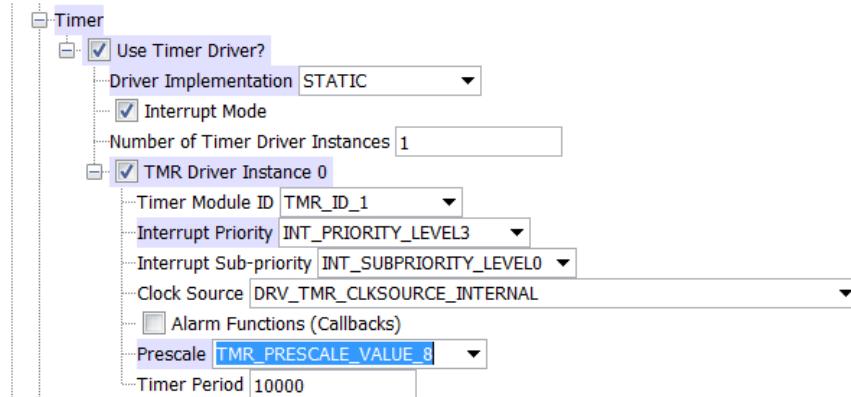
En utilisant "Remove from project", on supprime le répertoire de l'autre BSP et de la configuration avec FreeRTOS.

## 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 :

```

Merging: system_interrupt.c
Pending Merge Actions: 2
Generated Code
Note: The content of the right pane displays the current state of this file.

// ****
// ***** Section: System Interrupt Vector Functions *****
// ****
// ****
// ****
void __ISR(_ADC_VECTOR, ipl3AUTO) _IntHandlerDrvAdc(void)
{
    /* Clear ADC Interrupt Flag */
    PLIB_INT_SourceFlagClear(INT_ID_C);
}

void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
{
    PLIB_INT_SourceFlagClear(INT_ID_C);
}
// **** End of File ****

```

### 2.6.8.3. TEST DE COMPILEMENT

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/projcourslabo/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;
```

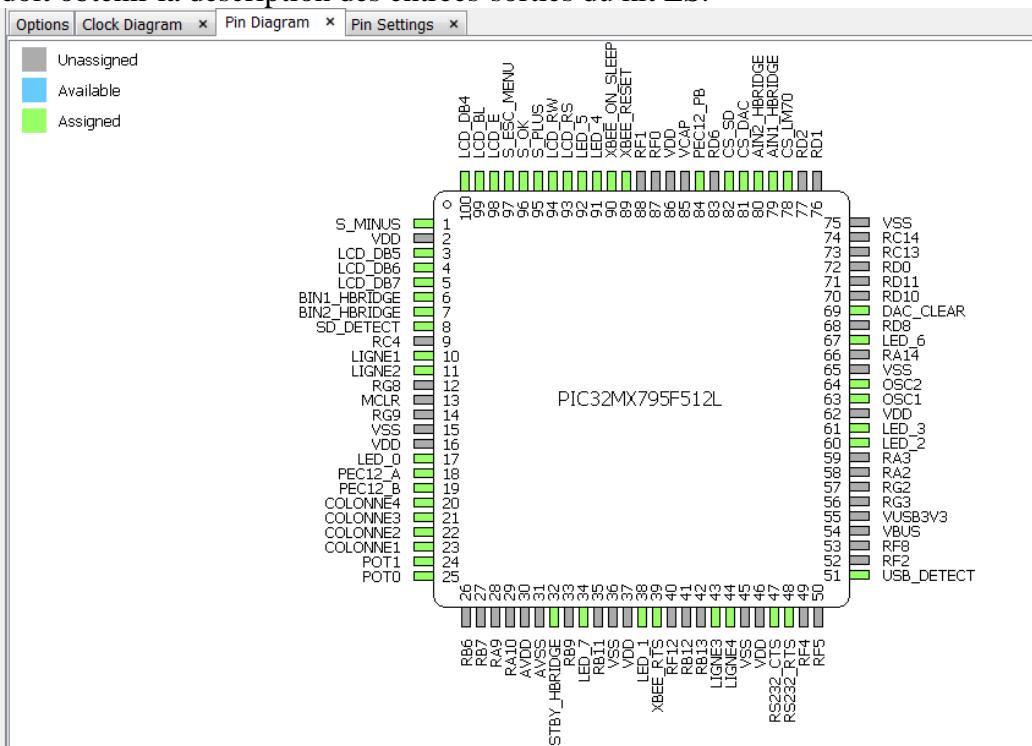
Et on obtient une compilation sans erreur ni warnings.

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

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

### 2.6.8.4. VERIFICATION PIN CONFIGURATION

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



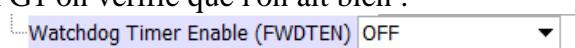
On contrôle encore au niveau "Pin Settings" que les sorties sont bien marquées out.

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

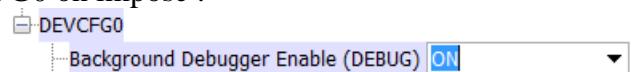
### 2.6.8.5. CONTROLE DEVICE CONFIGURATION

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 :



À niveau DEVCFG0 on impose :



⌚ 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 );
    sysObi.sysDevcon = SYS_DEVCON_Initialize(SYS_DEVCON_INDEX_0, (SYS_DEVCON_Index_Type)0);
    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       0xffff
#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, ipl3AUTO)
           _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

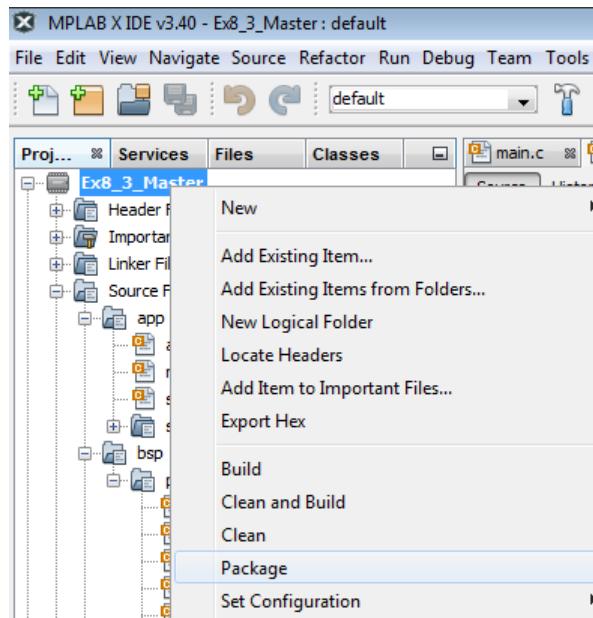
Il s'agit ici de définir une méthode de travail pour sauvegarder un projet Harmony afin d'y inclure tout le nécessaire à la reprise d'un projet.

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

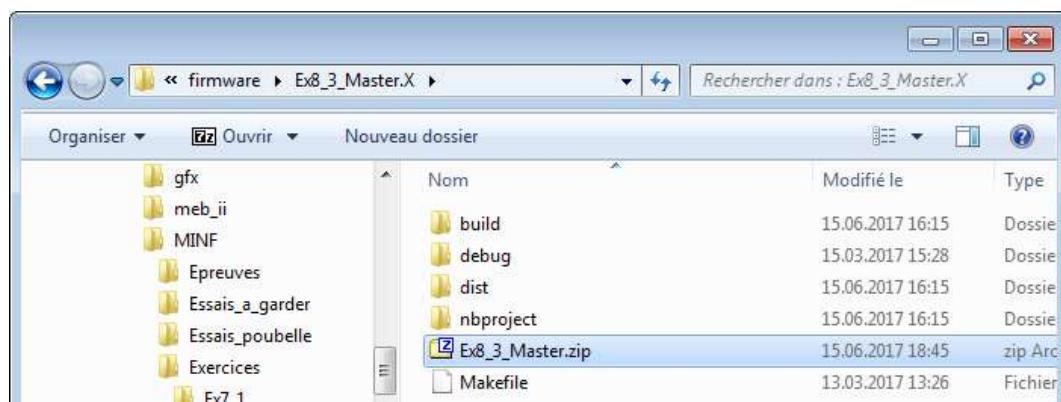
#### 2.7.1.1. FONCTION DE PACKAGE

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 :

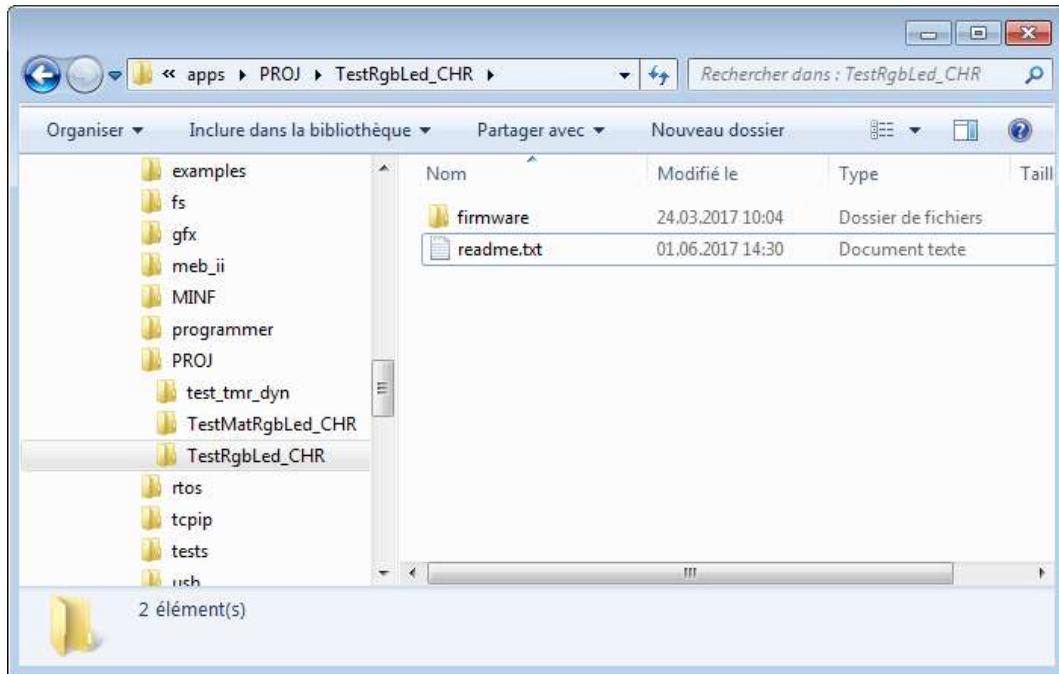


Sauvegarder uniquement les fichiers de code source ne suffit pas. Il manque par exemple le fichier de configuration du MHC et les fichiers web (dans le cas d'un projet serveur web).

### 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\_CmdBTDDirigeable
- 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\_sk****e**s) se trouve sous :  
...\\Maitres-Eleves\\SLO\\Modules\\SL229\_MINF\\Harmony\_ES\\v<n>

Mise à jour de la liste des BSP :

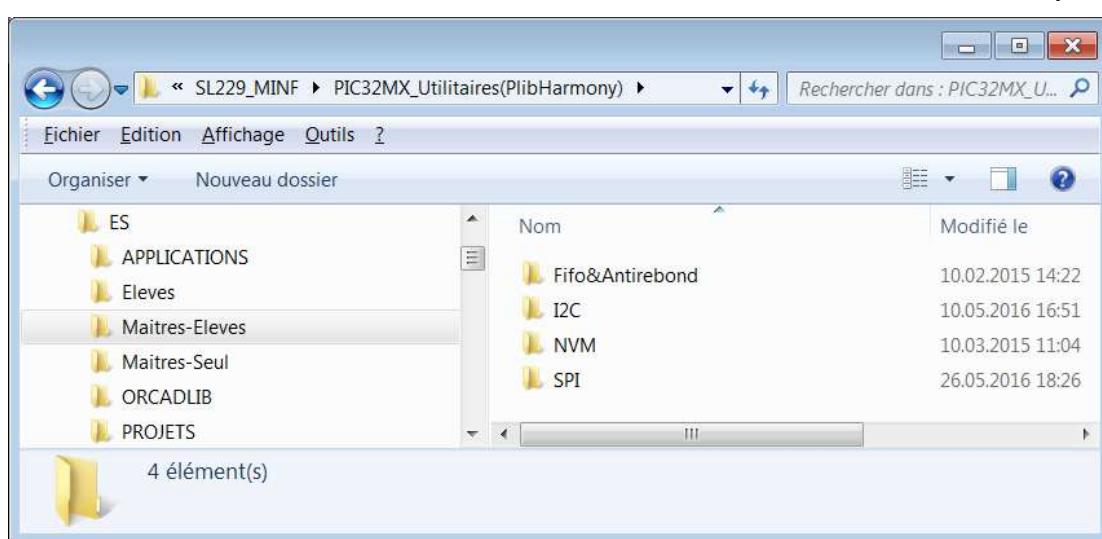
Il est nécessaire de modifier un fichier Harmony, cela s'effectue en copiant le fichier déjà modifié **DS60001156.hconfig** de  
...\\Maitres-Eleves\\SLO\\Modules\\SL229\_MINF\\Harmony\_ES\\v<n>\\config sous  
<Répertoire Harmony>\\v<n>\\bsp\\config

● 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.

**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

<b>3. Prise en main MPLAB X et ICD</b>	<b>3-1</b>
<b>3.1. Réglages et première utilisation</b>	<b>3-1</b>
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
<b>3.2. Exécution du projet avec ICD</b>	<b>3-6</b>
3.2.1. Run main project	3-6
3.2.2. Debug Main Project	3-6
3.2.2.1. Pause/Continuer	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 Curor	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
<b>3.3. Utilisation ICD avec IPE</b>	<b>3-15</b>
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
<b>3.4. Conclusion</b>	<b>3-18</b>
<b>3.5. Historique des versions</b>	<b>3-18</b>
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 débugger le PIC32.

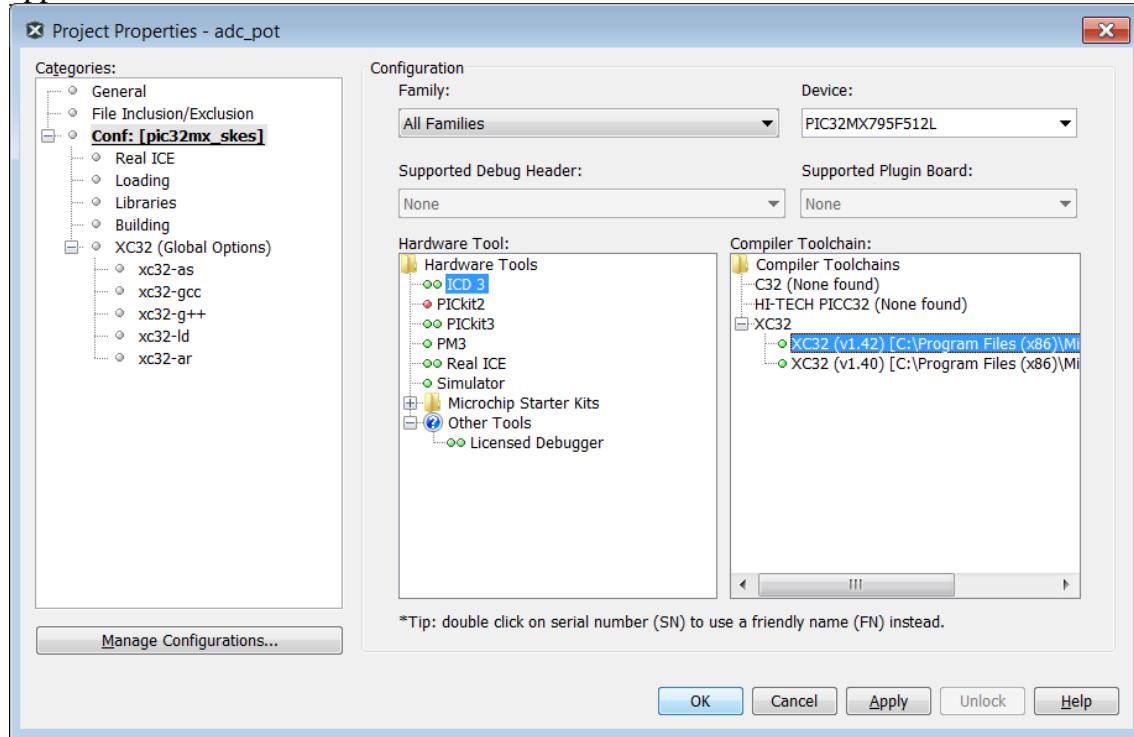


Sonde ICD3

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 ExempleIO (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 :

The screenshot shows the MPLAB X IDE's Output window with the tab 'ExempleIO (Build, Load, ...)' selected. The window displays the following text:

```
*****
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...

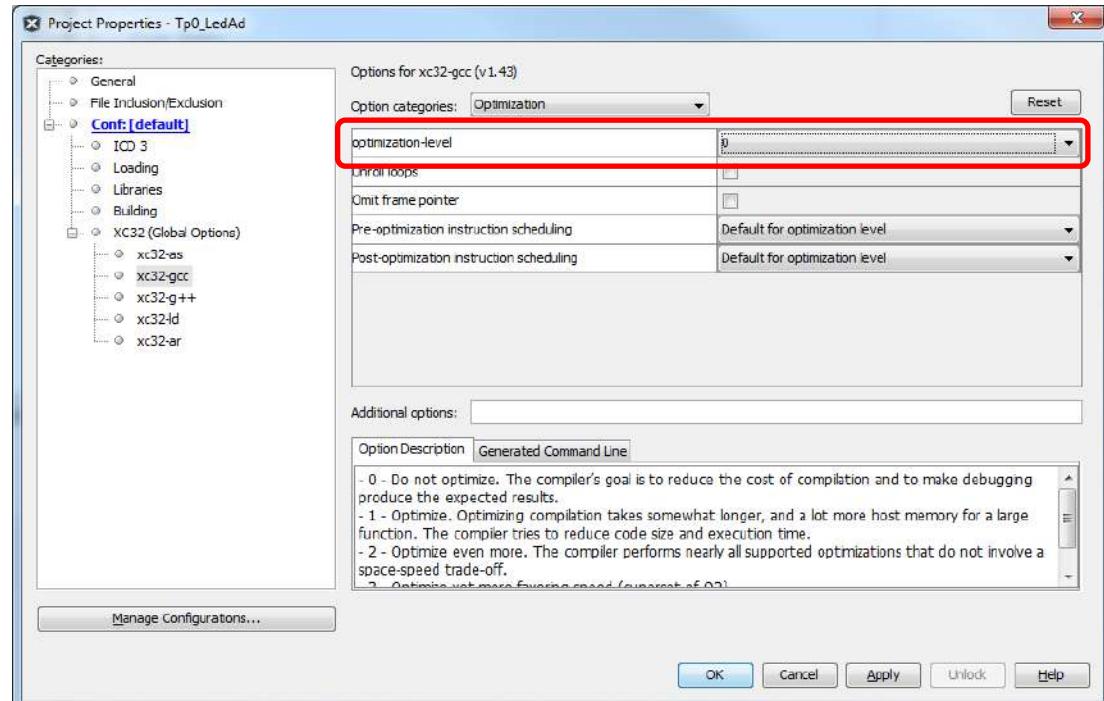
The following memory area(s) will be programmed:
program memory: start address = 0x0, end address = 0x67ff
boot config memory
configuration memory
Programming/Verify complete
```

### 3.1.4. CONFIGURATION DU PROJET

#### 3.1.4.1. NIVEAU D'OPTIMISATION

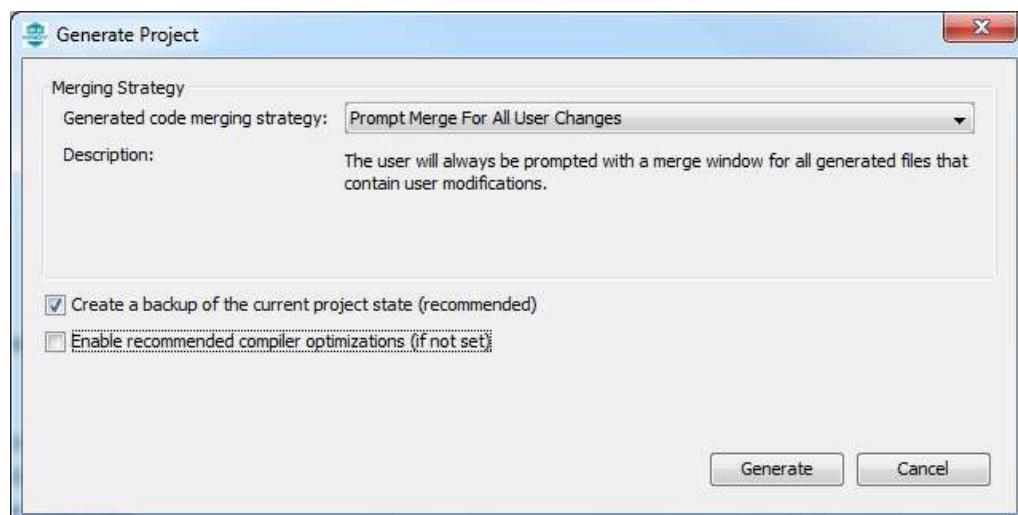
Dans les propriétés, sous xc32-gcc > optimization, par défaut le niveau d'optimisation est 1.

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



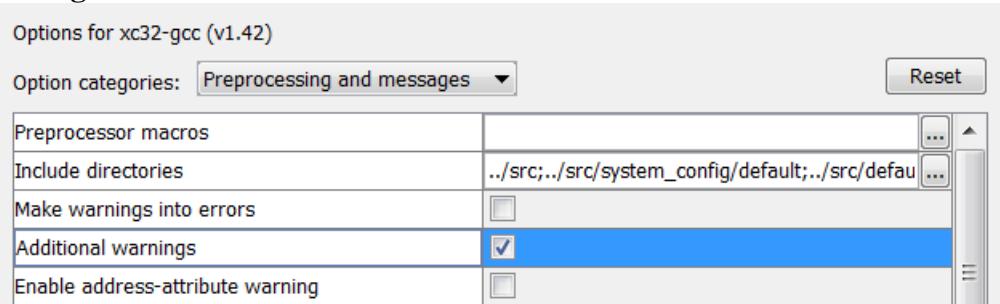
Un niveau d'optimisation à zéro ne sera pas toujours possible avec les exemples plus volumineux, comme ceux démontrant l'éthernet ou l'USB.

☞ 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 COMPILEMENT

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 :

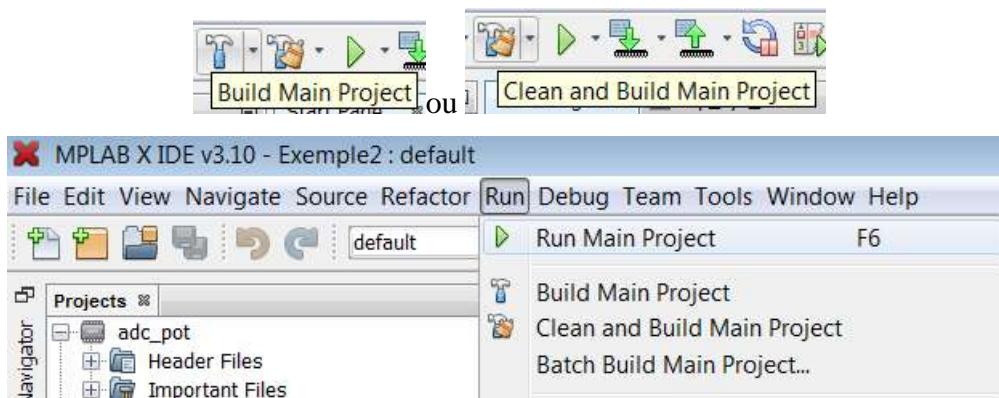


#### 3.2.1. RUN MAIN PROJECT

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

**Set as Main Project**

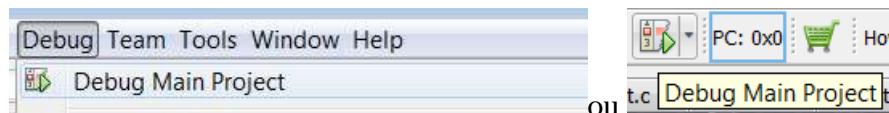
Exécution du projet sans debugging : Il suffit à partir du menu Run d'activer **Run Main Project**. Action directe avec F6, ceci après avoir effectué :



☺ 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 :

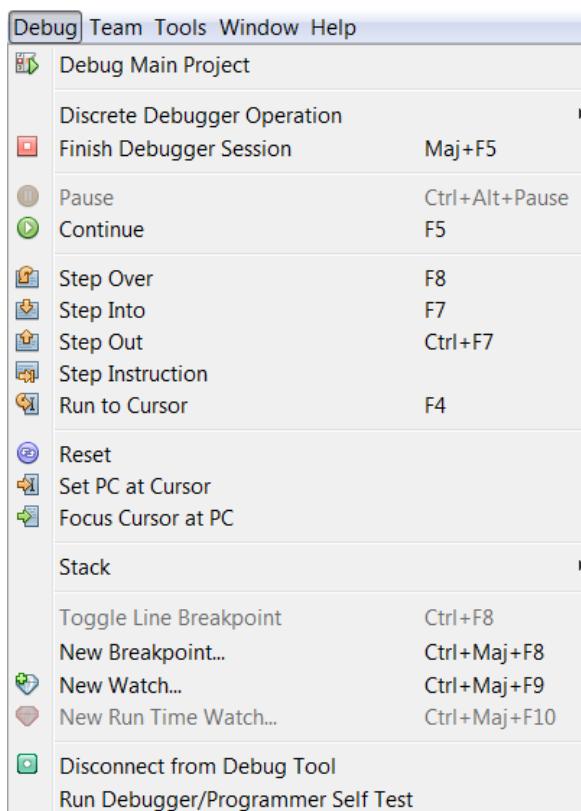


### 3.2.2.1. PAUSE/CONTINUE

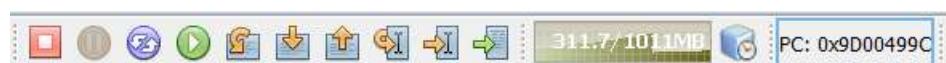
Pour stopper momentanément l'exécution du programme il faut activer au niveau du menu Debug ou dans la barre d'outils.

### 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 rAddress = 0xA0000204, compteurMain = 0x002608E2
180         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

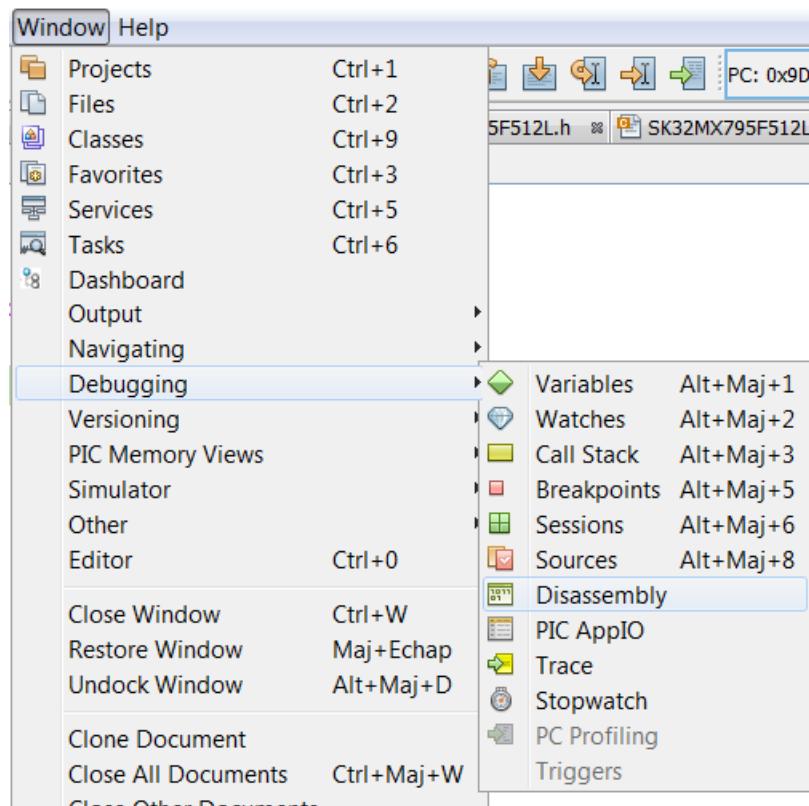
```

### 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



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) {
31     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)
32     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 CUROR

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
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

```

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

```
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
```

### 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
204          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.

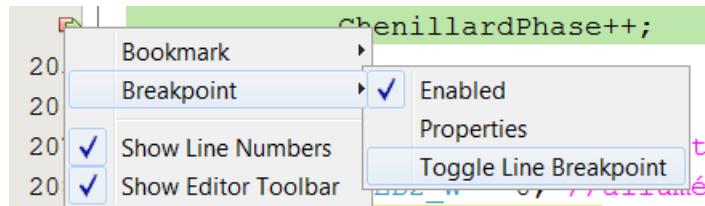
```

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

```

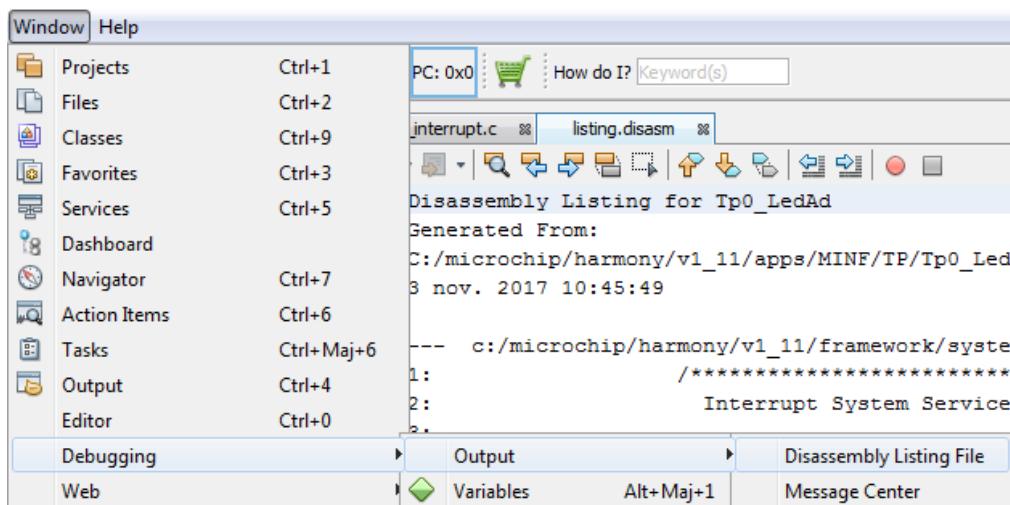
La ligne sur laquelle le programme est arrêté est colorée en vert.

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.

```

1 Disassembly Listing for TpTestKitPic32
2 Generated From:
3 D:/H/etLabo/SL229_MINF/Labo_2013_2014/TpTestPIC32/TpTestKitPic32.X/dist/default/deb
4 12 févr. 2014 17:34:08
5
6 --- d:/h/etlabo/sl229_minf/lab0_2013_2014/tptestpic32/sk32mx795f5121.c -----
7 1: /*-----*/
8 2: /* fichier SK32MX795F512L.c */
9 3: /*-----*/
10 4: /* Description : Initialisation du starter kit */
11 5: /*
12 6: /* Auteur : F. Dominé
13 7: /* Création : 15.12.2011
14 8: /* adaptation : 06.02.2014 C. Huber
15 9: /* Version : V1.0
16 10: /* Compilateur : C32 V2.01
17 11: /*-----*/
18

```

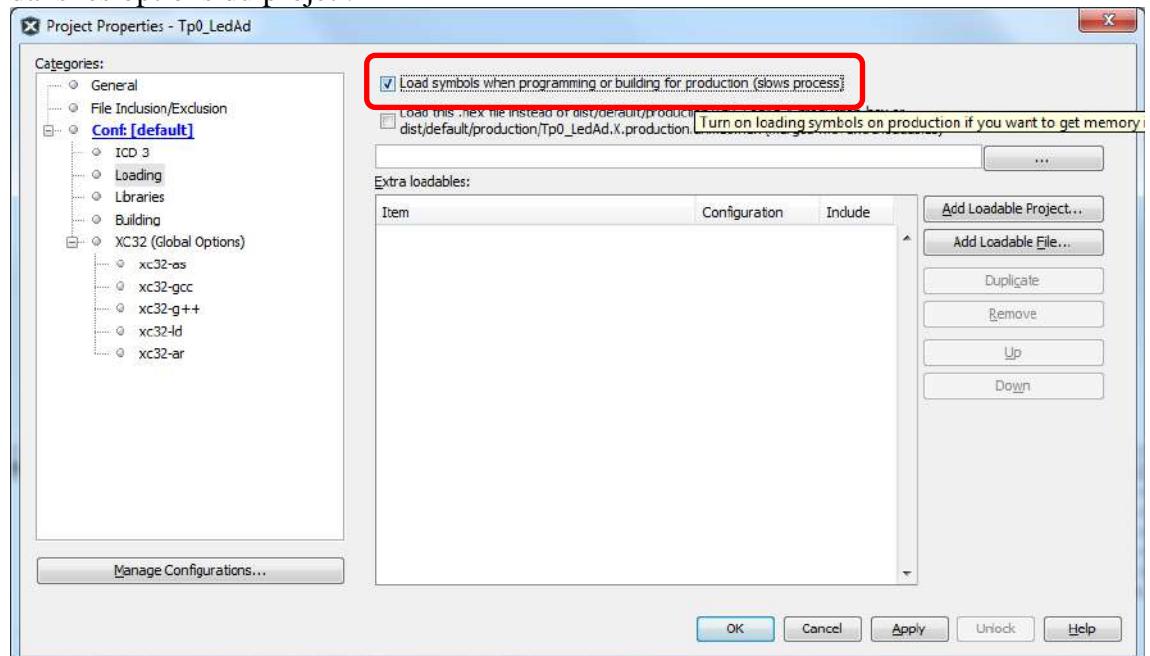
Avec par exemple la boucle du programme principal.

```

1858 178:           while(1) {
1859 179:                 // Ne rien faire (juste un comptage
1860 180:                 compteMain++;
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

```

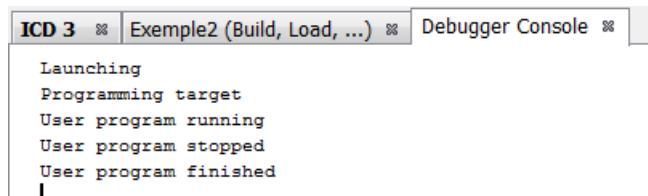
Pour que listing désassemblé soit disponible, il faut mettre la coche suivante dans les options du projet :



### 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 :



```
ICD 3 * Exemple2 (Build, Load, ...) * Debugger Console *  
Launching  
Programming target  
User program running  
User program stopped  
User program finished
```

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

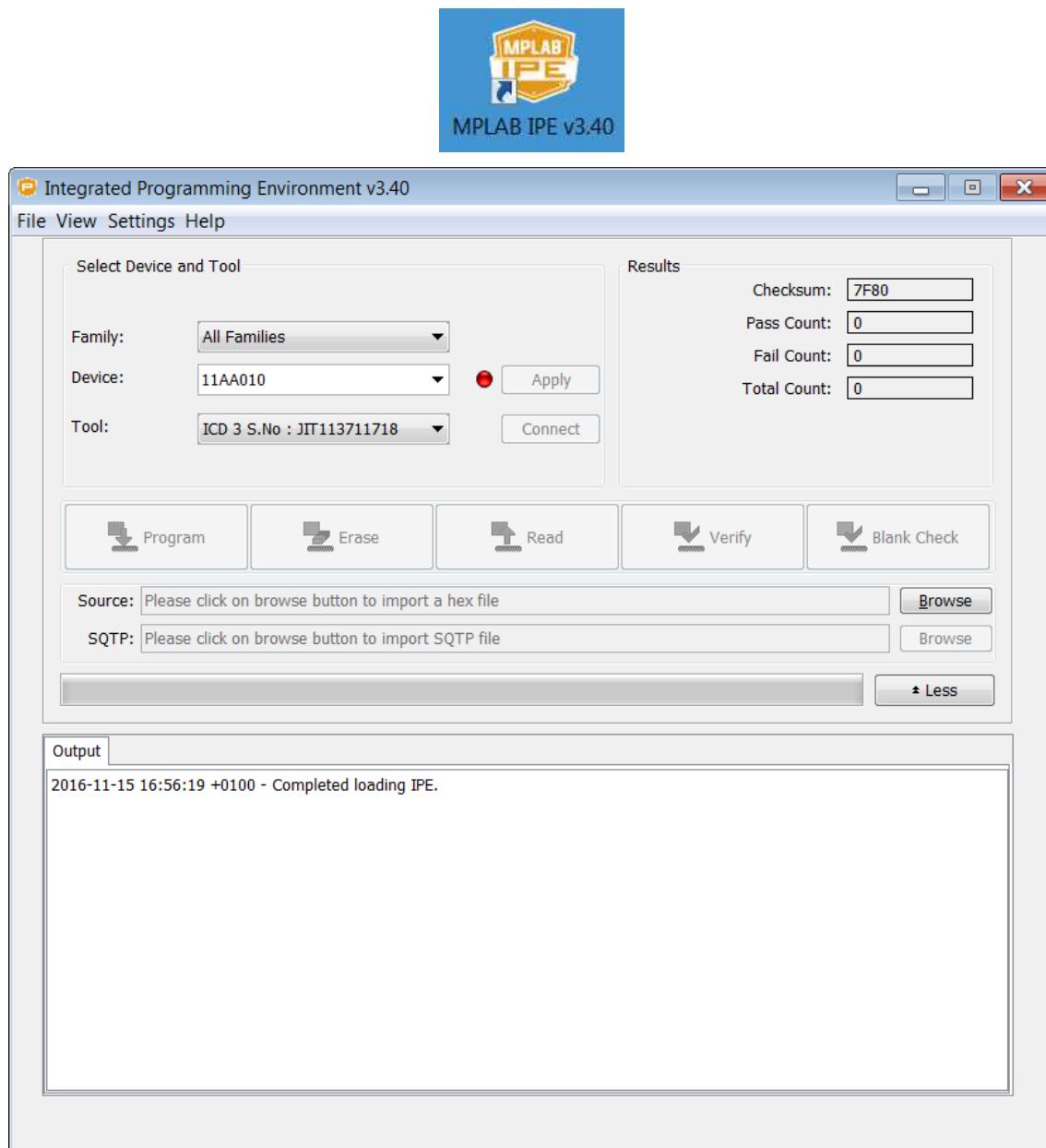
### 3.3. UTILISATION ICD AVEC IPE

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'outils IPE (Integrated Programming Environement) 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'IPÉ

Il faut lancer le MPLAB IPE :



### 3.3.2. CONFIGURATION

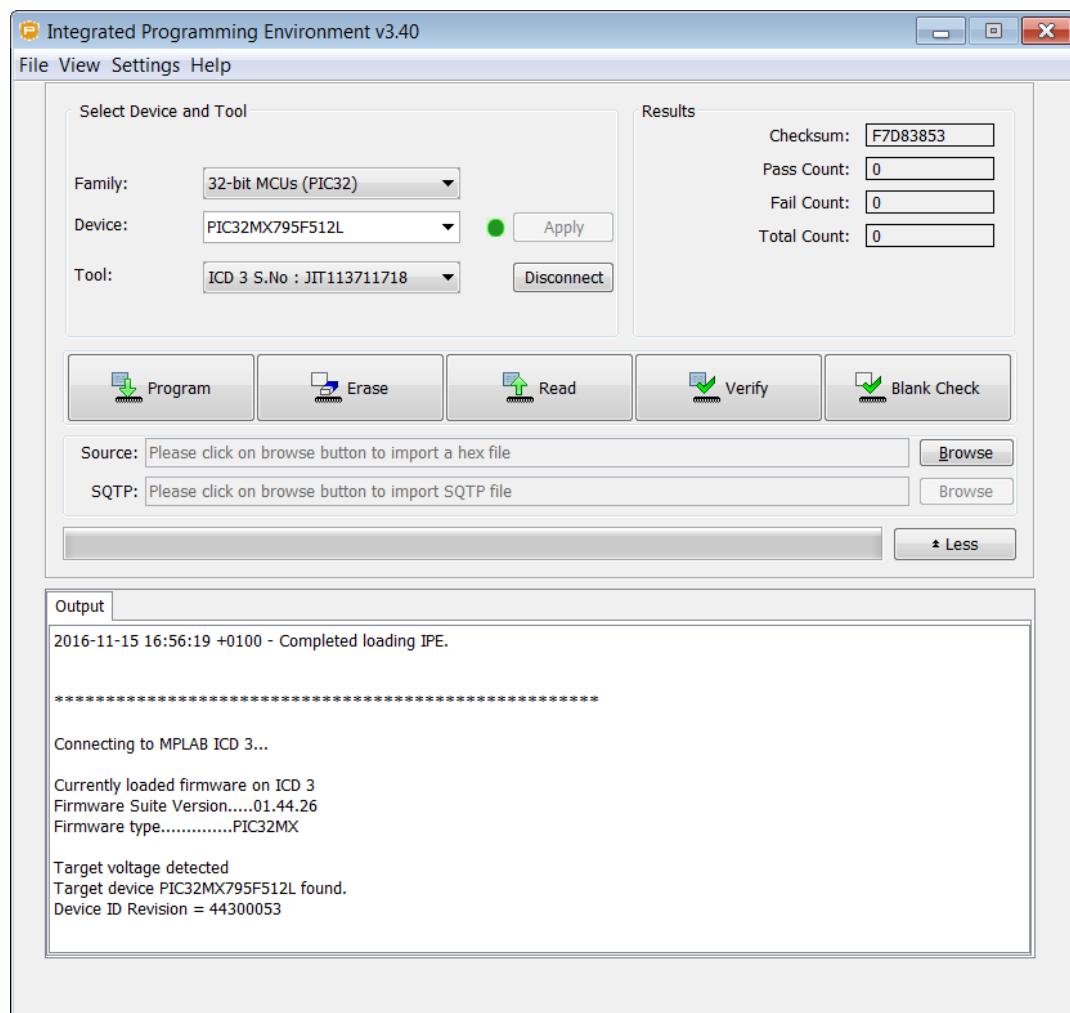
Il est nécessaire de sélectionner le modèle de uC, ainsi que l'outil utilisé :



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

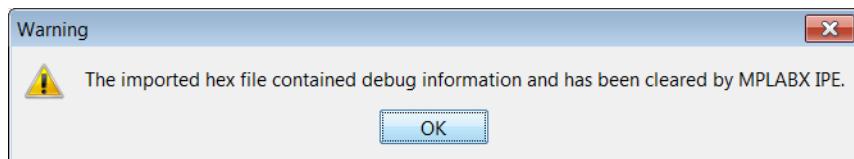
Pour charger un fichier hex, aller sous File > Import > Hex.

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

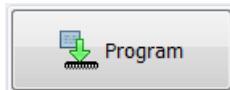
Dans notre exemple :



On peut obtenir :

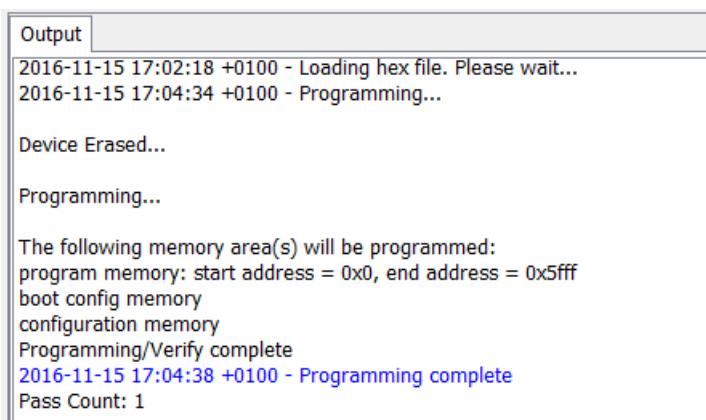


### 3.3.4. ACTION PROGRAM



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<sup>ère</sup> 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

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

# **Chapitre 4**

## **Gestion des entrées-sorties**

**❖ T.P. PIC32MX**

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



## CONTENU DU CHAPITRE 4

<b>4. Gestion des entrées-sorties</b>	<b>4-1</b>
<b>  4.1. Version MPLABX &amp; Harmony</b>	<b>4-1</b>
<b>  4.2. Configuration des fusibles</b>	<b>4-1</b>
4.2.1. Configuration générée dans system_init.c	4-2
4.2.2. Contenu du fichier system_init.c	4-2
<b>  4.3. Configuration des E/S</b>	<b>4-3</b>
4.3.1. Affichage du Pin Diagram	4-3
4.3.2. Pin Setting	4-4
4.3.3. Pin Table	4-5
4.3.4. Les fonctions liées au BSP	4-6
4.3.4.1. Localisation du fichier system_init.c	4-6
4.3.4.2. Contenu de la fonction SYS_Initialize	4-6
<b>  4.4. Concept des E/S digitales</b>	<b>4-14</b>
4.4.1. Schéma d'une ligne d'un port	4-14
4.4.2. Configuration d'une E/S digitale	4-15
4.4.3. Caractéristiques électriques des E/S	4-15
<b>  4.5. Les fonctions de plib_ports.h</b>	<b>4-16</b>
4.5.1. Vues d'ensemble des fonctions de gestion d'un port	4-16
4.5.2. Vue d'ensemble des fonctions de gestion d'un bit d'un port	4-17
4.5.3. Autres fonctions	4-17
4.5.4. Types de données et constantes	4-17
4.5.4.1. Spécification d'un port	4-18
4.5.4.2. Spécification de la position d'un bit	4-18
4.5.4.3. Spécification d'une broche (pin) d'un port	4-18
4.5.4.4. Spécification du mode d'une broche (pin) d'un port	4-18
4.5.5. Fonctions de configuration générale	4-19
4.5.6. Fonctions de configuration d'un port	4-20
4.5.6.1. Initialisation dans SYS_PORTS_Initialize : port A	4-20
4.5.6.2. Etablissement de la direction d'un groupe de broches d'un port	4-20
4.5.6.3. Etablissement de la direction d'une broche d'un port	4-21
4.5.6.4. Etablissement du mode d'une entrée analogique	4-21
4.5.6.5. Gestion Open Drain d'un groupe de broches d'un port	4-21
4.5.6.6. Gestion Open Drain d'une broche d'un port	4-22
4.5.7. Fonctions de lecture des entrées	4-22
4.5.7.1. Fonction PLIB_PORTS_Read	4-22
4.5.7.2. Fonction PLIB_PORTS_ReadLatched (lecture états sorties)	4-22
4.5.7.3. Fonction PLIB_PORTS_PinGet	4-23
4.5.7.4. Fonction PLIB_PORTS_PinGetLatched (lecture sortie)	4-23
4.5.8. Fonctions d'écritures des sorties	4-23
4.5.8.1. Fonction PLIB_PORTS_Clear	4-24
4.5.8.2. Fonction PLIB_PORTS_Set	4-24
4.5.8.3. Fonction PLIB_PORTS_Write	4-24
4.5.8.4. Fonction PLIB_PORTS_Toggle	4-25

4.5.8.5.	Fonction PLIB_PORTS_PinClear	4-25
4.5.8.6.	Fonction PLIB_PORTS_PinSet	4-25
4.5.8.7.	Fonction PLIB_PORTS_PinWrite	4-25
4.5.8.8.	Fonction PLIB_PORTS_PinToggle	4-26
4.5.8.9.	Réalisation du PinToggle, accès direct	4-26
<b>4.6.</b>	<b>BSP pic32mx_skies</b>	<b>4-27</b>
4.6.1.	Contenu du fichier BSP_config.h	4-27
4.6.2.	Contenu du fichier BSP_sys_init.c	4-30
4.6.2.1.	Extrait fonction BSP_Initialize	4-30
4.6.3.	Ecriture et lecture directe	4-31
4.6.4.	Ecriture et lecture avec les fonctions PLIB_PORTS	4-32
<b>4.7.</b>	<b>Gestion des leds et switch dans le BSP</b>	<b>4-33</b>
4.7.1.	Définitions BSP des leds	4-33
4.7.2.	La fonction BSP_LEDStateSet	4-33
4.7.3.	La fonction BSP_LEDOn	4-34
4.7.4.	La fonction BSP_LEDOff	4-34
4.7.5.	La fonction BSP_LEDToggle	4-34
4.7.6.	La fonction BSP_LEDStateGet	4-35
4.7.7.	Définitions BSP des switches	4-35
4.7.8.	La fonction BSP_SwitchStateGet	4-36
4.7.9.	La fonction BSP_EnableHbridge	4-36
<b>4.8.</b>	<b>Gestion du convertisseur AD</b>	<b>4-37</b>
4.8.1.	Schéma de principe en mode alterné	4-38
4.8.2.	Schéma de principe en mode scan	4-39
4.8.3.	Entrées analogiques du PIC32MX795F512L	4-40
<b>4.9.</b>	<b>Les fonctions de plib_adc.h</b>	<b>4-41</b>
4.9.1.	Exemple en mode alterné	4-42
4.9.1.1.	Fichier Mc32DriverAdcAlt.h	4-42
4.9.1.1.	Fichier Mc32DriverAdcAlt.c	4-43
4.9.2.	Exemple en mode scan	4-45
4.9.2.1.	Fichier Mc32DriverAdc.h	4-45
4.9.2.2.	Fichier Mc32DriverAdc.c	4-46
<b>4.10.</b>	<b>Application de test</b>	<b>4-48</b>
4.10.1.	Modification de l'initialisation et test préliminaire	4-48
4.10.2.	Modifications de app.h pour gestion de l'adc	4-48
4.10.3.	Modifications de app.c pour gestion des IO	4-49
4.10.3.1.	Ajout action IO dans case APP_STATE_INIT	4-49
4.10.3.2.	Ajout action IO dans case APP_STATE_SERVICE_TASKS	4-50
4.10.4.	Modification cyclique de appData.state	4-51
4.10.5.	Contrôle du fonctionnement	4-52
<b>4.11.</b>	<b>Historique des versions</b>	<b>4-53</b>
4.11.1.	V1.0 Mai 2013	4-53
4.11.2.	Version 1.1 Novembre 2013	4-53
4.11.3.	Version 1.2 Décembre 2013	4-53
4.11.4.	Version 1.3 Mars 2014	4-53
4.11.5.	Version 1.5 Octobre 2014	4-53
4.11.6.	Version 1.6 Septembre 2015	4-53
4.11.7.	Version 1.7 Novembre 2016	4-53

4.11.8.	Version 1.8 novembre 2017	4-53
4.11.9.	Version 1.81 décembre 2018	4-53



## 4. GESTION DES ENTRÉES-SORTIES

Dans ce chapitre, nous allons étudier la réalisation du BSP adapté au KIT PIC32MX et ce qu'il apporte lorsqu'il est utilisé dans un projet généré par le MHC (MPLAB Harmony Configurator).

Nous étudierons aussi les modifications et les compléments nécessaires pour obtenir un projet fonctionnel, ainsi que les fonctions PLIB à disposition pour la gestion d'entrées-sorties digitales, ainsi que pour les entrées analogiques.

Les documents de référence sont :

- La documentation "PIC32 Family Reference Manual" :  
Section 12 : I/O Ports
- Pour les détails spécifiques au PIC32MX795F512L, il faut se référer au document "PIC32MX5XX/6XX/7XX Family Data Sheet" :  
Section 12 : I/O Ports et section 23 : 10-bit Analog-to-Digital Converter
- La documentation d'Harmony, qui se trouve dans <Répertoire Harmony>\v<n>\doc :  
Section MPLAB Harmony Framework Reference > Peripheral Libraries Help,  
sous-sections Ports Peripheral Library et ADC Peripheral Library

### 4.1. VERSION MPLABX & HARMONY

L'exemple utilisé dans le chapitre 4 a été réalisé en utilisant :

- MPLABX 3.40
- Harmony 1.08\_01
- XC32 1.42

### 4.2. CONFIGURATION DES FUSIBLES

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

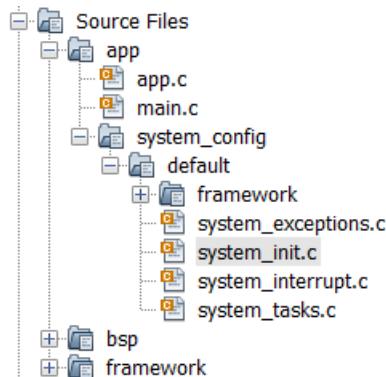
Le détail de la configuration ayant déjà été abordé au chapitre 2, nous aborderons ici uniquement le code résultant.

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.



#### 4.2.1. CONFIGURATION GÉNÉRÉE DANS SYSTEM\_INIT.C

L'ensemble des #pragma config est placé par le MHC dans le fichier system\_init.c qui se trouve à l'emplacement suivant dans l'arborescence.



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

```

// *****
// Section: Configuration Bits
// *****

/*** DEVCFG0 ***/
#pragma config DEBUG = ON
#pragma config ICESEL = ICS_PGx2
#pragma config PWP = OFF
#pragma config BWP = OFF
#pragma config CP = OFF

/*** DEVCFG1 ***/
#pragma config FNOSC = PRIPLL
#pragma config FSOSCEN = OFF
#pragma config IESO = OFF
#pragma config POSCMOD = XT
#pragma config OSCIOFNC = OFF
#pragma config FPBDIV = DIV_1
#pragma config FCKSM = CSECMD
#pragma config WDTPS = PS1048576
#pragma config FWDTEN = OFF

/*** DEVCFG2 ***/
#pragma config FPLLIDIV = DIV_2
#pragma config FPllMUL = MUL_20
#pragma config FPLLODIV = DIV_1
#pragma config UPLLIDIV = DIV_2
#pragma config UPLLEN = ON

/*** DEVCFG3 ***/
#pragma config USERID = 0xffff
#pragma config FSRSSEL = PRIORITY_7
#pragma config FMIEN = OFF
#pragma config FETHIO = ON
#pragma config FCANIO = ON
#pragma config FUSBIDIO = ON
#pragma config FVBUSONIO = ON

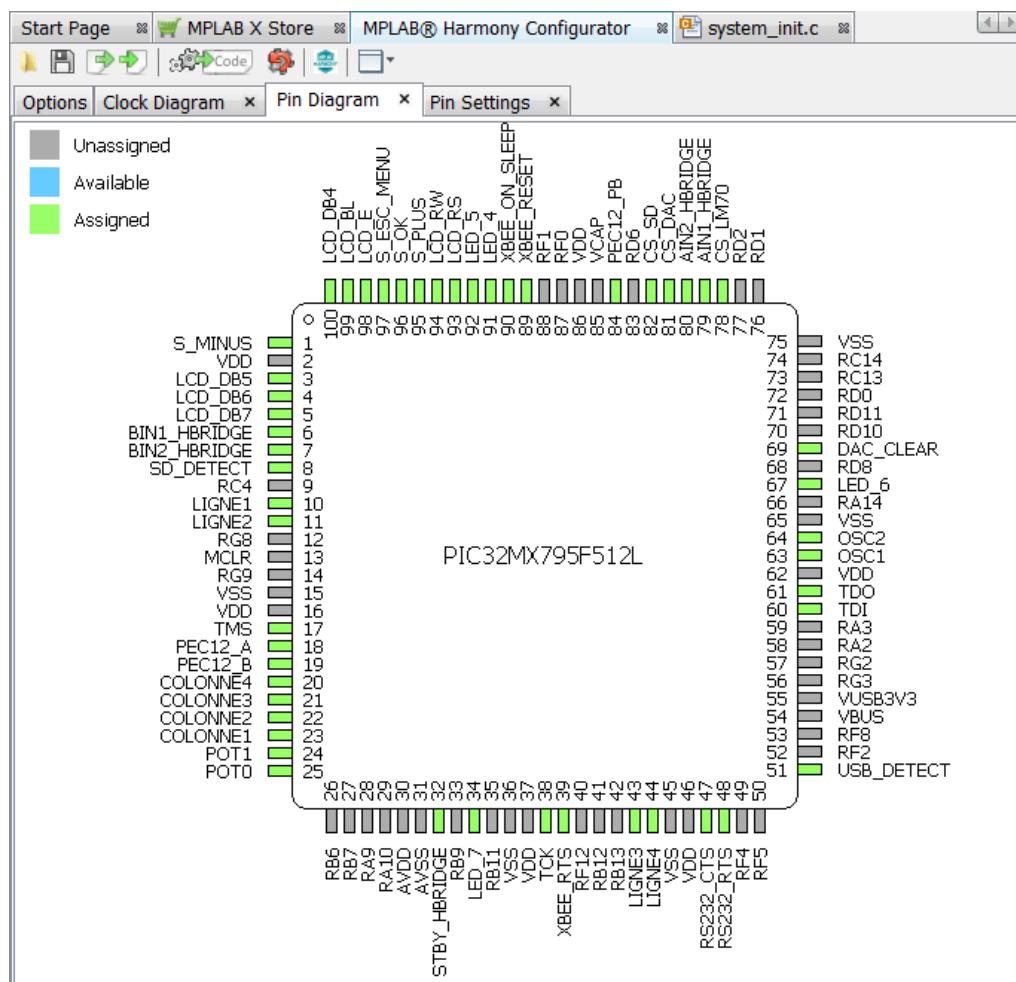
```

## 4.3. CONFIGURATION DES E/S

Une grande partie de la configuration des entrées-sorties est fournie par la sélection du BSP.

### 4.3.1. AFFICHAGE DU PIN DIAGRAM

Une partie des définitions des entrées-sorties est fournie par le fichier **BSP.xml** du BSP sélectionné. Les choix imposés par le BSP peuvent être visualisés au niveau du *Pin Diagram* :



On remarque la personnalisation des broches.

#### 4.3.2. PIN SETTING

Sous l'onglet Pin Settings, on peut observer le détail de la configuration et modifier les broches qui n'ont pas été configurées.

Pin	Name	Voltage Tolerance	Function	Direction (TRIS)	Latch (LAT)	Open Drain (ODC)	Mode (ADPCFG)	Change Notification	Pull Up (CNPUE)
1	RG15	5V	S_MINUS	In	Low	<input type="checkbox"/>	Digital		
2	VDD	5V		In	n/a	<input type="checkbox"/>	Digital		
3	RE5	5V	LCD_DB5	Out	High	<input type="checkbox"/>	Digital		
4	RE6	5V	LCD_DB6	Out	High	<input type="checkbox"/>	Digital		
5	RE7	5V	LCD_DB7	Out	High	<input type="checkbox"/>	Digital		
6	RC1	5V	BIN1_HBRIDGE	Out	Low	<input type="checkbox"/>	Digital		
7	RC2	5V	BIN2_HBRIDGE	Out	Low	<input type="checkbox"/>	Digital		
8	RC3	5V	SD_DETECT	In	Low	<input type="checkbox"/>	Digital		
9	RC4	5V		In	n/a	<input type="checkbox"/>	Digital		
10	RG6	5V	LIGNE1	In	Low	<input type="checkbox"/>	Digital	<input checked="" type="checkbox"/>	
11	RG7	5V	LIGNE2	In	Low	<input type="checkbox"/>	Digital	<input checked="" type="checkbox"/>	
12	RG8	5V		In	n/a	<input type="checkbox"/>	Digital	<input checked="" type="checkbox"/>	
13	MCLR	5V		In	n/a	<input type="checkbox"/>	Digital		
14	RG9	5V		In	n/a	<input type="checkbox"/>	Digital	<input checked="" type="checkbox"/>	
15	VSS	5V		In	n/a	<input type="checkbox"/>	Digital		
16	VDD	5V		In	n/a	<input type="checkbox"/>	Digital		

En effet, les éléments nommés (ayant une valeur sous Function) ne peuvent pas être modifiés.

On peut également lister par port :

Pin	Name	Voltage Tolerance	Function	Direction (TRIS)	Latch (LAT)	Open Drain (ODC)	Mode (ADPCFG)	Change Notification	Pull Up (CNPUE)
25	RB0		POTO	In	Low	<input type="checkbox"/>	Analog	<input type="checkbox"/>	<input type="checkbox"/>
24	RB1		POT1	In	Low	<input type="checkbox"/>	Analog	<input type="checkbox"/>	<input type="checkbox"/>
23	RB2		COLONNE1	In	Low	<input type="checkbox"/>	Digital	<input checked="" type="checkbox"/>	
22	RB3		COLONNE2	In	Low	<input type="checkbox"/>	Digital	<input checked="" type="checkbox"/>	
21	RB4		COLONNE3	In	Low	<input type="checkbox"/>	Digital	<input checked="" type="checkbox"/>	
20	RB5		COLONNE4	In	Low	<input type="checkbox"/>	Digital	<input checked="" type="checkbox"/>	
26	RB6			In	n/a	<input type="checkbox"/>	Analog		
27	RB7			In	n/a	<input type="checkbox"/>	Analog		
32	RB8		STBY_HBRIDGE	Out	Low	<input type="checkbox"/>	Digital		
33	RB9			In	n/a	<input type="checkbox"/>	Analog		
34	RB10		LED_7	Out	High	<input type="checkbox"/>	Digital		
35	RB11			In	n/a	<input type="checkbox"/>	Analog		
41	RB12			In	n/a	<input type="checkbox"/>	Analog		
42	RB13			In	n/a	<input type="checkbox"/>	Analog		
43	RB14		LIGNE3	In	Low	<input type="checkbox"/>	Digital		
44	RB15		LIGNE4	In	Low	<input type="checkbox"/>	Digital	<input checked="" type="checkbox"/>	
6	RC1	5V	BIN1_HBRIDGE	Out	Low	<input type="checkbox"/>	Digital		
7	RC2	5V	BIN2_HBRIDGE	Out	Low	<input type="checkbox"/>	Digital		
8	RC3	5V	SD_DETECT	In	Low	<input type="checkbox"/>	Digital		

Pour le port B, on remarque que les entrées non déclarées sont en analogique par défaut.

On peut ainsi observer le détail de la configuration effectuée.

### 4.3.3. PIN TABLE

Dans la fenêtre du bas, où l'on trouve Output, il y a aussi l'onglet **MPLAB® Harmony Configurator\*** qui permet d'observer l'attribution des broches associées à un driver.

⌚ Lors de l'ajout de driver comme par exemple OC, I2C, SPI ou UART, on verra apparaître l'utilisation des broches.

☺ La pin table permet aussi, pour d'autres modèles de PIC qui auraient des broches remappables, d'effectuer le choix.

Output		MPLAB® Harmony Configurator*															
Output		Pin Table															
		Pin conflict resolved. See output window.															
Module	Function	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Board Support Package	LED_5								🔒								
	LED_6									🔒							
	LED_7												🔒				
	CS_LM70																
	CS_DAC																
	CS_SD																
	SD_DETECT																
	DAC_CLEAR																🔒

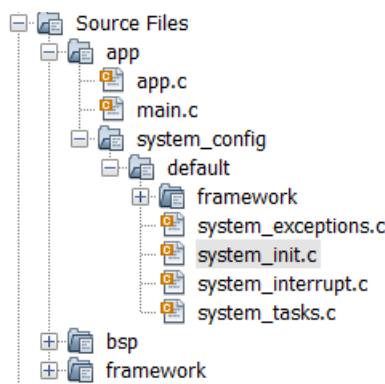
#### 4.3.4. LES FONCTIONS LIÉES AU BSP

Le choix d'un BSP a pour effet d'ajouter au projet les librairies propres à la carte en question ainsi que de générer et d'appeler quelques fonctions d'initialisation.

C'est la fonction `SYS_Initialize`, qui se trouve dans le fichier `system_init.c`, qui effectue les appels importants en ce qui concerne l'initialisation du système.

##### 4.3.4.1. LOCALISATION DU FICHIER SYSTEM\_INIT.C

Le fichier `system_init.c` se situe dans la sous-section `system_config` de l'application.



##### 4.3.4.2. CONTENU DE LA FONCTION SYS\_INITIALIZE

Voici le contenu de la fonction `SYS_Initialize` après la génération du code avec le MHC.

```

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 TMRO */
    DRV_TMR0_Initialize();

    /* Initialize System Services */
    SYS_INT_Initialize();

    /* Initialize Middleware */
    /* Enable Global Interrupts */
    SYS_INT_Enable();

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

Le contenu de la fonction SYS\_Initialize correspond à :

- Appel de différentes fonctions d'initialisation :
  - Horloge système. Ceci, en suppléments des configurations bits, donne dans notre cas SYSCLK et PBCLK à 80 MHz.
  - Performances  
La fonction SYS\_DEVCON\_PerformanceConfig() configure les wait states d'accès à la flash et le prefetch cache pour des performances maximales.
- Désactivation des pins JTAG. Ce sont les pins TDI, TDO, TCK et TMS. La fonctionnalité JTAG n'est pas utilisée sur le starter-kit ES est ces pins sont utilisées à d'autres fins.  
● Il faut bien l'appel de la fonction SYS\_DEVCON\_JTAGDisable et non pas SYS\_DEVCON\_JTAGEnable. Sinon, les fonctionnalités câblées sur les 4 pins concernées ne seront pas utilisables.  
Pour la programmation et le debug, ce sont les pins ICSP PGECh et PGEDn, propres à Microchip, qui sont utilisées.
- La fonction SYS\_PORTS\_Initialize est appelée. Cette dernière effectue l'initialisation des entrées-sorties en fonction des données du fichier BSP.xml.
- La fonction BSP\_Initialize du BSP sélectionné est appelée.
- Ensuite, avec la fonction SYS\_INT\_Initialize, il y a la configuration du mode multi vecteurs pour les interruptions.
- La section "Initialize drivers" varie en fonction des différents pilotes sélectionnés dans le MHC. Dans notre exemple il y a uniquement un timer.
- La fonction SYS\_INT\_Enable autorise globalement les interruptions.
- Et finalement avec la fonction APP\_Initialize il est possible d'effectuer une action d'initialisation spécifique aux besoins de l'application.

Certaines de ces fonctions sont détaillées ci-dessous.

#### 4.3.4.2.1. Contenu de la fonction SYS\_PORTS\_Initialize

Voici le contenu (qui dépend du BSP choisi) de la fonction SYS\_PORTS\_Initialize.

```
void SYS_PORTS_Initialize(void)
{
    /* AN and CN Pins Initialization */
    PLIB_PORTS_AnPinsModeSelect(PORTS_ID_0,
        SYS_PORT_AD1PCFG, PORTS_PIN_MODE_DIGITAL);
    PLIB_PORTS_Cn PinsPullUpEnable(PORTS_ID_0,
        SYS_PORT_CNPUE);
    PLIB_PORTS_Cn PinsEnable(PORTS_ID_0, SYS_PORT_CNEN);
    PLIB_PORTS_ChangeNoticeEnable(PORTS_ID_0);

    /* PORT A Initialization */
    PLIB_PORTS_OpenDrainEnable(PORTS_ID_0, PORT_CHANNEL_A,
        SYS_PORT_A_ODC);
    PLIB_PORTS_Write(PORTS_ID_0, PORT_CHANNEL_A,
```

```

        SYS_PORT_A_LAT);
PLIB_PORTS_DirectionOutputSet( PORTS_ID_0,
                                PORT_CHANNEL_A,   SYS_PORT_A_TRIS ^ 0xFFFF);

/* PORT C Initialization */
PLIB_PORTS_OpenDrainEnable(PORTS_ID_0,  PORT_CHANNEL_C,
                           SYS_PORT_C_ODC);
PLIB_PORTS_Write( PORTS_ID_0,  PORT_CHANNEL_C,
                  SYS_PORT_C_LAT);
PLIB_PORTS_DirectionOutputSet( PORTS_ID_0,
                                PORT_CHANNEL_C,   SYS_PORT_C_TRIS ^ 0xFFFF);

/* PORT D Initialization */
PLIB_PORTS_OpenDrainEnable(PORTS_ID_0,  PORT_CHANNEL_D,
                           SYS_PORT_D_ODC);
PLIB_PORTS_Write( PORTS_ID_0,  PORT_CHANNEL_D,
                  SYS_PORT_D_LAT);
PLIB_PORTS_DirectionOutputSet( PORTS_ID_0,
                                PORT_CHANNEL_D,   SYS_PORT_D_TRIS ^ 0xFFFF);

/* PORT E Initialization */
PLIB_PORTS_OpenDrainEnable(PORTS_ID_0,  PORT_CHANNEL_E,
                           SYS_PORT_E_ODC);
PLIB_PORTS_Write( PORTS_ID_0,  PORT_CHANNEL_E,
                  SYS_PORT_E_LAT);
PLIB_PORTS_DirectionOutputSet( PORTS_ID_0,
                                PORT_CHANNEL_E,   SYS_PORT_E_TRIS ^ 0xFFFF);

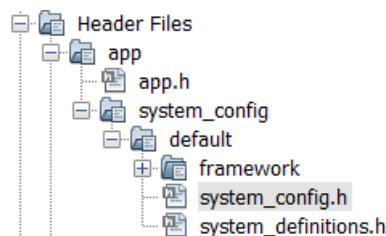
/* PORT F Initialization */
PLIB_PORTS_OpenDrainEnable(PORTS_ID_0,  PORT_CHANNEL_F,
                           SYS_PORT_F_ODC);
PLIB_PORTS_Write( PORTS_ID_0,  PORT_CHANNEL_F,
                  SYS_PORT_F_LAT);
PLIB_PORTS_DirectionOutputSet( PORTS_ID_0,
                                PORT_CHANNEL_F,   SYS_PORT_F_TRIS ^ 0xFFFF);

/* PORT G Initialization */
PLIB_PORTS_OpenDrainEnable(PORTS_ID_0,  PORT_CHANNEL_G,
                           SYS_PORT_G_ODC);
PLIB_PORTS_Write( PORTS_ID_0,  PORT_CHANNEL_G,
                  SYS_PORT_G_LAT);
PLIB_PORTS_DirectionOutputSet( PORTS_ID_0,
                                PORT_CHANNEL_G,   SYS_PORT_G_TRIS ^ 0xFFFF);

}

```

Les définitions des valeurs sont réalisées dans le fichier system\_config.h qui se trouve sous :



#### 4.3.4.2.2. Contenu du fichier system\_config.h

Voici le contenu (qui dépend de la device configuration et du BSP choisi) du fichier system\_config.h. Ce sont les valeurs de configuration des différents registres qui concernent les ports.

```
#include "BSP_config.h"

// *****
// Section: System Service Configuration
// *****

// *****
/* Common System Service Configuration Options
*/
#define SYS_VERSION_STR          "1.08.01"
#define SYS_VERSION                10801

// *****
/* Clock System Service Configuration Options
*/
#define SYS_CLK_FREQ                  80000000ul
#define SYS_CLK_BUS_PERIPHERAL_1      80000000ul
#define SYS_CLK_UPPLL_BEFORE_DIV2_FREQ 48000000ul
#define SYS_CLK_CONFIG_PRIMARY_XTAL   8000000ul
#define SYS_CLK_CONFIG_SECONDARY_XTAL 0ul

/*** Interrupt System Service Configuration ***/
#define SYS_INT                      true

/*** Ports System Service Configuration ***/
#define SYS_PORT_A_TRIS              0x460c
#define SYS_PORT_A_LAT                0x80c0
#define SYS_PORT_A_ODC                0x0

#define SYS_PORT_B_TRIS              0xffff
#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

#define SYS_PORT_D_TRIS              0x4dc7
#define SYS_PORT_D_LAT                0x8238
#define SYS_PORT_D_ODC                0x0

#define SYS_PORT_E_TRIS              0x300
#define SYS_PORT_E_LAT                0xf7
#define SYS_PORT_E_ODC                0x0

#define SYS_PORT_F_TRIS              0x113f
```

```

#define SYS_PORT_F_LAT          0x2000
#define SYS_PORT_F_ODC          0x0

#define SYS_PORT_G_TRIS         0xf3cc
#define SYS_PORT_G_LAT          0x2
#define SYS_PORT_G_ODC          0x0

// *****
// Section: Driver Configuration
// *****

/** Timer Driver Configuration */
#define DRV_TMR_INTERRUPT_MODE      true

/** Timer Driver 0 Configuration */
#define DRV_TMR_PERIPHERAL_ID_IDX0    TMR_ID_1
#define DRV_TMR_INTERRUPT_SOURCE_IDX0  INT_SOURCE_TIMER_1
#define DRV_TMR_INTERRUPT_VECTOR_IDX0  INT_VECTOR_T1
#define DRV_TMR_ISR_VECTOR_IDX0       TIMER_1_VECTOR
#define DRV_TMR_INTERRUPT_PRIORITY_IDX0 INT_PRIORITY_LEVEL3
#define DRV_TMR_INTERRUPT_SUB_PRIORITY_IDX0 INT_SUBPRIORITY_LEVEL0
#define DRV_TMR_CLOCK_SOURCE_IDX0      DRV_TMR_CLKSOURCE_INTERNAL
#define DRV_TMR_OPERATION_MODE_IDX0    DRV_TMR_OPERATION_MODE_16_BIT
#define DRV_TMR_ASYNC_WRITE_ENABLE_IDX0 false
#define DRV_TMR_POWER_STATE_IDX0

// *****
// Section: Middleware & Other Library Configuration
// *****

// *****
/* BSP Configuration Options
*/
#define BSP_OSC_FREQUENCY 8000000

```

#### 4.3.4.2.3. La fonction BSP\_Initialize

La fonction BSP\_Initialize est spécifique au starter-kit PIC32MX795F512L de l'ES, schéma 11020\_B.

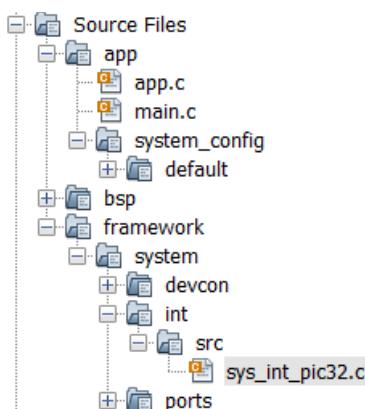
¶ La fonction est quasiment vide. A cause des broches non configurées, il est nécessaire d'effectuer une configuration des broches qui peuvent être analogiques ou digitales.

```
void BSP_Initialize(void )
{
    // Pour ne pas entrer en conflit avec le JTAG
    SYS_DEVCON_JTAGDisable(); // déjà fait mais si on oublie

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

#### 4.3.4.2.4. La fonction SYS\_INT\_Initialize

Cette fonction qui se trouve dans le fichier sys\_int\_pic32.c appelle une fonction de la librairie pour sélectionner la gestion des interruptions multi-vecteur. Localisation du fichier :

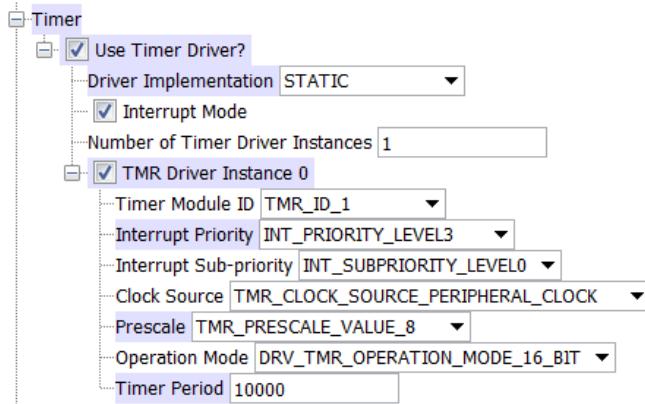


```
void SYS_INT_Initialize ( void )
{
    /* enable the multi vector */
    PLIB_INT_MultiVectorSelect( INT_ID_0 );
}
```

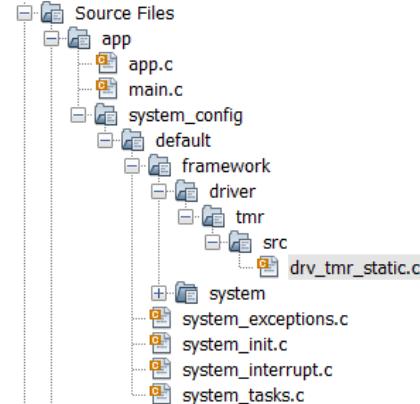
#### 4.3.4.2.5. La fonction DRV\_TMR0\_Initialize

Cette fonction existe car lors de la réalisation du projet Harmony avec le MHC, on a défini les éléments suivants :

Sélection d'un driver timer



Localisation du fichier



La fonction est implémentée dans le fichier `drv_timer_static.c`

```
void DRV_TMR0_Initialize(void)
{
    /* Initialize Timer Instance0 */
    PLIB_TMR_Stop(TMR_ID_1); /* Disable Timer */
    /* Select clock source */
    PLIB_TMR_ClockSourceSelect(TMR_ID_1,
                               TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
    /* Select prescalar value */
    PLIB_TMR_PrescaleSelect(TMR_ID_1,
                           TMR_PRESCALE_VALUE_8);
    /* Enable 16 bit mode */
    PLIB_TMR_Model16BitEnable(TMR_ID_1);
    /* Clear counter */
    PLIB_TMR_Counter16BitClear(TMR_ID_1);
    /*Set period */
    PLIB_TMR_Period16BitSet(TMR_ID_1, 10000);

    /* Setup Interrupt */
    PLIB_INT_SourceEnable(INT_ID_0, INT_SOURCE_TIMER_1);
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T1,
                           INT_PRIORITY_LEVEL3);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T1,
                           INT_SUBPRIORITY_LEVEL0);
}
```

¶ Comme la fonction stoppe le timer mais ne le démarre pas, il sera nécessaire d'utiliser la fonction `DRV_TMR0_Start`.

```
bool DRV_TMR0_Start(void)
{
    /* Start Timer*/
    DRV_TMR0_Resume(true);
    DRV_TMR0_Running = true;

    return true;
}
```

```
static void _DRV_TMR0_Resume(bool resume)
{
    if (resume)
    {
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                  INT_SOURCE_TIMER_1);
        PLIB_INT_SourceEnable(INT_ID_0,
                              INT_SOURCE_TIMER_1);
        PLIB_TMR_Start(TMR_ID_1);
    }
}
```

La fonction \_DRV\_TMR0\_Resume autorise la source d'interruption et finalement start le timer.

¶ Le détail de la configuration des timers et des interruptions sera traité dans le chapitre 5.

#### 4.3.4.2.6. La fonction APP\_Initialize

La fonction APP\_Initialize est implémentée dans le fichier app.c, son rôle est d'initialiser l'application et sa machine d'état.

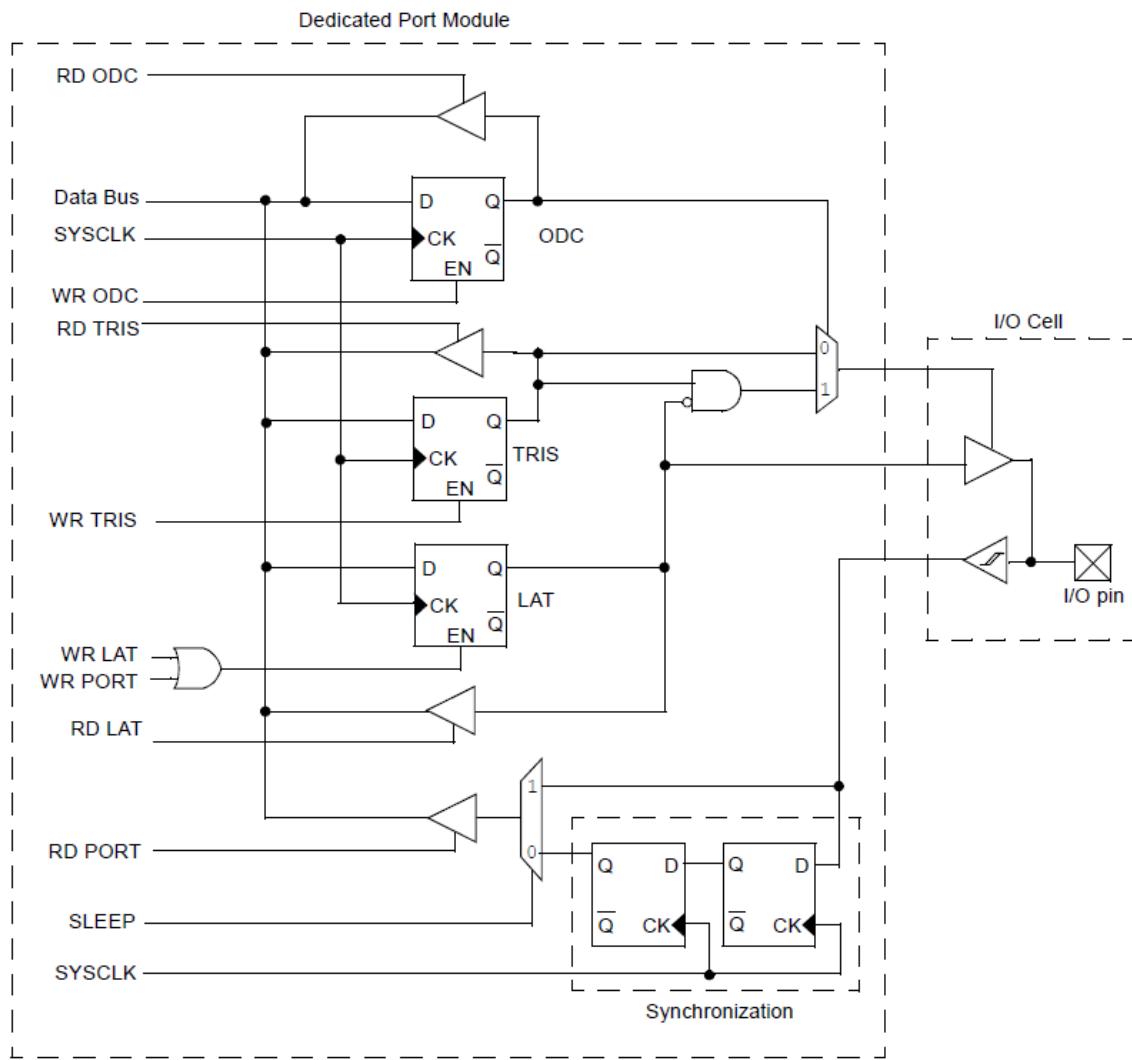
Un exemple d'application est présenté dans la dernière section de ce document.

## 4.4. CONCEPT DES E/S DIGITALES

Sur le PIC32MX, un port E/S comporte 16 lignes. Le PIC32MX795FL512L dispose de 7 ports A, B; C; D; E; F et G. Cependant tous les ports ne proposent pas 16 lignes, cela dépendant du nombre total de broches.

### 4.4.1. SCHÉMA D'UNE LIGNE D'UN PORT

Pour bien comprendre le nom et le rôle des différents éléments il faut observer le schéma d'une ligne E/S d'un port :



On peut observer 4 groupes de bascules correspondant à :

- PORT pour la lecture directe de l'état d'une ligne (entrée).
- LAT pour établir la valeur d'une sortie en écrivant dans son latch.
- TRIS (contrôle Tri-State) pour établir la direction de la pin.
- ODC pour contrôler la configuration Open Drain

Il est possible de lire l'état des bascules LAT, TRIS et ODC.

Remarque: on retrouvera PORT, LAT, TRIS et ODC au niveau du nom des registres. Par exemple pour le port A on aura PORTA, LATA, TRISA et ODCA.

#### 4.4.2. CONFIGURATION D'UNE E/S DIGITALE

Required Settings for Digital Pin Control							
Mode or Pin Usage	Pin Type	Buffer Type	TRIS Bit	ODC Bit	CNEN Bit	CNPUE Bit <sup>(1)</sup>	AD1PCFG Bit
Input	IN	ST	1	—	—	—	1
CN	IN	ST	1	—	1	1	1
Output	OUT	CMOS	0	0	—	—	1
Open Drain	OUT	OPEN	0	1	—	—	1

Les entrées peuvent fonctionner en entrée simple, ou en mode CN (Change Notification). Le mode CN correspond à utiliser l'entrée comme source d'interruption.

Pour les sorties digitales, il est possible de les configurer en sortie standard push-pull ou en drain ouvert, ce qui permet de s'adapter à un périphérique 5V par exemple.

#### 4.4.3. CARACTÉRISTIQUES ÉLECTRIQUES DES E/S

##### Absolute Maximum Ratings<sup>(1)</sup>

Ambient temperature under bias.....	-40°C to +85°C
Storage temperature .....	-65°C to +150°C
Voltage on VDD with respect to Vss .....	-0.3V to +4.0V
Voltage on any pin that is not 5V tolerant, with respect to Vss ( <b>Note 3</b> ).....	-0.3V to (VDD + 0.3V)
Voltage on any 5V tolerant pin with respect to Vss when VDD ≥ 2.3V ( <b>Note 3</b> ).....	-0.3V to +5.5V
Voltage on any 5V tolerant pin with respect to Vss when VDD < 2.3V ( <b>Note 3</b> ).....	-0.3V to +3.6V
Voltage on VBUS with respect to Vss .....	-0.3V to +5.5V
Voltage on VCORE with respect to Vss .....	-0.3V to 2.0V
Maximum current out of Vss pin(s).....	300 mA
Maximum current into VDD pin(s) ( <b>Note 2</b> ).....	300 mA
Maximum output current sunk by any I/O pin.....	25 mA
Maximum output current sourced by any I/O pin .....	25 mA
Maximum current sunk by all ports .....	200 mA
Maximum current sourced by all ports ( <b>Note 2</b> ).....	200 mA

On en conclut qu'il est possible de fournir ou d'absorber 25 mA pour chaque ligne d'E/S configurée en sortie.

⚠ Le courant total fourni ou absorbé ne doit pas dépasser 200 mA pour l'ensemble des ports.

## 4.5. LES FONCTIONS DE PLIB\_PORTS.H

C'est le *framework Harmony* qui fournit la librairie plib\_ports.h. Cette librairie fournit un set de fonctions et de macros permettant de configurer, de lire ou d'écrire sur les entrées-sorties.

¶ Les fonctions fournies sont principalement basées sur 2 paramètres : l'ID du port et l'ID du bit, ce qui ne permet pas une définition générale. D'où le choix de garder la possibilité d'accès directs dans le BSP.

Les actions directes conviennent bien et permettent une personnalisation des noms des E/S. On aura recours aux fonctions et macros pour certains cas de configuration ou d'action particulière.

Ces fonctions sont réparties en 4 groupes. En ce qui concerne le groupe Ports Function Remap, il ne s'applique pas à tous les modèles de PIC32MX. En particulier, le PIC32MX795F512L n'en dispose. Dans la documentation, cette fonctionnalité est appelée PPS (Peripheral Pin Select). Dans ce qui suit, nous traiterons principalement de la section Ports Control.

- [Ports Control](#)
- [Ports Function Remap](#)
- [Ports Change Notification](#)
- [Special Considerations](#)

### 4.5.1. VUES D'ENSEMBLE DES FONCTIONS DE GESTION D'UN PORT

#### b) Port Functions

	Name	Description
≡	<a href="#">PLIB_PORTS_Clear</a>	Clears the selected digital port/latch bits.
≡	<a href="#">PLIB_PORTS_DirectionGet</a>	Reads the direction of the selected digital port.
≡	<a href="#">PLIB_PORTS_DirectionInputSet</a>	Makes the selected pins direction input
≡	<a href="#">PLIB_PORTS_DirectionOutputSet</a>	Makes the selected pins direction output.
≡	<a href="#">PLIB_PORTS_PinOpenDrainDisable</a>	Disables the open drain functionality for the selected pin.
≡	<a href="#">PLIB_PORTS_PinOpenDrainEnable</a>	Enables the open drain functionality for the selected pin.
≡	<a href="#">PLIB_PORTS_Read</a>	Reads the selected digital port.
≡	<a href="#">PLIB_PORTS_Set</a>	Sets the selected bits of the Port
≡	<a href="#">PLIB_PORTS_Toggle</a>	Toggles the selected digital port/latch.
≡	<a href="#">PLIB_PORTS_Write</a>	Writes the selected digital port/latch.
≡	<a href="#">PLIB_PORTS_OpenDrainDisable</a>	Disables the open drain functionality for the selected port.
≡	<a href="#">PLIB_PORTS_OpenDrainEnable</a>	Enables the open drain functionality for the selected port pins.
≡	<a href="#">PLIB_PORTS_ReadLatched</a>	Reads/Gets data from the selected Latch.
≡	<a href="#">PLIB_PORTS_ChannelModeSelect</a>	Enables the selected channel pins as analog or digital.

#### 4.5.2. VUE D'ENSEMBLE DES FONCTIONS DE GESTION D'UN BIT D'UN PORT

##### a) Port Pin Functions

	Name	Description
•	PLIB_PORTS_PinClear	Clears the selected digital pin/latch.
•	PLIB_PORTS_PinDirectionInputSet	Makes the selected pin direction input
•	PLIB_PORTS_PinDirectionOutputSet	Makes the selected pin direction output
•	PLIB_PORTS_PinGet	Reads/Gets data from the selected digital pin.
•	PLIB_PORTS_PinModeSelect	Enables the selected pin as analog or digital.
•	PLIB_PORTS_PinSet	Sets the selected digital pin/latch.
•	PLIB_PORTS_PinToggle	Toggles the selected digital pin/latch.
•	PLIB_PORTS_PinWrite	Writes the selected digital pin/latch.
•	PLIB_PORTS_PinModePerPortSelect	Enables the selected port pin as analog or digital.
•	PLIB_PORTS_PinGetLatched	Reads/Gets data from the selected latch.

#### 4.5.3. AUTRES FONCTIONS

Ces fonctions de configurations sont décrites dans la section

##### d) Change Notification Functions

•	PLIB_PORTS_AnPinsModeSelect	Enables the selected AN pins as analog or digital.
•	PLIB_PORTS_CnPinsDisable	Disables CN interrupt for the selected pins of a channel.
•	PLIB_PORTS_CnPinsEnable	Enables CN interrupt for the selected pins of a channel.
•	PLIB_PORTS_CnPinsPullUpDisable	Disables change notice pull-up for the selected channel pins.
•	PLIB_PORTS_CnPinsPullUpEnable	Enables change notice pull-up for the selected channel pins.

#### 4.5.4. TYPES DE DONNÉES ET CONSTANTES

Voici les définitions permettant de spécifier le choix du port, du bit et de l'action.

##### Data Types and Constants

	Name	Description
	PORTS_ANALOG_PIN	Data type defining the different Analog input pins
	PORTS_BIT_POS	Lists the constants that hold different bit positions of PORTS.
	PORTS_CHANGE_NOTICE_PIN	Data type defining the different Change Notification Pins enumeration
	PORTS_CHANNEL	Identifies the PORT Channels Supported.
	PORTS_DATA_MASK	Data type defining the PORTS data mask
	PORTS_DATA_TYPE	Data type defining the PORTS data type.
	PORTS_MODULE_ID	Identifies the PORT Modules Supported.
	PORTS_PERIPHERAL_OD	Data type defining the different Peripherals available for Open drain Configuration
	PORTS_PIN	Data type defining the different PORTS IO Pins enumeration
	PORTS_PIN_MODE	Identifies the ports pin mode
	PORTS_REMAP_FUNCTION	Data type defining the different remap function enumeration
	PORTS_REMAP_INPUT_FUNCTION	Data type defining the different remap input function enumeration.
	PORTS_REMAP_INPUT_PIN	Data type defining the different Ports I/O input pins enumeration.
	PORTS_REMAP_OUTPUT_FUNCTION	Data type defining the different remap output function enumeration.
	PORTS_REMAP_OUTPUT_PIN	Data type defining the different Ports I/O output pins enumeration.
	PORTS_REMAP_PIN	Data type defining the different remappable input/output enumeration

#### 4.5.4.1. SPÉCIFICATION D'UN PORT

Les ports sont spécifiés par le type énuméré PORTS\_CHANNEL.

```
typedef enum {
    PORT_CHANNEL_A,
    PORT_CHANNEL_B,
    PORT_CHANNEL_C,
    PORT_CHANNEL_D,
    PORT_CHANNEL_E,
    PORT_CHANNEL_F,
    PORT_CHANNEL_G,
    PORT_CHANNEL_H,
    PORT_CHANNEL_J,
    PORT_NUMBER_OF_CHANNELS
} PORTS_CHANNEL;
```

Et ceci jusque à G pour le PIC32MX795F512L.

#### 4.5.4.2. SPÉCIFICATION DE LA POSITION D'UN BIT

La position d'un bit d'un port est spécifiée par le type énuméré PORTS\_BIT\_POS, dont voici un aperçu de la définition :

```
typedef enum {
    PORTS_BIT_POS_0,
    PORTS_BIT_POS_1,
    PORTS_BIT_POS_2,
    PORTS_BIT_POS_3,
    PORTS_BIT_POS_13,
    PORTS_BIT_POS_14,
    PORTS_BIT_POS_15
} PORTS_BIT_POS;
```

#### 4.5.4.3. SPÉCIFICATION D'UNE BROCHE (PIN) D'UN PORT

La sélection d'une ligne d'un port est spécifiée par le type énuméré PORTS\_PIN., dont voici un aperçu de la définition :

```
typedef enum {
    PORTS_PIN_0,
    PORTS_PIN_1,
    PORTS_PIN_2,
    PORTS_PIN_3,
    PORTS_PIN_13,
    PORTS_PIN_14,
    PORTS_PIN_15
} PORTS_PIN;
```

#### 4.5.4.4. SPÉCIFICATION DU MODE D'UNE BROCHE (PIN) D'UN PORT

Le type énuméré PORTS\_PIN\_MODE permet de spécifier si une broche est en mode digital ou analogique.

```
typedef enum {
    PORTS_PIN_MODE_ANALOG,
    PORTS_PIN_MODE_DIGITAL
} PORTS_PIN_MODE;
```

#### 4.5.5. FONCTIONS DE CONFIGURATION GÉNÉRALE

Dans la fonction SYS\_PORTS\_Initialize, on trouve un 1<sup>er</sup> groupe de fonctions.

```
/* AN and CN Pins Initialization */
PLIB_PORTS_An PinsModeSelect(PORTS_ID_0,
                               SYS_PORT_AD1PCFG, PORTS_PIN_MODE_DIGITAL);
PLIB_PORTS_Cn PinsPullUpEnable(PORTS_ID_0,
                               SYS_PORT_CNPUE);
PLIB_PORTS_Cn PinsEnable(PORTS_ID_0, SYS_PORT_CNEN);
PLIB_PORTS_ChangeNoticeEnable(PORTS_ID_0);
```

##### 4.5.5.1.1. La fonction PLIB\_PORTS\_An PinsModeSelect

La fonction PLIB\_PORTS\_An PinsModeSelect permet d'établir si une pin est utilisée en digital ou en analogique (pour les pins ayant une fonction analogique).

```
PLIB_PORTS_An PinsModeSelect(PORTS_ID_0,
                               SYS_PORT_AD1PCFG, PORTS_PIN_MODE_DIGITAL);
```

Avec :

```
#define SYS_PORT_AD1PCFG ~0x3ac3
```

En supposant que les bits à 1 effectuent la sélection

0x3ac3	→	0011'1010'1100'0011
~0x3ac3	→	1100'0101'0011'1100

Ce qui configure en digital AN2, AN3, AN4, AN5, AN8, AN10, AN14 et AN15.

##### 4.5.5.1.2. La fonction PLIB\_PORTS\_Cn PinsPullUpEnable

La fonction PLIB\_PORTS\_Cn PinsPullUpEnable permet d'activer les "change notice pullup". L'appel présent dans SYS\_PORTS\_Initialize n'effectuant aucune configuration, on se reportera à l'exemple de la documentation Harmony :

```
// Enable pull-up for CN5, CN8 and CN13 pins
PLIB_PORTS_Cn PinsPullUpEnable(PORTS_ID_0, CHANGE_NOTICE_PIN_5 |
                               CHANGE_NOTICE_PIN_8 |
                               CHANGE_NOTICE_PIN_13);
```

##### 4.5.5.1.3. La fonction PLIB\_PORTS\_Cn PinsEnable

La fonction PLIB\_PORTS\_Cn PinsEnable permet d'activer le "CN interrupt" pour les broches sélectionnées. L'appel présent dans SYS\_PORTS\_Initialize n'effectuant aucune configuration, on se reportera à l'exemple de la documentation Harmony :

```
// Enable CN interrupt for CN5, CN8 and CN13 pins
PLIB_PORTS_Cn PinsEnable(PORTS_ID_0,
                        CHANGE_NOTICE_PIN_5 |
                        CHANGE_NOTICE_PIN_8 |
                        CHANGE_NOTICE_PIN_13);
```

##### 4.5.5.1.4. La fonction PLIB\_PORTS\_ChangeNoticeEnable

La fonction PLIB\_PORTS\_ChangeNoticeEnable permet d'activer "the global Change Notice feature".

```
PLIB_PORTS_ChangeNoticeEnable(PORTS_ID_0);
```

#### 4.5.6. FONCTIONS DE CONFIGURATION D'UN PORT

On dispose des fonctions suivantes pour la configuration de la direction et du mode des broches.

```
void PLIB_PORTS_DirectionOutputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_DATA_MASK mask);

void PLIB_PORTS_DirectionInputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK
mask);

void PLIB_PORTS_PinDirectionOutputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos);

void PLIB_PORTS_PinDirectionInputSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos);

void PLIB_PORTS_PinModeSelect(PORTS_MODULE_ID index, PORTS_ANALOG_PIN pin, PORTS_PIN_MODE mode);

void PLIB_PORTS_PinModePerPortSelect(PORTS_MODULE_ID index, PORTS_CHANNEL channel,
PORTS_BIT_POS bitPos, PORTS_PIN_MODE mode);

void PLIB_PORTS_OpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK
mask);

void PLIB_PORTS_OpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK
mask);

void PLIB_PORTS_PinOpenDrainDisable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);

void PLIB_PORTS_PinOpenDrainEnable(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS
bitPos);
```

##### 4.5.6.1. INITIALISATION DANS SYS\_PORTS\_INITIALIZE : PORT A

Voici l'exemple de configuration du port A dans la fonction SYS\_PORTS\_Initialize :

```
/* PORT A Initialization */
PLIB_PORTS_OpenDrainEnable(PORTS_ID_0, PORT_CHANNEL_A,
                           SYS_PORT_A_ODC);
PLIB_PORTS_Write(PORTS_ID_0, PORT_CHANNEL_A,
                  SYS_PORT_A_LAT);
PLIB_PORTS_DirectionOutputSet(PORTS_ID_0,
                             PORT_CHANNEL_A, SYS_PORT_A_TRIS ^ 0xFFFF);
```

##### 4.5.6.2. ETABLISSEMENT DE LA DIRECTION D'UN GROUPE DE BROCHES D'UN PORT

Voici des exemples pour établir en sortie ou en entrée les broches d'un port.

```
// Etablit en entrée les bits 7, 6, 5, 4 du port D
PLIB_PORTS_DirectionInputSet( PORTS_ID_0, PORT_CHANNEL_D,
0x00F0 );
```

Les bits à "1" du masque sélectionnent les broches du port spécifié.

```
PLIB_PORTS_DirectionOutputSet( PORTS_ID_0,
PORT_CHANNEL_A, SYS_PORT_A_TRIS ^ 0xFFFF);
```

Avec #define SYS\_PORT\_A\_TRIS 0x460C on obtient :

0x460C	0100'0110'0000'1100
0x460C ^ 0xFFFF	1011'1001'1111'0011

Etablissement en sortie pour les bits à 0 dans la valeur de SYS\_PORT\_A\_TRIS. Cela correspond à la polarité des registres TRISx .

#### 4.5.6.3. ETABLISSEMENT DE LA DIRECTION D'UNE BROCHE D'UN PORT

Voici des exemples pour établir en sortie ou en entrée une broche d'un port.

```
// Etablit en sortie la broche RC4
PLIB_PORTS_PinDirectionOutputSet( PORTS_ID_0,
                                    PORT_CHANNEL_C, PORTS_BIT_POS_4 );

// Etablit en entrée la broche RC5
PLIB_PORTS_PinDirectionInputSet( PORTS_ID_0,
                                  PORT_CHANNEL_C, PORTS_BIT_POS_5 );

En utilisant les définitions introduites dans BSP_config.h, par exemple pour la touche OK,
on a :
PLIB_PORTS_PinDirectionInputSet( PORTS_ID_0,
                                  S_OK_PORT, S_OK_BIT);
```

#### 4.5.6.4. ETABLISSEMENT DU MODE D'UNE ENTRÉE ANALOGIQUE

Voici un exemple pour établir en mode analogique AN0 et AN1 en utilisant deux fois la fonction.

```
// CHR config AN0 et AN1 en Analogique
PLIB_PORTS_PinModeSelect(PORTS_ID_0, PORTS_ANALOG_PIN_0,
                         PORTS_PIN_MODE_ANALOG);
PLIB_PORTS_PinModeSelect(PORTS_ID_0, PORTS_ANALOG_PIN_1,
                         PORTS_PIN_MODE_ANALOG);
```

On remarque l'utilisation du type énuméré PORTS\_ANALOG\_PIN dont voici un extrait.

```
typedef enum {
    PORTS_ANALOG_PIN_0,
    PORTS_ANALOG_PIN_1,
    PORTS_ANALOG_PIN_2,
} PORTS_ANALOG_PIN;
```

Remarque : Dans le cas du PIC32MX795F512L, on dispose de 16 entrées analogiques AN0-AN15 correspondant respectivement à RB0-RB15.

Dans la fonction BSP\_Initialize, on utilise une autre fonction pour mettre explicitement les pins voulues en analogique, et les autres en digital :

#### 4.5.6.5. GESTION OPEN DRAIN D'UN GROUPE DE BROCHES D'UN PORT

Voici un exemple pour activer l'open drain sur les broches d'un port :

```
// Enable Open Drain des broches 3, 2, 1, 0 du port D
PLIB_PORTS_OpenDrainEnable( PORTS_ID_0, PORT_CHANNEL_D,
                            0x000F );
```

Les bits à "1" du masque sélectionnent les broches du port spécifié.

La fonction PLIB\_PORTS\_OpenDrainDisable procède à la désactivation selon le même principe.

#### 4.5.6.6. GESTION OPEN DRAIN D'UNE BROCHE D'UN PORT

Voici deux exemples pour activer l'open drain sur une une broche d'un port.

```
// Enable Open Drain pour la broche RC4
PLIB_PORTS_PinOpenDrainEnable( PORTS_ID_0,
                                PORT_CHANNEL_C, PORTS_BIT_POS_4 );
```

La fonction PLIB\_PORTS\_PinOpenDrainDisable procède à la désactivation selon le même principe.

#### 4.5.7. FONCTIONS DE LECTURE DES ENTRÉES

On dispose d'un certain nombre de fonctions pour la lecture des entrées.

Traitement au niveau du port :

PLIB_PORTS_Read	: Lecture via registre PORT
PLIB_PORTS_ReadLatched	: Lecture via registre LAT

Ces fonctions retournent une valeur du type PORTS\_DATA\_TYPE défini de la manière suivante :

```
typedef uint32_t PORTS_DATA_TYPE;
```

Traitement au niveau du bit :

PLIB_PORTS_PinGet	: Lecture via registre PORT
PLIB_PORTS_PinGetLatched	: Lecture via registre LAT

##### 4.5.7.1. FONCTION PLIB\_PORTS\_READ

Cette fonction fournit la valeur du port spécifié. Voici son prototype :

```
PORTS_DATA_TYPE PLIB_PORTS_Read(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Les 16 bits du port sont retournés dans une variable 32 bits.

Exemple :

```
ValPortC = PLIB_PORTS_Read(PORTS_ID_0, PORT_CHANNEL_C);
```

##### 4.5.7.2. FONCTION PLIB\_PORTS\_READLATCHED (LECTURE ÉTATS SORTIES)

Cette fonction fournit la valeur du latch du port spécifié, ce qui permet d'obtenir l'état des broches d'un port configuré en sortie. Voici son prototype :

```
PORTS_DATA_TYPE PLIB_PORTS_ReadLatched(PORTS_MODULE_ID index, PORTS_CHANNEL channel);
```

Les 16 bits du latch du port sont retournés dans une variable 32 bits.

Exemple :

```
ValLatchA = PLIB_PORTS_ReadLatched(PORTS_ID_0,
```

```
                                PORT_CHANNEL_A);
```

#### 4.5.7.3. FONCTION PLIB\_PORTS\_PinGet

Cette fonction fournit la valeur du bit du port spécifié. Voici son prototype :

```
bool PLIB_PORTS_PinGet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Exemple avec bool ToucheOK (S\_OK correspond à RG12) :

```
ToucheOK = PLIB_PORTS_PinGet(PORTS_ID_0, PORT_CHANNEL_G,
                             PORTS_BIT_POS_12);
```

A l'aide des définitions ajoutées dans le BSP, on écrira plus avantageusement :

```
ToucheOK = PLIB_PORTS_PinGet(PORTS_ID_0,
                             S_OK_PORT, S_OK_BIT);
```

#### 4.5.7.4. FONCTION PLIB\_PORTS\_PinGetLatched (LECTURE SORTIE)

Cette fonction fournit la valeur mémorisée du bit spécifié du port spécifié. **Elle permet de lire l'état d'une sortie.** Voici son prototype :

```
bool PLIB_PORTS_PinGetLatched(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Voici un exemple avec la lecture des deux leds qui clignotent et de lecture de la touche OK qui correspond à RG12. Exemple avec bool ToucheOK et en utilisant les définitions du BSP :

```
bool ToucheOK;
bool EtatLed5, EtatLed6;

EstatLed5 = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                      LED5_PORT, LED5_BIT);
EstatLed6 = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                      LED6_PORT, LED6_BIT);
ToucheOK = PLIB_PORTS_PinGet(PORTS_ID_0,
                            S_OK_PORT, S_OK_BIT);

lcd_gotoxy(1,3);
printf_lcd("led5 %1d led6 %1d OK = %1d",
          EstatLed5, EstatLed6, ToucheOK);
```

On obtient l'état de la touche ainsi que le changement de l'état des leds conformément au clignotement.

#### 4.5.8. FONCTIONS D'ÉCRITURES DES SORTIES

On dispose d'un certain nombre de fonctions pour l'écriture des sorties.

Traitement au niveau du port :

**PLIB\_PORTS\_Clear**  
**PLIB\_PORTS\_Set**  
**PLIB\_PORTS\_Write**  
**PLIB\_PORTS\_Toggle**

Ces fonctions utilisent pour la valeur un paramètre du type PORTS\_DATA\_TYPE défini de la manière suivante :

```
typedef uint32_t PORTS_DATA_TYPE;
```

Ces fonctions utilisent pour le masque de sélection des sorties un paramètre du type PORTS\_DATA\_MASK défini de la manière suivante :

```
typedef uint16_t PORTS_DATA_MASK;
```

Traitement au niveau du bit :

```
PLIB_PORTS_PinClear
PLIB_PORTS_PinSet
PLIB_PORTS_PinWrite
PLIB_PORTS_PinToggle
```

#### 4.5.8.1. FONCTION PLIB\_PORTS\_CLEAR

Cette fonction met au niveau bas le groupe de sorties spécifiées sur le port spécifié. Voici son prototype :

```
void PLIB_PORTS_Clear(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK clearMask);
```

Exemple :

```
PLIB_PORTS_Clear(PORTS_ID_0, PORT_CHANNEL_A, 0x80C0);
```

Etablit à '0' les bits 15, 7, 6, du port A, ce qui allume les leds 4,5,6.

#### 4.5.8.2. FONCTION PLIB\_PORTS\_SET

Cette fonction effectue un ET bit à bit entre le paramètre **value** et le paramètre **mask**, les bits à "1" qui en résultent s'appliquent aux sorties du port spécifié. Voici son prototype :

```
void PLIB_PORTS_Set(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_TYPE value,
PORTS_DATA_MASK mask);
```

Exemple :

```
PLIB_PORTS_Set(PORTS_ID_0, PORT_CHANNEL_A, 0xFFFF, 0x80F3);
```

Etablit à '1' les bits 15, 7, 6, 5, 4, 1 et 0 du port A, ce qui éteint les leds 6 à 0.

#### 4.5.8.3. FONCTION PLIB\_PORTS\_WRITE

Cette fonction écrit la valeur fournie en paramètre sur l'entier du port spécifié.

Il y a écriture dans le latch de sortie ; Le résultat sur les broches dépend de la direction établie. Voici son prototype :

```
void PLIB_PORTS_Write(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_TYPE value);
```

Exemple :

```
PLIB_PORTS_Write(PORTS_ID_0, PORT_CHANNEL_A, 0x80F3);
```

Etablit à '1' les bits 15, 7, 6, 5, 4, 1 et 0 du port A, ce qui éteint les leds 6 à 0. ☹ Etablit à '0' les autres bits.

Avec :

```
PLIB_PORTS_Write(PORTS_ID_0, PORT_CHANNEL_A, 0);
```

Etablit tous les bits à '0', ce qui allume aussi les leds 6 à 0.

#### 4.5.8.4. FONCTION PLIB\_PORTS\_TOGGLE

Cette fonction écrit la valeur fournie en paramètre sur l'entier du port spécifié.

Il y a écriture dans le latch de sortie ; Le résultat sur les broches dépend de la direction établie. Voici son prototype :

```
void PLIB_PORTS_Toggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_DATA_MASK toggleMask);
```

Exemple :

```
PLIB_PORTS_Toggle(PORTS_ID_0, PORT_CHANNEL_A, 0x80F3);
```

Etablit à '0' RA15, ce qui éteint la led 6.

#### 4.5.8.5. FONCTION PLIB\_PORTS\_PINCLEAR

Cette fonction met à '0' le bit spécifié sur le port spécifié. Voici son prototype :

```
void PLIB_PORTS_PinClear(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Exemple :

```
PLIB_PORTS_PinClear(PORTS_ID_0,
                     PORT_CHANNEL_A, PORTS_BIT_POS_15);
```

Etablit à '0' RA15, ce qui éteint la led 6.

En utilisant les définitions du BSP on obtient :

```
PLIB_PORTS_PinClear(PORTS_ID_0, LED6_PORT, LED6_BIT);
```

#### 4.5.8.6. FONCTION PLIB\_PORTS\_PINSET

Cette fonction met à '1' le bit spécifié sur le port spécifié. Voici son prototype :

```
void PLIB_PORTS_PinSet(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Exemple :

```
PLIB_PORTS_PinSet(PORTS_ID_0,
                   PORT_CHANNEL_A, PORTS_BIT_POS_15);
```

Etablit à '1' RA15, ce qui allume la led 6.

En utilisant les définitions du BSP on obtient :

```
PLIB_PORTS_PinSet(PORTS_ID_0, LED6_PORT, LED6_BIT);
```

#### 4.5.8.7. FONCTION PLIB\_PORTS\_PINWRITE

Cette fonction établit à l'état spécifié le bit spécifié sur le port spécifié. Voici son prototype :

```
void PLIB_PORTS_PinWrite(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos,
                         bool value);
```

Exemple1 :

```
PLIB_PORTS_PinWrite(PORTS_ID_0, PORT_CHANNEL_A,
                     PORTS_BIT_POS_15, 1);
```

Etablit à '1' RA15, ce qui allume la led 6.

En utilisant les définitions du BSP on obtient :

```
PLIB_PORTS_PinWrite(PORTS_ID_0, LED6_PORT, LED6_BIT, 1);
```

Exemple2, inversion avec lecture de l'état de la led5 et action Write :

```
bool EtatLed5;
EstatLed5 = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                         LED5_PORT, LED5_BIT);
EstatLed5 = !EstatLed5;
PLIB_PORTS_PinWrite(PORTS_ID_0, LED5_PORT,
                     LED5_BIT, EstatLed5);
```

Ø On remarque l'utilisation de la fonction **PinGetLatched** pour connaitre l'état de la broche correspondant à LED5.

#### 4.5.8.8. FONCTION PLIB\_PORTS\_PINTOGGLE

Cette fonction inverse (toggle) l'état du bit spécifié sur le port spécifié. Voici son prototype :

```
void PLIB_PORTS_PinToggle(PORTS_MODULE_ID index, PORTS_CHANNEL channel, PORTS_BIT_POS bitPos);
```

Exemple :

```
PLIB_PORTS_PinToggle(PORTS_ID_0,
                      PORT_CHANNEL_A, PORTS_BIT_POS_7);
```

Toggle RA7, ce qui inverse l'état de la led 5.

En utilisant les définitions du BSP on obtient :

```
PLIB_PORTS_PinToggle (PORTS_ID_0, LED5_PORT, LED5_BIT);
```

#### 4.5.8.9. RÉALISATION DU PINTOGGLE, ACCÈS DIRECT

A titre de comparaison, les définitions directes des bits d'un port permettent une écriture plus compacte.

```
// Toggle led4 par inversion état, accès direct
LED4_W = !LED4_R;
```

**Au niveau mécanisme d'accès, il faut toujours lire PORT et écrire dans LAT.**

Rappel des définitions :

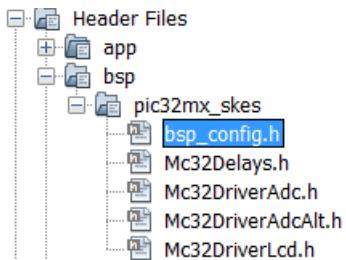
#define LED4_W	LATABits.LATA6	// Led4 écriture
#define LED4_R	PORTAbits.RA6	// Led4 lecture

## 4.6. BSP PIC32MX\_SKES

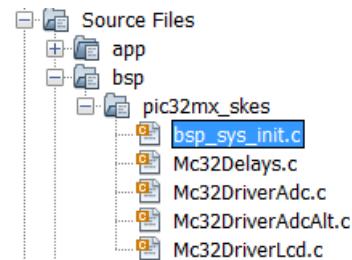
Le BSP (Board Support Package) spécifique au starter-kit ES a été réalisé en modifiant les fichiers d'un BSP microchip et sur la base du starter-kit version B.

Les définitions et les prototypes des fonctions sont dans le fichier `BSP_config.h` tandis que l'implémentation des fonctions est dans le fichier `BSP_sys_init.c`.

Localisation du fichier `BSP_config.h`



Localisation du fichier `BSP_sys_init.c`



### 4.6.1. CONTENU DU FICHIER `BSP_CONFIG.H`

Ce fichier contient les redéfinitions pour le kit des éléments d'entrées-sorties pour réaliser un accès direct.

En plus, pour permettre l'utilisation des fonctions de traitement au niveau bit énumérées ci-dessous, les définitions de PORT et BIT\_POS ont été ajoutées.

<code>PLIB_PORTS_PinClear</code>	<code>PLIB_PORTS_PinToggle</code>
<code>PLIB_PORTS_PinSet</code>	<code>PLIB_PORTS_PinGet</code>
<code>PLIB_PORTS_PinWrite</code>	<code>PLIB_PORTS_PinGetLatched</code>

Voici quelques extraits du fichier `bsp_config.h`, avec en commentaires le contenu correspondant du fichier `bsp.xml`.

Le fichier `bsp.xml` contient par exemple :

- Les noms des signaux rattachés aux différentes pins. Ces derniers seront affichés dans le pin diagram et pin settings du MHC.
- Les configurations par défaut des pins, de manière à générer les constantes d'initialisation adéquates.

```

// Section: Included Files
// ****
#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include "peripheral/ports/plib_ports.h"


// ****
// Oscillator Frequency
// Description:
//     Defines frequency value of crystal/oscillator
//     used on the board
#define BSP_OSC_FREQUENCY 8000000 // 8 MHz
  
```

```

/*-----*/
// Analogique
/*-----*/
// Uniquement pour Info
#define Analog0      PORTBbits.RB0
#define Analog1      PORTBbits.RB1

// Definitions dans le fichier XML (BSP.xml)
// <!-- Analog IN -->
// <function name="POT0" pin="RB0" mode="analog" />
// <function name="POT1" pin="RB1" mode="analog" />

/*-----*/
// Touches
/*-----*/
// Definitions directes
#define S_OK          PORTGbits.RG12
#define S_ESC_MENU    PORTGbits.RG13
#define S_PLUS         PORTGbits.RG14
#define S_MINUS        PORTGbits.RG15

// Definitions pour fonctions PLIB_PORTS
#define S_OK_PORT     PORT_CHANNEL_G
#define S_OK_BIT       PORTS_BIT_POS_12
#define S_ESC_MENU_PORT PORT_CHANNEL_G
#define S_ESC_MENU_BIT PORTS_BIT_POS_13
#define S_PLUS_PORT   PORT_CHANNEL_G
#define S_PLUS_BIT    PORTS_BIT_POS_14
#define S_MINUS_PORT  PORT_CHANNEL_G
#define S_MINUS_BIT   PORTS_BIT_POS_15

// Definitions dans le fichier XML (BSP.xml)
// <function name="S_OK" pin="RG12" mode="digital"
//                                         pullup="true"/>
// <function name="S_ESC_MENU" pin="RG13" mode="digital"
//                                         pullup="true"/>
// <function name="S_PLUS" pin="RG14" mode="digital"
//                                         pullup="true"/>
// <function name="S_MINUS" pin="RG15" mode="digital"
//                                         pullup="true"/>

/*-----*/
// LEDs
/*-----*/

// Attention 11020_B modification du câblage
// -----
// Led0  RA0   D6
// Led1  RA1   D10
// Led2  RA4   D7
// Led3  RA5   D11
// Led4  RA6   D8
// Led5  RA7   D12
// Led6  RA15  D9
// Led7  RB10  D13  !!! port B

```

```

// On écrit dans le latch pour éviter les problèmes de R/W
#define LED0_W LATAbits.LATA0 //Led0
#define LED1_W LATAbits.LATA1 //Led1
#define LED2_W LATAbits.LATA4 //Led2
#define LED3_W LATAbits.LATA5 //Led3
#define LED4_W LATAbits.LATA6 //Led4
#define LED5_W LATAbits.LATA7 //Led5
#define LED6_W LATAbits.LATA15 //Led6
#define LED7_W LATBbits.LATB10 //Led7
// On lit directement sur le port, sinon on obtient la
// valeur précédemment écrite dans le latch!!
#define LED0_R PORTAbits.RA0 //Led0
#define LED1_R PORTAbits.RA1 //Led1
#define LED2_R PORTAbits.RA4 //Led2
#define LED3_R PORTAbits.RA5 //Led3
#define LED4_R PORTAbits.RA6 //Led4
#define LED5_R PORTAbits.RA7 //Led5
#define LED6_R PORTAbits.RA15 //Led6
#define LED7_R PORTBbits.RB10 //Led7

#define LED0_T TRISAbits.TRISA0
#define LED1_T TRISAbits.TRISA1
#define LED2_T TRISAbits.TRISA4
#define LED3_T TRISAbits.TRISA5
#define LED4_T TRISAbits.TRISA6
#define LED5_T TRISAbits.TRISA7
#define LED6_T TRISAbits.TRISA15
#define LED7_T TRISBbits.TRISB10

// Definitions pour fonction PLIB_PORTS
#define LED0_PORT PORT_CHANNEL_A
#define LED0_BIT PORTS_BIT_POS_0
#define LED1_PORT PORT_CHANNEL_A
#define LED1_BIT PORTS_BIT_POS_1
#define LED2_PORT PORT_CHANNEL_A
#define LED2_BIT PORTS_BIT_POS_4
#define LED4_PORT PORT_CHANNEL_A
#define LED4_BIT PORTS_BIT_POS_6
#define LED5_PORT PORT_CHANNEL_A
#define LED5_BIT PORTS_BIT_POS_7
#define LED6_PORT PORT_CHANNEL_A
#define LED6_BIT PORTS_BIT_POS_15
#define LED7_PORT PORT_CHANNEL_B
#define LED7_BIT PORTS_BIT_POS_10

```

```

// Definitions dans le fichier XML (BSP.xml)
// Avec essais effet latch low ou High
// <!-- LEDS -->
// <function name="LED_0" pin="RA0" mode="digital"
//           direction="out" latch="low"/>
// <function name="LED_1" pin="RA1" mode="digital"
//           direction="out" latch="low"/>
// <function name="LED_2" pin="RA4" mode="digital"
//           direction="out" latch="low"/>
// <function name="LED_3" pin="RA5" mode="digital"
//           direction="out" latch="low"/>
// <function name="LED_4" pin="RA6" mode="digital"
//           direction="out" latch="high"/>
// <function name="LED_5" pin="RA7" mode="digital"
//           direction="out" latch="high"/>
// <function name="LED_6" pin="RA15" mode="digital"
//           direction="out" latch="high"/>
// <function name="LED_7" pin="RB10" mode="digital"
//           direction="out" latch="high"/>

```

On constate que pour les leds, on a réalisé 3 groupes de définitions, pour l'écriture, la lecture et la configuration de la direction.

#### 4.6.2. CONTENU DU FICHIER BSP\_SYS\_INIT.C

Le fichier BSP\_sys\_init.c contient la fonction **BSP\_Initialize**, ainsi que des fonctions pour agir sur les leds et les switch du BSP (fonctions compatibles avec les exemples Microchip).

##### 4.6.2.1. EXTRAIT FONCTION BSP\_INITIALIZE

Voici un extrait de la nouvelle fonction **BSP\_Initialize**. Dû à l'évolution des versions, les actions directes sont en commentaires. Des appels aux fonctions PLIB\_PORTS sont réalisés à la place. De plus, une grande partie de la configuration par défaut est réalisée dans via le fichier bsp.xml.

```

void BSP_Initialize(void )
{
    // Pour ne pas entrer en conflit avec le JTAG
    SYS_DEVCON_JTAGDisable(); // déjà fait mais si on oublie

    /*-----*/
    // Analogique
    /*-----*/
    /*
        TRISBbits.TRISB0 = 1; //Analog0 en entrée
        TRISBbits.TRISB1 = 1; //Analog1 en entrée
    */
}

```

```

// Config AN0, AN1 en Analogique, les autres en digital
PLIB_PORTS_AnPinsModeSelect(PORTS_ID_0, 0x0003,
                             PORTS_PIN_MODE_ANALOG);
PLIB_PORTS_AnPinsModeSelect(PORTS_ID_0, ~0x0003,
                             PORTS_PIN_MODE_DIGITAL);

/*
    PLIB_PORTS_PinModeSelect(PORTS_ID_0,
                           PORTS_ANALOG_PIN_0, PORTS_PIN_MODE_ANALOG);
    PLIB_PORTS_PinModeSelect(PORTS_ID_0,
                           PORTS_ANALOG_PIN_1, PORTS_PIN_MODE_ANALOG);
 */

/*-----*/
/* LEDs
/*-----*/

/*
LED0_T = 0; //LED_D6 (Led0) en sortie
LED0_W = 1; //LED_D6 (Led0) = 1
LED1_T = 0; //LED_D10 (Led1) en sortie
LED1_W = 1; //LED_D10 (Led1) = 1
LED2_T = 0; //LED_D7 (Led2) en sortie
LED2_W = 1; //LED_D7 (Led2) = 1
LED3_T = 0; //LED_D11 (Led3) en sortie
LED3_W = 1; //LED_D11 (Led3) = 1
LED4_T = 0; //LED_D8 (Led4) en sortie
LED4_W = 1; //LED_D8 (Led4) = 1
LED5_T = 0; //LED_D12 (Led5) en sortie
LED5_W = 1; //LED_D12 (Led5) = 1
LED6_T = 0; //LED_D9 (Led6) en sortie
LED6_W = 1; //LED_D9 (Led6) = 1
LED7_T = 0; //LED_D13 (Led7) en sortie
LED7_W = 1; //LED_D13 (Led7) = 1
*/

```

#### 4.6.3. ECRITURE ET LECTURE DIRECTE

Ces définitions permettent d'accéder directement à la valeur d'une ligne d'un port.

Par exemple pour allumer la LED5 on écrira :

```
LED5_W = 0;
```

Par exemple pour tester l'état de la touche OK on écrira :

```
if (S_OK == 0) { // si touche pressée;
```

Il est nécessaire d'inclure le fichier BSP\_config.h pour disposer des définitions.

#### 4.6.4. ECRITURE ET LECTURE AVEC LES FONCTIONS PLIB\_PORTS

En utilisant les définitions PORTS et BIT\_POS, il est possible d'utiliser les fonctions de la PLIB\_PORTS de la manière suivante :

Par exemple pour allumer la LED5 on écrira :

```
PLIB_PORTS_PinClear(PORTS_ID_0, LED5_PORT, LED5_BIT);
```

Par exemple pour tester l'état de la touche OK on écrira :

```
// si touche pressée  
if (PLIB_PORTS_PinGet(PORTS_ID_0, S_OK_PORT, S_OK_BIT) ==
```

## 4.7. GESTION DES LEDS ET SWITCH DANS LE BSP

Pour gérer les leds et les switchs du BSP pic32mx\_skes, les définitions et les fonctions ont été effectuées de la manière suivante :

### 4.7.1. DÉFINITIONS BSP DES LEDS

Voici le type énuméré adapté à la situation des 8 leds du kit ES.

```
typedef enum
{
    BSP_LED_0 = /*DOM-IGNORE-BEGIN*/ PORTS_BIT_POS_0
                    /*DOM-IGNORE-END*/,
    BSP_LED_1 = /*DOM-IGNORE-BEGIN*/ PORTS_BIT_POS_1
                    /*DOM-IGNORE-END*/,
    BSP_LED_2 = /*DOM-IGNORE-BEGIN*/ PORTS_BIT_POS_4
                    /*DOM-IGNORE-END*/,
    BSP_LED_3 = /*DOM-IGNORE-BEGIN*/ PORTS_BIT_POS_5
                    /*DOM-IGNORE-END*/,
    BSP_LED_4 = /*DOM-IGNORE-BEGIN*/ PORTS_BIT_POS_6
                    /*DOM-IGNORE-END*/,
    BSP_LED_5 = /*DOM-IGNORE-BEGIN*/ PORTS_BIT_POS_7
                    /*DOM-IGNORE-END*/,
    BSP_LED_6 = /*DOM-IGNORE-BEGIN*/ PORTS_BIT_POS_15
                    /*DOM-IGNORE-END*/,
    // Attention led7 port B
    BSP_LED_7 = /*DOM-IGNORE-BEGIN*/ PORTS_BIT_POS_10
                    /*DOM-IGNORE-END*/,
} BSP_LED;
```

### 4.7.2. LA FONCTION BSP\_LEDSTATESET

Utilisation de la fonction PLIB\_PORTS\_PinWrite pour réaliser la fonction BSP\_LEDStateSet.

```
void BSP_LEDStateSet(BSP_LED led, BSP_LED_STATE state)
{
    if (led == BSP_LED_7) {
        PLIB_PORTS_PinWrite ( PORTS_ID_0 , PORT_CHANNEL_B ,
                                led, state );
    } else {
        PLIB_PORTS_PinWrite ( PORTS_ID_0 , PORT_CHANNEL_A ,
                                led, state );
    }
}
```

#### 4.7.3. LA FONCTION **BSP\_LEDOn**

Utilisation de la fonction PLIB\_PORTS\_PinClear pour la fonction BSP\_LEDOn.

```
void BSP_LEDOn(BSP_LED led)
{
    // Led On pour Clear
    if (led == BSP_LED_7) {
        PLIB_PORTS_PinClear( PORTS_ID_0, PORT_CHANNEL_B,
                               led);
    } else {
        PLIB_PORTS_PinClear( PORTS_ID_0, PORT_CHANNEL_A,
                               led);
    }
}
```

#### 4.7.4. LA FONCTION **BSP\_LEDOff**

Utilisation de la fonction PLIB\_PORTS\_PinSet pour la fonction BSP\_LEDOff.

```
void BSP_LEDOff(BSP_LED led)
{
    // Led Off pour Set
    if (led == BSP_LED_7) {
        PLIB_PORTS_PinSet( PORTS_ID_0, PORT_CHANNEL_B,
                             led);
    } else {
        PLIB_PORTS_PinSet( PORTS_ID_0, PORT_CHANNEL_A,
                             led);
    }
}
```

#### 4.7.5. LA FONCTION **BSP\_LEDToggle**

Utilisation de la fonction PLIB\_PORTS\_PinToggle pour la fonction BSP\_LEDToggle.

```
void BSP_LEDToggle(BSP_LED led)
{
    if (led == BSP_LED_7) {
        PLIB_PORTS_PinToggle(PORTS_ID_0, PORT_CHANNEL_B,
                               led);
    } else {
        PLIB_PORTS_PinToggle(PORTS_ID_0, PORT_CHANNEL_A,
                               led );
    }
}
```

#### 4.7.6. LA FONCTION **BSP\_LEDSTATEGET**

Utilisation de la fonction **PLIB\_PORTS\_PinGetLatched** pour réaliser la fonction **BSP\_LEDStateGet**.

Le type **BSP\_LED\_STATE** est défini ainsi :

```
typedef enum
{
    /* LED State is on */
    BSP_LED_STATE_OFF = /*DOM-IGNORE-BEGIN*/ 0,
                           /*DOM-IGNORE-END*/
    /* LED State is off */
    BSP_LED_STATE_ON = /*DOM-IGNORE-BEGIN*/ 1,
                           /*DOM-IGNORE-END*/
} BSP_LED_STATE;
```

D'où la fonction :

```
BSP_LED_STATE BSP_LEDStateGet(BSP_LED led)
{
    BSP_LED_STATE tmp;
    if (led == BSP_LED_7) {
        tmp = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                         PORT_CHANNEL_B, led);
    } else {
        tmp = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                         PORT_CHANNEL_A, led);
    }
    return (tmp);
}
```

#### 4.7.7. DÉFINITIONS BSP DES SWITCHES

Voici le type énuméré adapté à la situation des 4 switches du kit ES.

```
// #define S_OK           PORTGbits.RG12   SWITCH_1
// #define S_ESC_MENU     PORTGbits.RG13   SWITCH_2
// #define S_PLUS         PORTGbits.RG14   SWITCH_3
// #define S_MINUS        PORTGbits.RG15   SWITCH_4

typedef enum
{
    /* SWITCH 1 */
    BSP_SWITCH_1 = /*DOM-IGNORE-BEGIN*/ PORTS_BIT_POS_12
                      /*DOM-IGNORE-END*/,
    /* SWITCH 2 */
    BSP_SWITCH_2 = /*DOM-IGNORE-BEGIN*/ PORTS_BIT_POS_13
                      /*DOM-IGNORE-END*/,
    /* SWITCH 3 */
    BSP_SWITCH_3 = /*DOM-IGNORE-BEGIN*/ PORTS_BIT_POS_14
                      /*DOM-IGNORE-END*/,
    /* SWITCH 4 */
    BSP_SWITCH_4 = /*DOM-IGNORE-BEGIN*/ PORTS_BIT_POS_15
                      /*DOM-IGNORE-END*/
} BSP_SWITCH;
```

#### 4.7.8. LA FONCTION **BSP\_SWITCHSTATEGET**

Cette fonction permet d'obtenir l'état d'un switch.

```
// CHR les switches sont sur le port G
BSP_SWITCH_STATE BSP_SwitchStateGet( BSP_SWITCH BSPSwitch )
{
    return ( PLIB_PORTS_PinGet(PORTS_ID_0, PORT_CHANNEL_G,
                                BSPSwitch) );
}
```

#### 4.7.9. LA FONCTION **BSP\_ENABLEHBRIGE**

Cette fonction spécifique au BSP du kit ES permet d'activer les ponts en H. Elle est réalisée à l'aide d'actions directes. Elle confirme ou modifie les directions.

```
void BSP_EnableHbridge(void)
{
    TRISBbits.TRISB8 = 0; //STBY_HBRIDGE en sortie
    STBY_HBRIDGE_W = 0; // STBY low durant init

    TRISDbits.TRISD12 = 0; //AIN1_HBRIDGE en sortie
    TRISDbits.TRISD13 = 0; //AIN2_HBRIDGE en sortie
    // Ne pas toucher PWM à cause init OC avant
    // TRISDbits.TRISD1 = 0; //PWMA_HBRIDGE en sortie
    // Mise en short brake PWM dont care
    AIN1_HBRIDGE_W = 1; //AIN1 High
    AIN2_HBRIDGE_W = 1; //AIN2 High
    // PWMA_HBRIDGE_W = 0; //PWMA low

    TRISCbits.TRISC1 = 0; //BIN1_HBRIDGE en sortie
    TRISCbits.TRISC2 = 0; //BIN2_HBRIDGE en sortie
    // Ne pas toucher PWM à cause init OC avant
    // TRISDbits.TRISD2 = 0; //PWMB_HBRIDGE en sortie

    // Mise en short brake PWM dont care
    BIN1_HBRIDGE_W = 1; //BIN1 High
    BIN2_HBRIDGE_W = 1; //BIN2 High
    // PWMB_HBRIDGE_W = 0; //PWMB low

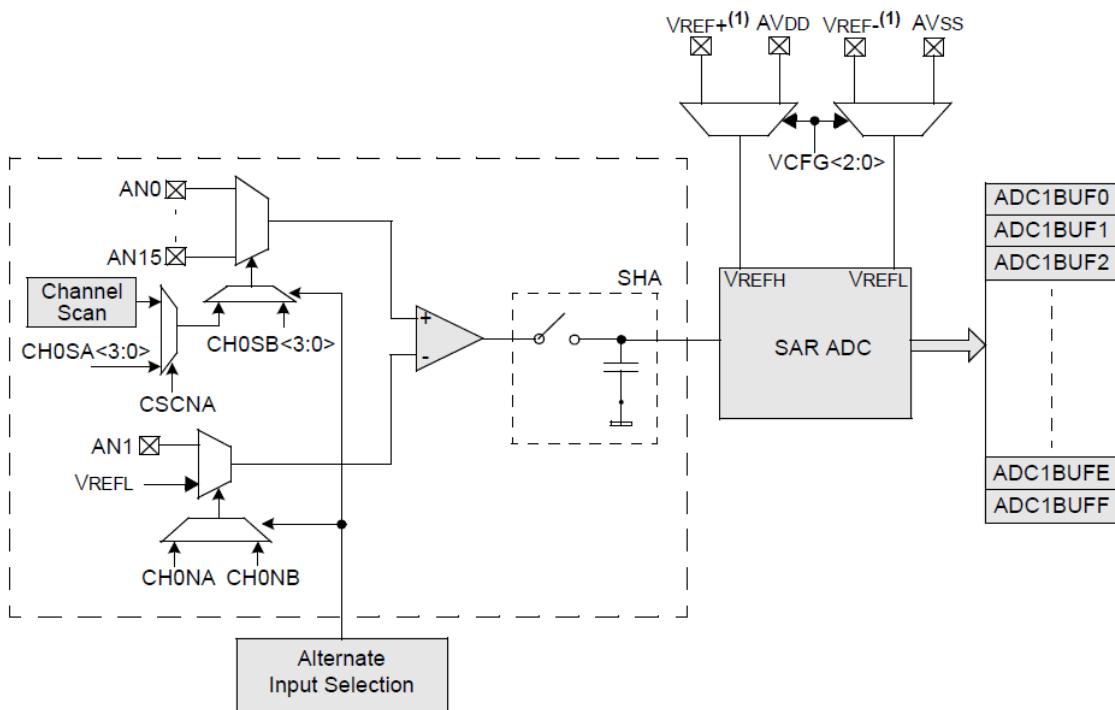
    STBY_HBRIDGE_W = 1; // STBY High
}
```

## 4.8. GESTION DU CONVERTISSEUR AD

Les documents de référence pour cette partie sont :

- La documentation "PIC32 Family Reference Manual" :
  - Sect. 17 10-Bit A-D Converter
- La section ADC Peripheral library de la documentation Harmony.

Le schéma-bloc général du convertisseur est le suivant :

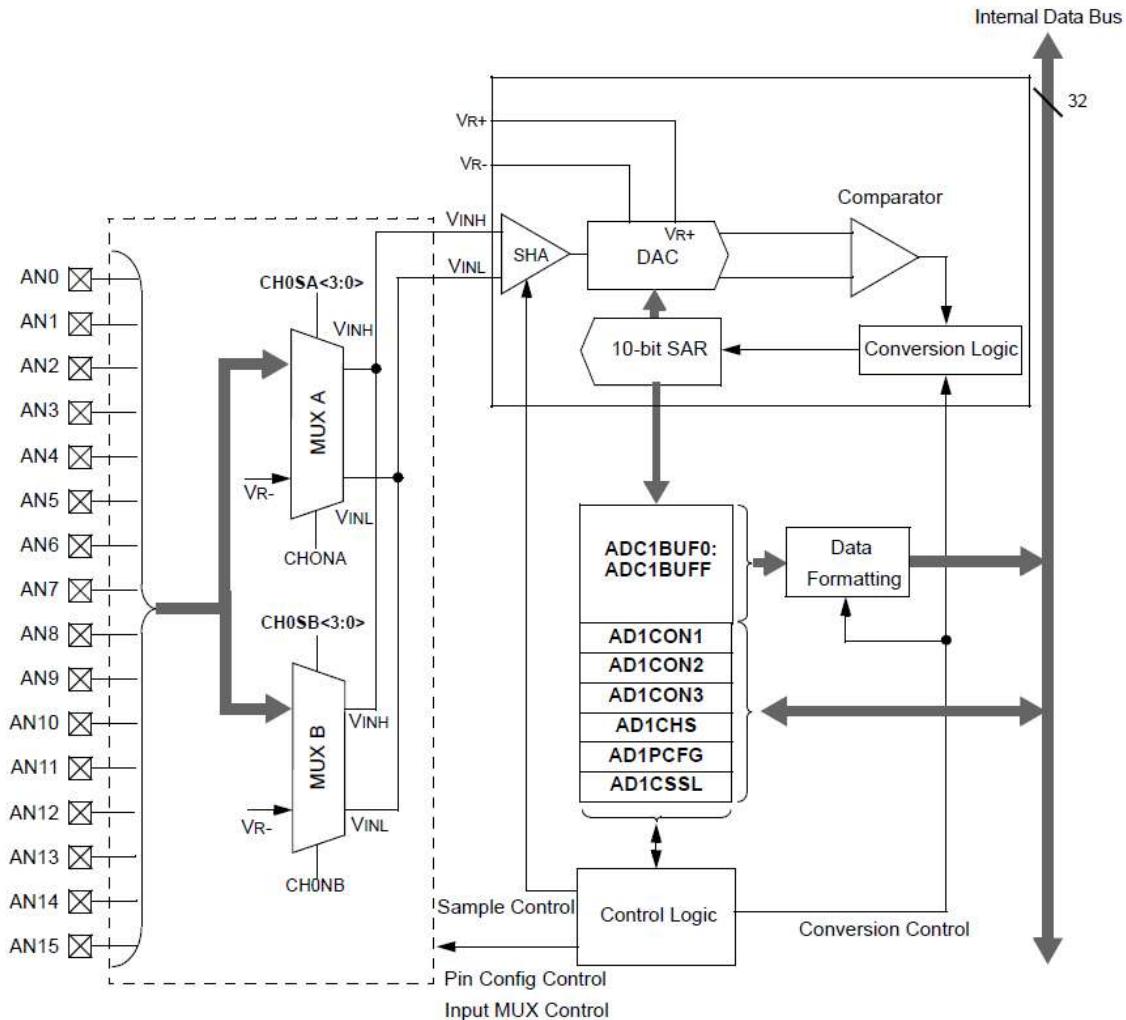


Ses caractéristiques sont :

- Une chaîne de conversion interne :
  - Sample & Hold (échantillonneur-bloqueur) unipolaire, entrée différentielle
  - Convertisseur de type approximations successives, 10 bits
- Jusqu'à 16 entrées (suivant le modèle de PIC32)
- Référence externe possible
- Possibilités de séquençage automatique des entrées à convertir (modes scan et alternatif, les 2 modes pouvant être utilisés simultanément).
- Signal de lancement de conversion configurable (software, par timer, via patte externe)
- Buffer de stockage des résultats :
  - 16 emplacements
  - 2 modes de remplissage : 16-word ou dual 8-word (afin de pouvoir effectuer des conversions en continu)
  - Différents formats de nombres possibles
- Fonctionnement possible lorsque le CPU est en sleep ou idle.

#### 4.8.1. SCHÉMA DE PRINCIPE EN MODE ALTERNÉ

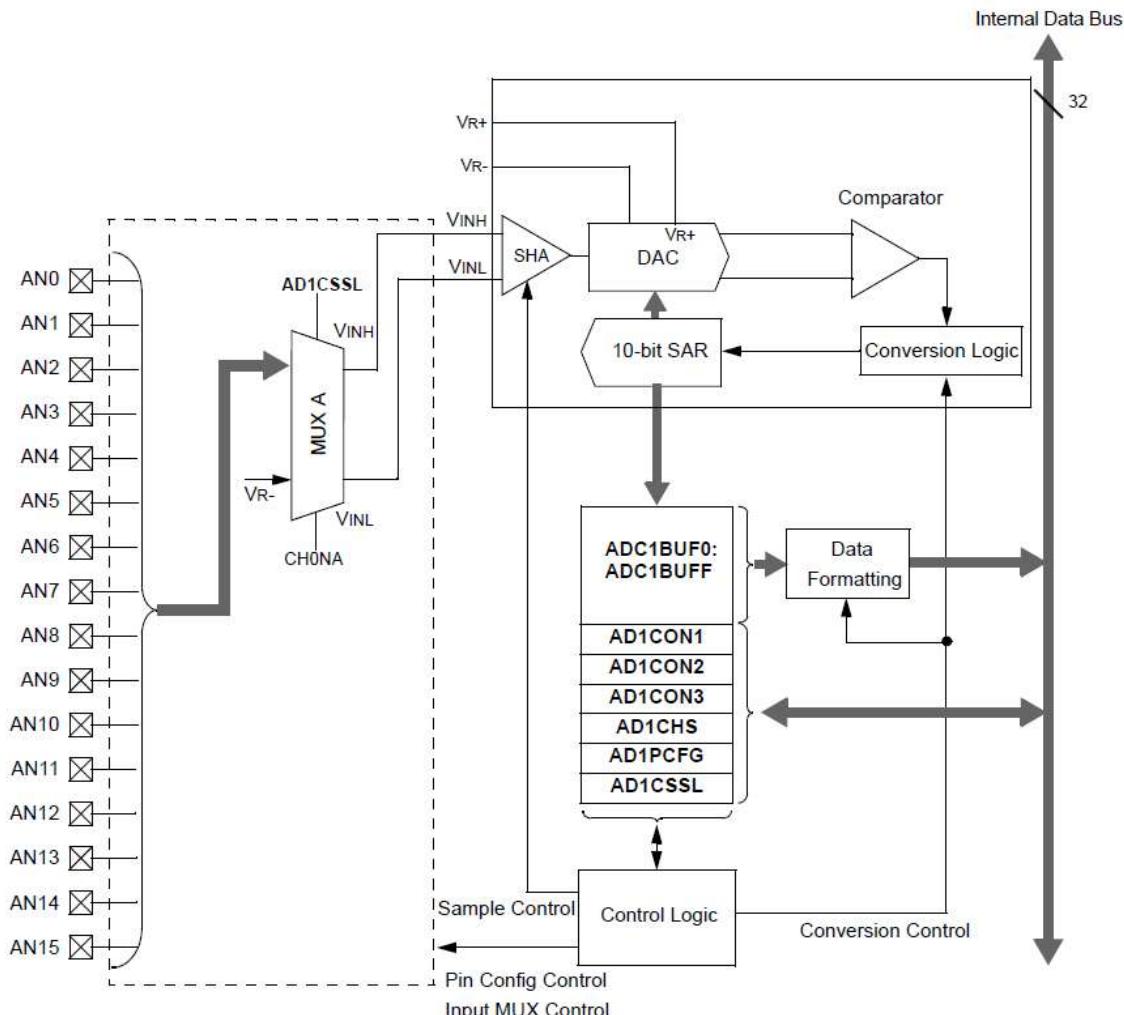
En mode alterné, les entrées provenant du MUX A et du MUX B sont converties en alternance.



Les explications seront faites ci-dessous sur la base d'un exemple (voir fichier Mc32DriverAdcAlt.c du BSP pic32mx\_skes).

#### 4.8.2. SCHÉMA DE PRINCIPE EN MODE SCAN

En mode scan, il faut établir une liste des entrées à convertir, on obtient le résultat dans le buffer du convertisseur.



Les explications seront faites ci-dessous sur la base d'un exemple (voir fichier Mc32DriverAdc.c du BSP pic32mx\_skes.c).

#### 4.8.3. ENTRÉES ANALOGIQUES DU PIC32MX795F512L

On dispose de 16 entrées analogiques AN0 à AN15, elles correspondent à l'entier du port B.

<b>ANx</b>	<b>Port</b>	<b>ANx</b>	<b>Port</b>
AN0	RB0 (Pot 0 kit)	AN8	RB8
AN1	RB1 (Pot 1 kit)	AN9	RB9
AN2	RB2	AN10	RB10
AN3	RB3	AN11	RB11
AN4	RB4	AN12	RB12
AN5	RB5	AN13	RB13
AN6	RB6	AN14	RB14
AN7	RB7	AN15	RB15

⚠️ Attention : Par défaut, l'entier du port B est en analogique. Si on n'utilise que quelques entrées analogiques, il faut configurer en digital les autres en utilisant la fonction PLIB\_PORTS\_AnPinsModeSelect.

Par exemple, pour AN0 et AN1 en analogique et les autres en digital, on écrira :

```
PLIB_PORTS_AnPinsModeSelect(PORTS_ID_0, ~0x0003,
                           PORTS_PIN_MODE_DIGITAL);
```

## 4.9. LES FONCTIONS DE PLIB\_ADC.H

Pour utiliser les nombreuses fonctions il faut inclure le fichier **plib\_adc.h**

Voici les fonctions à disposition pour la gestion du convertisseur AD 10 bits :

Number	Description	Functions associated
1	Selecting the voltage reference source Idle mode control	<a href="#">PLIB_ADC_VoltageReferenceSelect</a> <a href="#">PLIB_ADC_StopInIdleEnable</a> <a href="#">PLIB_ADC_StopInIdleDisable</a>
2	Selecting the ADC conversion clock	<a href="#">PLIB_ADC_ConversionClockSet</a>
3	Input channel selection  Configuring MUX A and MUX B inputs, Alternating MUX A and MUX B input selections, Scanning through several inputs	<b>Scan Mask Selection</b> <a href="#">PLIB_ADC_InputScanMaskAdd</a> <a href="#">PLIB_ADC_InputScanMaskRemove</a> <b>Positive Inputs</b> <a href="#">PLIB_ADC_InputSelectPositive</a> <a href="#">PLIB_ADC_MuxChannel0InputPositiveSelect</a> <a href="#">PLIB_ADC_MuxChannel123InputPositiveSelect</a> <b>Negative Inputs</b> <a href="#">PLIB_ADC_InputSelectNegative</a> <a href="#">PLIB_ADC_MuxChannel0InputNegativeSelect</a> <a href="#">PLIB_ADC_MuxChannel123InputNegativeSelect</a> <b>Scan Mode Selection</b> <a href="#">PLIB_ADC_MuxAInputScanEnable</a> <a href="#">PLIB_ADC_MuxAInputScanDisable</a>
4	Enabling the ADC module	<a href="#">PLIB_ADC_Enable</a>
5	Determine how many S&H channels will be used	<a href="#">PLIB_ADC_ChannelGroupSelect</a>
6	Determine how sampling will occur	<b>Sampling Control</b> <a href="#">PLIB_ADC_SamplingModeEnable</a> <a href="#">PLIB_ADC_SamplingModeDisable</a> <a href="#">PLIB_ADC_SampleAcquisitionTimeSet</a>
7	Selecting Manual or Auto-Sampling	<a href="#">PLIB_ADC_SampleAutoStartEnable</a> <a href="#">PLIB_ADC_SampleAutoStartDisable</a> <a href="#">PLIB_ADC_SamplingStart</a>
8	Select conversion trigger and sampling time	<a href="#">PLIB_ADC_ConversionStart</a> <a href="#">PLIB_ADC_ConversionClockSourceSelect</a> <a href="#">PLIB_ADC_ConversionTriggerSourceSelect</a> <a href="#">PLIB_ADC_ConversionStopSequenceEnable</a> <a href="#">PLIB_ADC_ConversionStopSequenceDisable</a>
9	Select how conversion results are stored in buffer	<a href="#">PLIB_ADC_ResultBufferModeSelect</a>
10	Select the result format Result sign	<a href="#">PLIB_ADC_ResultFormatSelect</a> <a href="#">PLIB_ADC_ResultSignGet</a>
11	Select the number of readings per interrupt	<a href="#">PLIB_ADC_SamplesPerInterruptSelect</a>
12	Select number of samples in DMA buffer for each ADC module and select how the DMA will access the ADC buffers	<a href="#">PLIB_ADC_DMAEnable</a> <a href="#">PLIB_ADC_DMADisable</a> <a href="#">PLIB_ADC_DMABufferModeSelect</a> <a href="#">PLIB_ADC_DMAddressIncrementSelect</a> <a href="#">PLIB_ADC_DMInputChangeBufferSelect</a>
13	Select the 10-bit or 12-bit mode	<a href="#">PLIB_ADC_ResultSizeSelect</a>

14	Channel pair configuration	<code>PLIB_ADC_PairTriggerSourceSelect</code> <code>PLIB_ADC_PairConversionStart</code> <code>PLIB_ADC_PairInterruptRequestEnable</code> <code>PLIB_ADC_PairInterruptRequestDisable</code>
15	Miscellaneous ADC functions:  Asynchronous sampling selection  Early interrupt control  Conversion order selection Global software trigger control User ISR jump address	<code>PLIB_ADC_AsyncronousDedicatedSamplingEnable</code> <code>PLIB_ADC_AsyncronousDedicatedSamplingDisable</code> <code>PLIB_ADC_PairInterruptAfterFirstConversion</code> <code>PLIB_ADC_PairInterruptAfterSecondConversion</code> <code>PLIB_ADC_ConversionOrderSelect</code> <code>PLIB_ADC_GlobalSoftwareTriggerSet</code> <code>PLIB_ADC_IsrJumpTableBaseAddressSet</code>

Les différentes fonctions ne seront pas présentées en détails, mais deux exemples complets sont fournis pour illustrer l'usage de ces différentes fonctions.

#### 4.9.1. EXEMPLE EN MODE ALTERNÉ

Dans cet exemple, on effectue la lecture alternée des 2 entrées analogiques du kit. Cet exemple utilise les fichiers Mc32DriverAdcAlt.h et Mc32DriverAdcAlt.c fournis dans le BSP pic32mx\_skes.

Cet exemple est limité à 2 entrées analogiques. Si on a besoin de plus il vaut mieux utiliser l'exemple en mode scan.

Le principe utilisé consiste à mettre le convertisseur en conversion automatique dans le mode où il commute du MUXA au MUXB. Le MUXA traite AN0 et le MUXB AN1. On utilise les buffers alternés pour éviter de lire en même temps que le convertisseur écrit les résultats.

##### 4.9.1.1. FICHIER MC32DRIVERADCALT.H

Ce fichier contient la définition d'une structure pour récolter la valeur convertie des deux canaux.

```
#include "BSP_config.h"

// Structure pour 2 canaux
typedef struct {
    uint16_t Chan0;
    uint16_t Chan1;
} S_ADCResultsAlt;

/*-----
// Fonction BSP_InitADC10Alt
-----*/
void BSP_InitADC10Alt(void);

/*-----
// Fonction BSP_ReadADCAlt()
-----*/
S_ADCResultsAlt BSP_ReadADCAlt();
```

#### 4.9.1.1. FICHIER MC32DRIVERADCALT.C

Ce fichier contient l'implémentation des deux fonctions. Avec les include suivants :

```
#include "system_config.h"
#include "Mc32DriverAdcAlt.h"
#include "peripheral/adc/plib_adc.h"
```

##### 4.9.1.1.1. Fonction BSP\_InitADC10Alt

Cette fonction, à appeler lors de l'initialisation, configure l'ADC.

```
void BSP_InitADC10Alt(void)
{
    // Configure l'ADC en mode alterné
    PLIB_ADC_ResultFormatSelect(ADC_ID_1,
        ADC_RESULT_FORMAT_INTEGER_16BIT);
    PLIB_ADC_ResultBufferModeSelect(ADC_ID_1,
        ADC_BUFFER_MODE_TWO_8WORD_BUFFERS);
    PLIB_ADC_SamplingModeSelect(ADC_ID_1,
        ADC_SAMPLING_MODE_ALTERNATE_INPUT);

    PLIB_ADC_ConversionTriggerSourceSelect(ADC_ID_1,
        ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);
    PLIB_ADC_VoltageReferenceSelect(ADC_ID_1,
        ADC_REFERENCE_VDD_TO_AVSS );
    PLIB_ADC_SampleAcquisitionTimeSet(ADC_ID_1, 0x1F);
    PLIB_ADC_ConversionClockSet(ADC_ID_1,
        SYS_CLK_FREQ, 32);

    // configure MUXA - traitement AN0
    PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1,
        ADC_MUX_A, ADC_INPUT_POSITIVE_AN0);
    PLIB_ADC_MuxChannel0InputNegativeSelect(ADC_ID_1,
        ADC_MUX_A, ADC_INPUT_NEGATIVE_VREF_MINUS);

    // configure MUXB - traitement AN1
    PLIB_ADC_MuxChannel0InputPositiveSelect(ADC_ID_1,
        ADC_MUX_B, ADC_INPUT_POSITIVE_AN1);
    PLIB_ADC_MuxChannel0InputNegativeSelect(ADC_ID_1,
        ADC_MUX_B, ADC_INPUT_NEGATIVE_VREF_MINUS);
    // Rem CHR le nb d'échantillon par interruption
    // doit correspondre à 2
    PLIB_ADC_SamplesPerInterruptSelect(ADC_ID_1,
        ADC_2SAMPLES_PER_INTERRUPT);
    // Disable scan des 16 canaux
    PLIB_ADC_InputScanMaskRemove(ADC_ID_1, 0xFFFF) ;
    // Start auto sampling
    PLIB_ADC_SampleAutoStartEnable(ADC_ID_1);
    // Enable the ADC module
    PLIB_ADC_Enable(ADC_ID_1);
}
```

Remarque : le principe de la plib\_adc est d'avoir une découpe en fonctions accomplissant des actions élémentaires.

#### 4.9.1.1.1. Fonction **BSP\_ReadADCAlt**

Cette fonction s'utilise chaque fois que l'on veut la valeur des 2 canaux.

L'échantillonnage et la conversion se font automatiquement. Il suffit de stopper l'action automatique afin de lire le buffer sans conflit.

```
S_ADCResultsAlt BSP_ReadADCAlt()
{
    S_ADCResultsAlt result;
    unsigned int offset;
    ADC_RESULT_BUF_STATUS BufStatus;

    PLIB_ADC_SampleAutoStartDisable(ADC_ID_1);
    // on exploite un résultat déjà converti
    // mais on bloque durant la lecture du buffer

    // Traitement avec buffer alterné
    BufStatus = PLIB_ADC_ResultBufferStatusGet(ADC_ID_1);
    if (BufStatus == ADC_FILLING_BUF_0TO7) {
        result.Chan0 =
            PLIB_ADC_ResultGetByIndex(ADC_ID_1, 0);
        result.Chan1 =
            PLIB_ADC_ResultGetByIndex(ADC_ID_1, 1);
    } else {
        result.Chan0 =
            PLIB_ADC_ResultGetByIndex(ADC_ID_1, 8);
        result.Chan1 =
            PLIB_ADC_ResultGetByIndex(ADC_ID_1, 9);
    }

    // Retablit Auto start sampling
    PLIB_ADC_SampleAutoStartEnable(ADC_ID_1);

    return result;
}
```

⌚ La fonction PLIB\_ADC\_ConversionHasCompleted ne retourne jamais OK (conversions en continu) et n'est donc pas utilisable dans ce contexte.

On également observer :

- Il n'y a pas besoin de lancer de conversion, elle se fait automatiquement en continu dès l'initialisation.
- On exploite le double buffer en lisant la dernière conversion mise en place dans le buffer. Le fait de stopper l'auto-conversion empêche une nouvelle écriture dans le buffer.

#### 4.9.2. EXEMPLE EN MODE SCAN

Dans cet exemple, on configure le scan automatique des 2 entrées analogiques du kit. Cet exemple utilise les fichiers Mc32DriverAdc.h et Mc32DriverAdc.h fournis dans le BSP pic32mx\_skes.

☺ Cette solution est facilement adaptable pour un projet avec davantage d'entrées analogiques (au maximum 8 avec les buffers alternés).

Le principe utilisé consiste à mettre le convertisseur en conversion automatique avec une liste d'entrées à scanner. On utilise les buffers alternés pour éviter de lire en même temps que le convertisseur écrit les résultats.

##### 4.9.2.1. FICHIER MC32DRIVERADC.H

Ce fichier contient la définition d'une structure pour récolter la valeur convertie de plusieurs canaux.

```
#include "BSP_config.h"

// Structure à adapter selon le nombre de canaux
// Limite à 8 avec les buffers alterné

typedef struct {
    uint16_t Chan0;
    uint16_t Chan1;
} S_ADCResults ;

/*-----
// Fonction BSP_InitADC10
-----*/
void BSP_InitADC10(void);

/*-----
// Fonction BSP_ReadAllADC()
-----*/

S_ADCResults BSP_ReadAllADC();
```

#### 4.9.2.2. FICHIER MC32DRIVERADC.C

Ce fichier contient l'implémentation des deux fonctions ainsi que la définition de la liste des entrées à scanner.

```
#include "system_config.h"
#include "Mc32DriverAdc.h"
#include "peripheral/adc/plib_adc.h"

// Create the list of channels to scan
// Bit a 1 pour SCAN Bit0 = AN0, Bit1 = AN1 etc...
#define configscan 0x0003 // SCAN AN1 AN0 (Pots du kit)
```

##### 4.9.2.2.1. Fonction BSP\_InitADC10

Cette fonction à appeler lors de l'initialisation, configure l'ADC.

```
/*-----
// Fonction BSP_InitADC10
-----*/
void BSP_InitADC10(void)
{
    // Configure l'ADC
    PLIB_ADC_InputScanMaskAdd(ADC_ID_1, configscan) ;
    PLIB_ADC_ResultFormatSelect(ADC_ID_1,
        ADC_RESULT_FORMAT_INTEGER_16BIT) ;
    PLIB_ADC_ResultBufferModeSelect(ADC_ID_1,
        ADC_BUFFER_MODE_TWO_8WORD_BUFFERS) ;
    PLIB_ADC_SamplingModeSelect(ADC_ID_1,
        ADC_SAMPLING_MODE_MUXA) ;
    PLIB_ADC_ConversionTriggerSourceSelect(ADC_ID_1,
        ADC_CONVERSION_TRIGGER_INTERNAL_COUNT) ;
    PLIB_ADC_VoltageReferenceSelect(ADC_ID_1,
        ADC_REFERENCE_VDD_TO_AVSS ) ;
    PLIB_ADC_SampleAcquisitionTimeSet(ADC_ID_1, 0x1F) ;
    PLIB_ADC_ConversionClockSet(ADC_ID_1,
        SYS_CLK_FREQ, 32) ;

    // Rem CHR le nb d'échantillon par interruption doit
    // correspondre au nb d'entrées de la liste de scan
    PLIB_ADC_SamplesPerInterruptSelect(ADC_ID_1,
        ADC_2SAMPLES_PER_INTERRUPT) ;
    PLIB_ADC_MuxAInputScanEnable(ADC_ID_1) ;

    // Enable the ADC module
    PLIB_ADC_Enable(ADC_ID_1) ;
}
```

Remarque : La séquence d'initialisation a été établie sur la base d'un exemple fourni et des exemples dans la documentation Harmony. Quelques ajustements ont été nécessaires pour obtenir un bon résultat.

#### 4.9.2.2.2. Fonction BSP\_ReadAllADC

Cette fonction s'utilise chaque fois que l'on veut la valeur des canaux de la liste de scan.

L'échantillonnage et la conversion se font automatiquement Il suffit de stopper l'action automatique afin de lire le buffer sans conflit.

```
S_ADCResults BSP_ReadAllADC()
{
    S_ADCResults result;
    unsigned int offset;
    ADC_RESULT_BUF_STATUS BufStatus;

    // Stop sample/convert
    PLIB_ADC_SampleAutoStartDisable(ADC_ID_1);

    // Traitement avec buffer alterné
    BufStatus = PLIB_ADC_ResultBufferStatusGet(ADC_ID_1);
    if (BufStatus == ADC_FILLING_BUF_0TO7) {
        result.Chan0 =
            PLIB_ADC_ResultGetByIndex(ADC_ID_1, 0);
        result.Chan1 =
            PLIB_ADC_ResultGetByIndex(ADC_ID_1, 1);
    } else {
        result.Chan0 =
            PLIB_ADC_ResultGetByIndex(ADC_ID_1, 8);
        result.Chan1 =
            PLIB_ADC_ResultGetByIndex(ADC_ID_1, 9);
    }

    // Auto start sampling
    PLIB_ADC_SampleAutoStartEnable(ADC_ID_1);

    return result;
}
```

☞ Comme dans le cas du mode alterné, la fonction PLIB\_ADC\_ConversionHasCompleted ne retourne jamais OK (conversions en continu) et n'est donc pas utilisable dans ce contexte.

On également observer :

- Egalement identique au mode alterné, il n'y a pas besoin de lancer de conversion, elle se fait automatiquement en continu dès l'initialisation.
- On exploite également le double buffer en lisant la dernière conversion mise en place dans le buffer. Le fait de stopper l'auto-conversion empêche une nouvelle écriture dans le buffer.

De plus, au niveau de la situation des valeurs dans le buffer, on constate que AN0 va dans Buffer[0] et que AN1 va dans Buffer[1]. Si on scannait AN12 en plus par exemple, il serait en Buffer[2].

## 4.10. APPLICATION DE TEST

Cet exemple d'application se base sur une reprise du projet "Exemple1" du chapitre 2, dont les grandes lignes sont les suivantes :

- Timer 1 configuré via le MHC pour une interruption cyclique toutes les 1 ms
- Initialisation de l'affichage, puis affichage d'un message fixe.
- Dans l'application :
  - Ajout de la fonction APP\_UpdateState
  - Ajout de l'écat APP\_STATE\_SERVICE\_TASKS
- L'interruption du timer 1 met l'application dans cet état.

### 4.10.1. MODIFICATION DE L'INITIALISATION ET TEST PRÉLIMINAIRE

Avant d'introduire les actions sur les entrées analogiques et l'AD, on teste le projet pour vérifier si le build est sans erreur et l'exécution conforme. On modifie uniquement le message de bienvenue pour être sûr que l'on utilise bien la copie.

Ce qui donne la situation suivante dans le case APP\_STATE\_INIT :

```
case APP_STATE_INIT:
{
    // Init du LCD
    lcd_init();
    lcd_bl_on();
    // Start du Timer1
    DRV_TMR0_Start();
    printf_lcd("App Exemple IO");
    lcd_gotoxy(1,2);
    printf_lcd("C. Huber 14.11.2016");
    appData.Count = 0;
    appData.state = APP_STATE_WAIT;
    break;
}
```

On obtient bien les leds 0 à 3 allumées et les leds 4 à 7 éteintes ainsi que l'affichage du message, ce qui montre que l'initialisation effectuée par le BSP fonctionne.

### 4.10.2. MODIFICATIONS DE APP.H POUR GESTION DE L'ADC

Pour utiliser le convertisseur AD à partir du BSP, nous introduisons un champ dans la structure APP\_DATA :

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

    S_ADCResults AdcRes;
} APP_DATA;
```

¶ Il est nécessaire d'inclure :  
`#include "Mc32DriverAdc.h"`

#### 4.10.3. MODIFICATIONS DE APP.C POUR GESTION DES IO

Nous allons introduire des actions sur les entrées-sorties dans le case APP\_STATE\_INIT et dans le case APP\_STATE\_SERVICE\_TASKS.

Le traitement correspond à la lecture des 2 potentiomètres, au clignotement de 4 leds et de lecture d'état avec les affichages des résultats. Sans oublier le lancement du Timer pour obtenir le traitement cyclique.

##### 4.10.3.1. AJOUT ACTION IO DANS CASE APP\_STATE\_INIT

Voici les actions d'initialisation :

```
void APP_Tasks (void )
{
    uint16_t ValLatchA;
    bool ToucheOK ;
    bool EtatLed5, EtatLed6;

    /* Check the application's current state. */
    switch ( appData.state )
    {
        /* Application's initial state. */
        case APP_STATE_INIT:
            // Init du LCD
            lcd_init();
            lcd_b1_on();

            printf_lcd("App Exemple IO");
            lcd_gotoxy(1,2);
            printf_lcd("C. Huber 14.11.2016");

            // Init AD en mode scan
            BSP_InitADC10();

            // Eteint led0 et 1, action directe
            LED0_W = 1;
            LED1_W = 1;
            // Eteint led2 avec PLIB_PORTS_PinWrite
            PLIB_PORTS_PinWrite(PORTS_ID_0, LED2_PORT,
                                LED2_BIT, 1);
            // Eteint led3 avec fonction BSP
            BSP_LEDOff(BSP_LED_3);

            // Allume les leds 4 à 6 avec PLIB_PORTS_CLEAR
            PLIB_PORTS_Clear(PORTS_ID_0, PORT_CHANNEL_A,
                            0x80C0);
            // Allume led 7 avec fonction BSP
            BSP_LEDOn(BSP_LED_7);

            // Start du Timer1
            DRV_TMR0_Start();
            appData.state = APP_STATE_WAIT;
            break;

        case APP_STATE_WAIT :
            // nothing to do
            break;
    }
}
```

#### 4.10.3.2. AJOUT ACTION IO DANS CASE APP\_STATE\_SERVICE\_TASKS

Voici les actions d'exécution :

```

case APP_STATE_SERVICE_TASKS:
{
    // Lecture canaux AD et affichage
    appData.AdcRes = BSP_ReadAllADC();
    lcd_gotoxy(1,2);
    printf_lcd("Ch0 %4d Ch1 %4d ",
               appData.AdcRes.Chan0,
               appData.AdcRes.Chan1);

    // Toggle led4 par inversion état, accès direct
    LED4_W = !LED4_R;

    // Toggle led5 par inversion de l'état
    EstatLed5 = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                             LED5_PORT, LED5_BIT);
    EstatLed5 = !EstatLed5;
    PLIB_PORTS_PinWrite(PORTS_ID_0, LED5_PORT,
                         LED5_BIT, EstatLed5);

    // Toggle led 6
    PLIB_PORTS_PinToggle(PORTS_ID_0, LED6_PORT,
                          LED6_BIT);

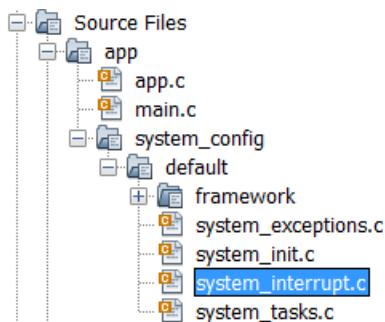
    // Toggle led 7 (fonction BSP)
    BSP_LEDToggle(BSP_LED_7);
    // Lecture état led 5 & 6 et touche OK
    EstatLed5 = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                           LED5_PORT, LED5_BIT);
    EstatLed6 = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                           LED6_PORT, LED6_BIT);
    ToucheOK = PLIB_PORTS_PinGet(PORTS_ID_0,
                                  S_OK_PORT, S_OK_BIT);
    lcd_gotoxy(1,3);
    printf_lcd("led5 %1d led6 %1d OK = %1d",
               EstatLed5, EstatLed6, ToucheOK);
    ValLatchA = PLIB_PORTS_ReadLatched(PORTS_ID_0,
                                         PORT_CHANNEL_A);

    lcd_gotoxy(1,4);
    printf_lcd("PortA = %08X ", ValLatchA);
    appData.state = APP_STATE_WAIT;
    break;
}

```

#### 4.10.4. MODIFICATION CYCLIQUE DE APPDATA.STATE

Cette action est ajoutée dans le fichier system\_interrupt.c



Lors de l'utilisation du MHC nous avions fait le nécessaire pour disposer de l'interruption cyclique d'un timer. C'est dans la routine de réponse que nous ajoutons une action pour modifier cycliquement l'état de l'application.

```

void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
    _IntHandlerDrvTmrInstance0(void)
{
    static int16_t waitCount = 0;

    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
    // Pour test de la période du Timer1
    BSP_LEDToggle(BSP_LED_0);

    waitCount++;
    // Attentes de 3 secondes, puis cycle de 100 ms
    if (waitCount >= 3000) {
        APP_UpdateState(APP_STATE_SERVICE_TASKS);
        waitCount = 2900;
    }
}
    
```

L'interruption cyclique étant à la ms (ce que l'on peut vérifier avec la LED\_0), il est aisément de déclencher une action toutes les n ms à l'aide d'une variable servant de compteur. Dans notre exemple : Toutes les 100 ms après une attente de 3000 ms.

Remarque : comme le fichier system\_interrupt.c inclut déjà **app.h**, il est ainsi possible d'utiliser la fonction **APP\_UpdateState**.

#### 4.10.5. CONTRÔLE DU FONCTIONNEMENT

Le cycle d'interruption à 1 ms se vérifie avec la LED\_0, on observe un signal d'une période de 2 ms. Le cycle application à 100 ms se vérifie avec LED\_5, on obtient un signal d'une période de 200 ms.

Les leds 1 à 3 sont éteintes. Le clignotement des leds 4, 5, 6, 7 est correct.

L'affichage des valeurs brutes des deux potentiomètres est correct.

L'affichage de l'état de la touche OK fonctionne, ainsi que l'état des leds 5 et 6 avec le GetLatched.

La lecture du port A complet nous donne 0x0032 et 0x80F2. Avec 0x0032, les leds 3, 2 et 1 sont à 1 donc éteintes. Avec 0x80F2, la led 6 (bit 15), la led 5 (bit 7) et la led 4 (bit 6) sont à 1 donc éteintes. La led 0 qui est inversée à un rythme plus rapide reste allumée avec le bit 0 = 0.

## 4.11. HISTORIQUE DES VERSIONS

### 4.11.1. V1.0 MAI 2013

Ebauche du document.

### 4.11.2. VERSION 1.1 NOVEMBRE 2013

Complément de l'ébauche pour traiter l'ensemble des actions de gestion des E/S. La section A/D est encore incomplète.

### 4.11.3. VERSION 1.2 DÉCEMBRE 2013

La section A/D est à jour. Annexes à compléter.

### 4.11.4. VERSION 1.3 MARS 2014

Adaptation à la version B du kit PIC32MX795F512L dont le layout a quelque peu changé. Le PIC32MX795F512L remplace le PIC32MX775F512L. Annexes toujours à compléter.

### 4.11.5. VERSION 1.5 OCTOBRE 2014

Refonte du document pour prendre en compte la nouvelle PLIB Harmony et les projets générés par le MHC.

### 4.11.6. VERSION 1.6 SEPTEMBRE 2015

Adaptation du document pour prendre en compte l'évolution du MHC avec MPLABX 3.10 et Harmony 1.06. Ajout utilisation, modification et intégration d'un BSP dans le MHC avec les modifications liées à la fonction SYS\_PORTS\_Initialize();

### 4.11.7. VERSION 1.7 NOVEMBRE 2016

Adaptation du document pour prendre en compte l'évolution du MHC avec MPLABX 3.40 et Harmony 1.08\_01. Adaptation à l'évolution de la réalisation du BSP. Contrôle et mise à jour des actions d'entrées-sorties.

### 4.11.8. VERSION 1.8 NOVEMBRE 2017

Reprise et relecture par SCA.

Enlevé la partie création et intégration d'un BSP.

### 4.11.9. VERSION 1.81 DÉCEMBRE 2018

Supprimé "Annexe A – Liste E/S du PIC32MX795F512L" car redondant avec annexe chapitre 2 théorie.

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

# **Chapitre 5**

## **Timers, interruptions & PWM**

**❖ T.P. PIC32MX**

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



## CONTENU DU CHAPITRE 5

<b>5. Timers, interruptions et PWM</b>	<b>5-1</b>
<b>5.1. Création du projet avec le MHC</b>	<b>5-1</b>
5.1.1. Sélection des timers	5-1
5.1.1.1. Config TMR Driver Instance 0	5-2
5.1.1.2. Config TMR Driver Instance 1	5-2
5.1.1.3. Config TMR Driver Instance 2	5-2
5.1.1.4. Config TMR Driver Instance 3	5-3
5.1.2. Sélection des OC	5-3
5.1.2.1. Config OC Driver instance 0	5-4
5.1.2.2. Config OC Driver instance 1	5-4
<b>5.2. Exploitation des éléments générés par le MHC</b>	<b>5-5</b>
5.2.1. La fonction SYS_Initialize	5-5
5.2.2. Initialisation du timer 1 (DRV_TMR0)	5-6
5.2.3. Initialisation du timer 2 (DRV_TMR1)	5-7
5.2.4. Initialisation du timer 3 (DRV_TMR2)	5-8
5.2.5. Initialisation du timer 4 (DRV_TMR3)	5-10
5.2.6. Initialisation de OC2 (DRV_OC0)	5-11
5.2.7. Initialisation de OC3 (DRV_OC1)	5-12
5.2.8. Principe de fonctionnement des OC	5-13
<b>5.3. Réalisation du test des timers &amp; OC</b>	<b>5-14</b>
5.3.1. Ajout dans initialisation de l'application	5-14
5.3.2. Ajout action Led dans réponses aux interruptions	5-14
5.3.2.1. Action Led dans interruption timer 1	5-14
5.3.2.2. Action Led dans interruption timer 5	5-15
5.3.3. Observation des cycles sur les LED	5-15
5.3.4. Test fonctionnement PWM sur OC2	5-15
5.3.5. Test fonctionnement impulsion OC3	5-16
<b>5.4. Complément application de test des timers &amp; OC</b>	<b>5-18</b>
5.4.1. Adaptation interruption timer 1	5-18
5.4.2. Machine d'état de l'application	5-18
5.4.3. Description de App_task	5-19
5.4.3.1. Détail modulation des PWM	5-20
5.4.3.2. Résultat modulation du PWM	5-21
5.4.3.3. Détail modulation de l'impulsion	5-21
5.4.3.4. Résultat modulation de l'impulsion	5-22
<b>5.5. Conclusion</b>	<b>5-23</b>
<b>5.6. Historique des versions</b>	<b>5-23</b>
5.6.1. V1.0 Juin 2013	5-23
5.6.2. V1.1 Mars 2014	5-23
5.6.3. V1.5 Novembre 2014	5-23
5.6.4. V1.6 Novembre 2015	5-23
5.6.5. V1.7 Novembre 2016	5-23

5.6.6.	V1.8 novembre 2017	5-23
5.6.7.	V1.81 décembre 2018	5-23

## 5. TIMERS, INTERRUPTIONS ET PWM

Dans ce chapitre nous allons étudier sur la base du MHC (MPLAB Harmony Configurator) : comment mettre en œuvre les timers, répondre à une interruption cyclique, ainsi que générer un signal PWM.

Les documents de référence sont :

- La documentation "PIC32 Family Reference Manual" :
  - Section 14 : Timers
- Pour les détails spécifiques au PIC32MX795F512L, il faut se référer au document "PIC32MX5XX/6XX/7XX Family Data Sheet" :
  - Section 13 : Timer 1 et section 14 : Timer 2/3, Timer 4/5
- La documentation d'Harmony, qui se trouve dans <Répertoire Harmony>\v<n>\doc :
  - Section MPLAB Harmony Framework Reference > Peripheral Libraries Help,
  - sous-sections Interrupt Peripheral Library, Output Compare Peripheral Library et Timer Peripheral Library

### 5.1. CRÉATION DU PROJET AVEC LE MHC

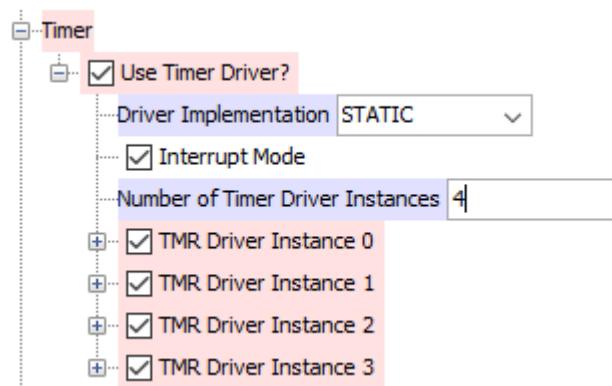
On reprend le même principe de création du projet tel que décrit dans les chapitres précédents. Par contre au niveau de Harmony Framework Configuration dans la section Drivers, nous allons utiliser la section Timer et la section OC (pour PWM).

Nous allons configurer les éléments suivants :

- Timer 1, période 50 ms, interruption niveau 3
- Timer 2, base de temps pour PWM 10 kHz → période 100 µs, pas d'interruption
- Timer 3, base de temps 10 ms pour impulsions, pas d'interruption
- Timer 4&5, période 500 ms, interruption niveau 2
- OC2 en PWM en relation avec timer 2
- OC3 en Continuous pulse en relation avec timer 3

#### 5.1.1. SÉLECTION DES TIMERS

Sous Harmony Framework Configuration > Drivers > Timer, nous introduisons la configuration ci-dessous pour obtenir 4 timers avec des interruptions. Choix de l'implémentation STATIC.



### 5.1.1.1. CONFIG TMR DRIVER INSTANCE 0

Configuration du timer 1 pour une période de 50 ms et priorité d'interruption 3.

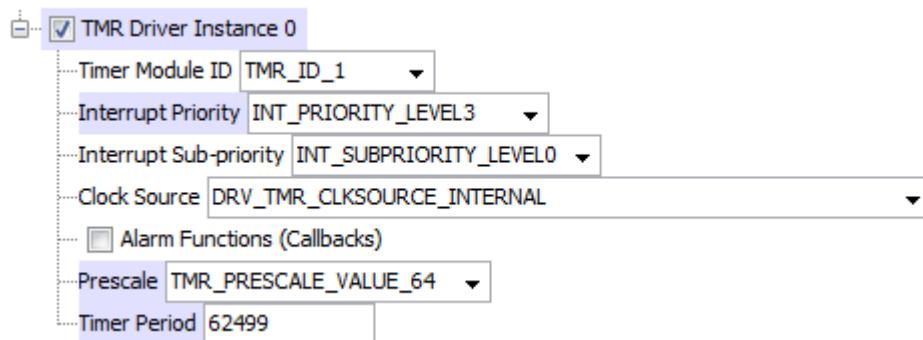
Avec PB\_CLOCK 80 MHz :

TimerPeriode =  $50'000 \mu\text{s} * 80 = 4000000$  dépasse valeur maximale pour timer 16 bits.

Besoin d'un prescaler de  $4000000 / 65536 = 61,03 \Rightarrow 64$

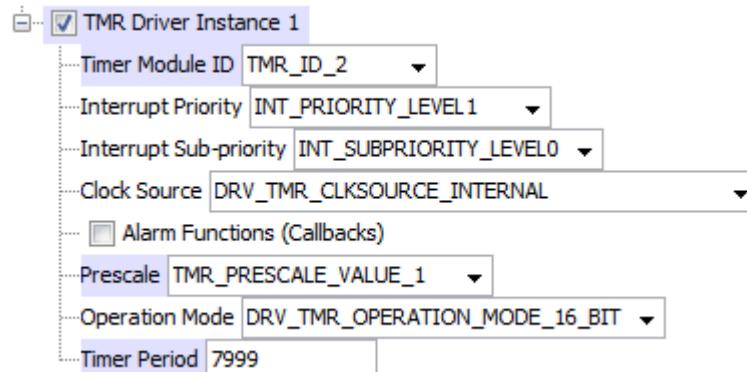
Le timer 1 ne supporte que 1, 8, 64, 256 contrairement à la liste proposée !

TimerPeriode =  $(50'000 \mu\text{s} * 80 / 64) - 1 = 62'499$



### 5.1.1.2. CONFIG TMR DRIVER INSTANCE 1

Configuration du timer 2 pour obtenir un PWM à 10 kHz, donc une période de 100 µs. Avec PB\_CLOCK 80 MHz et un prescaler de 1, il faut une valeur de comparaison de  $(100 \mu\text{s} * 80) - 1 = 7'999$  pour obtenir 100 µs. L'interruption ne sera pas utilisée.



### 5.1.1.3. CONFIG TMR DRIVER INSTANCE 2

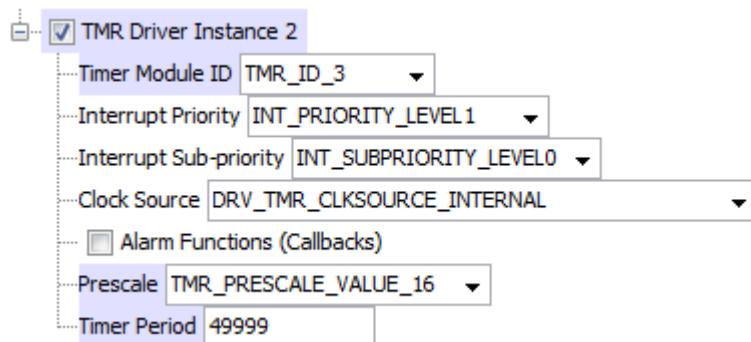
Configuration du timer 3 pour obtenir une période PWM de 10 ms. L'interruption ne sera pas utilisée.

Avec PB\_CLOCK 80 MHz :

TimerPeriode =  $10'000 \mu\text{s} * 80 = 800000$  dépasse valeur maximale pour timer 16 bits.

Besoin d'un prescaler de  $800000 / 65536 = 12,2 \Rightarrow 16$

La période du timer sera donc  $(10000 \mu\text{s} * 80 / 16) - 1 = 49'999$ .



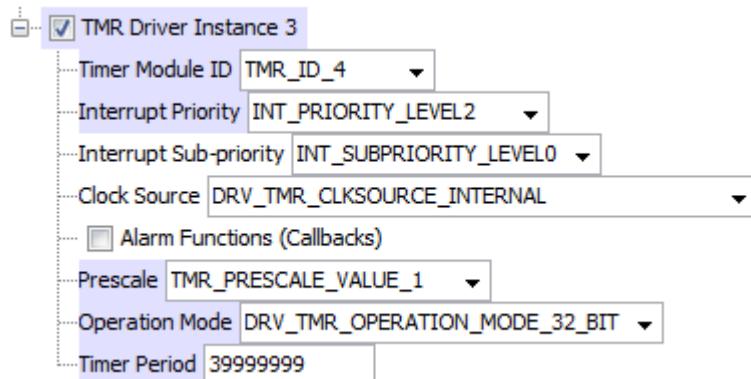
#### 5.1.1.4. CONFIG TMR DRIVER INSTANCE 3

Configuration du timer 4 en 32 bits pour obtenir une période de 500 ms, interruption de niveau 2.

Avec PB\_CLOCK 80 MHz:

$$\text{TimerPeriod} = (500'000 \mu\text{s} * 80) - 1 = 39'999'999 < 2^{32} = 4'294'967'296.$$

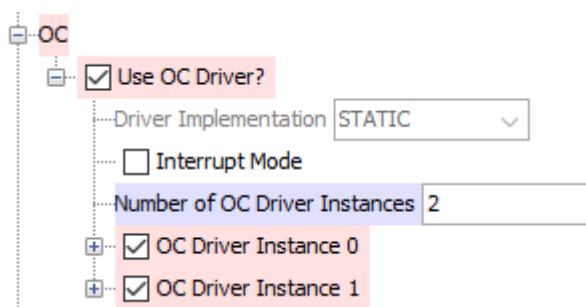
La période du timer sera donc de 39'999'999.



Remarque : en configurant le timer 4 en 32 bits, il y a utilisation de la paire T4 & T5.

#### 5.1.2. SÉLECTION DES OC

Sous Harmony Framework Configuration Drivers, OC nous introduisons la configuration ci-dessous pour configurer 2 OC (OC = Output Compare).



Remarques :

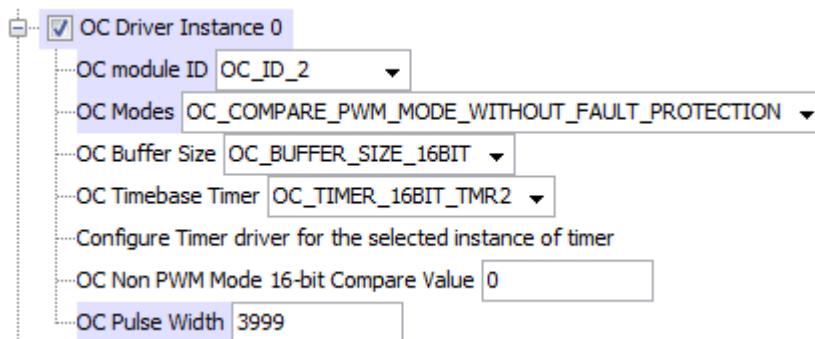
- Pour les drivers OC il n'y a que les static à disposition.
- Les interruptions des OC ne sont pas utiles dans l'application prévue.
- Le lecteur est invité à se reporter aux datasheet du PIC32 ainsi qu'à l'aide de Harmony concernant les différents modes de fonctionnement des OC.

### 5.1.2.1. CONFIG OC DRIVER INSTANCE 0

- Choix de l'OC2, qui correspond à PWMA\_Hbridge sur le kit,
- Utilisation du timer 2 comme base de temps,
- Choix du mode  
OC\_COMPARE\_PWM\_MODE\_WITHOUT\_FAULT\_PROTECTION, qui semble correspondre au PWM souhaité.

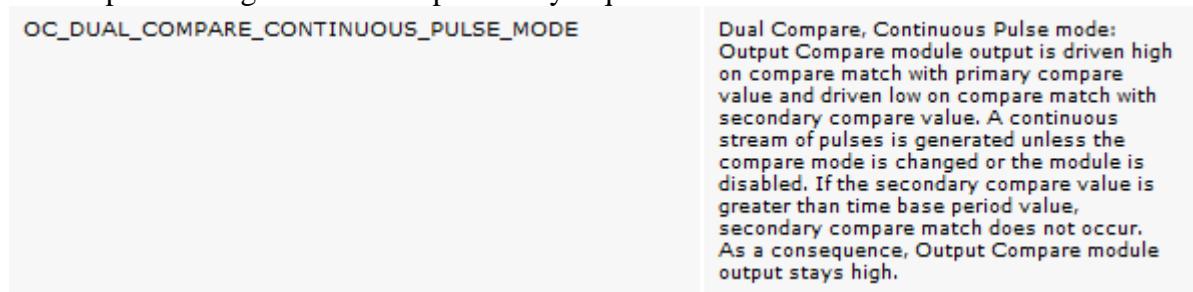


Choix de OC Pulse Width à 3'999, ce qui correspond à un PWM de 50%, car la période du timer 2 correspond à une valeur de 7'999.

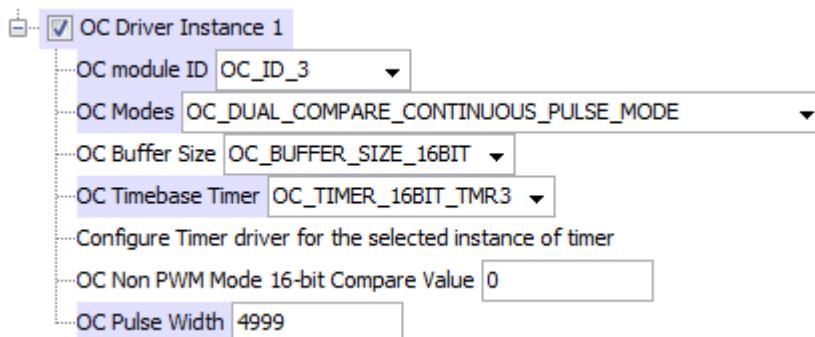


### 5.1.2.2. CONFIG OC DRIVER INSTANCE 1

- Choix de l'OC3, qui correspond à PWMB\_Hbridge sur le kit,
- Utilisation du timer 3 comme base de temps,
- Choix du mode OC\_DUAL\_COMPARE\_CONTINUOUS\_PULSE\_MODE, qui permet de générer une impulsion cyclique.

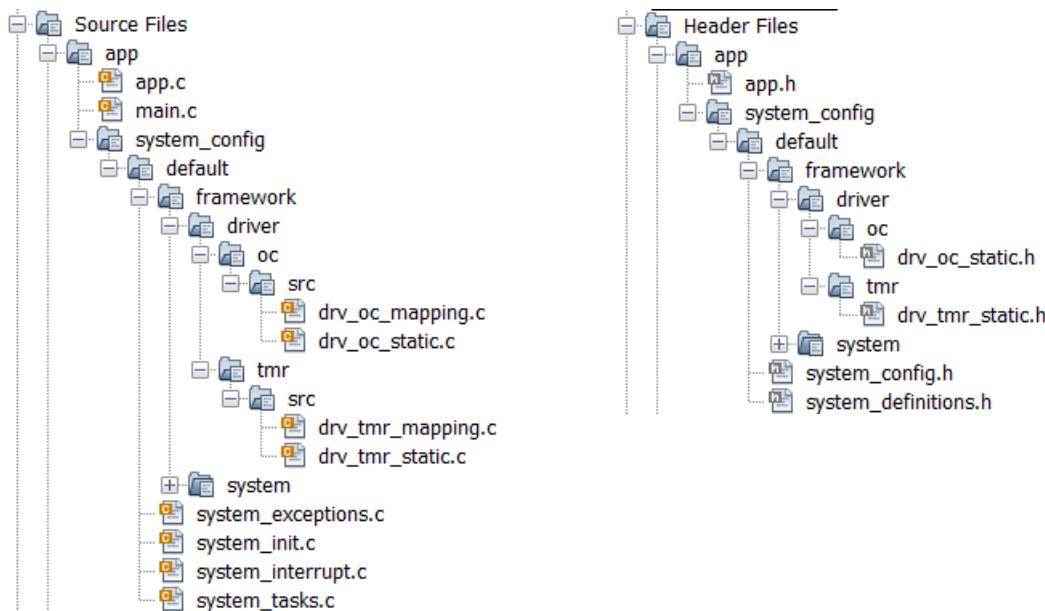


Choix de OC Pulse Width à 4'999, ce qui devrait correspondre à une impulsion de 1 ms car la période du timer 3 de 10 ms correspond à une valeur de 49'999.



## 5.2. EXPLOITATION DES ÉLÉMENTS GÉNÉRÉS PAR LE MHC

Après la génération suite à la configuration effectuée, nous trouvons l'arborescence suivante dans les Sources Files et les Header Files sous app\system\_config.



Le fichier system\_init.c contient les appels aux fonctions de configuration des timers et des OC.

### 5.2.1. LA FONCTION SYS\_INITIALIZE

Voici le contenu de la fonction SYS\_Initialize que l'on trouve dans le fichier system\_init.c :

```
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 the OC Driver */
    DRV_OC0_Initialize();
    DRV_OC1_Initialize();
```

```

/* Initialize TMR0 */
DRV_TMR0_Initialize();
/* Initialize TMR1 */
DRV_TMR1_Initialize();
/* Initialize TMR2 */
DRV_TMR2_Initialize();
/* Initialize TMR3 */
DRV_TMR3_Initialize();

/* Initialize System Services */
/*** Interrupt Service Initialization Code ***/
SYS_INT_Initialize();

/* Initialize Middleware */
/* Enable Global Interrupts */
SYS_INT_Enable();

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

```

Les fonctions d'initialisation des drivers sont automatiquement ajoutées dans la fonction SYS\_Initialize.

### 5.2.2. INITIALISATION DU TIMER 1 (DRV\_TMR0)

La fonction DRV\_TMR0\_Initialize nous permet de découvrir comment sont utilisées les fonctions de PLIB\_TMR pour initialiser le timer, ainsi que celles de PLIB\_INT pour configurer l'interruption.

La fonction DRV\_TMR0\_Initialize se trouve dans le fichier drv\_tmr\_static.c.

```

void DRV_TMR0_Initialize(void)
{
    /* Initialize Timer Instance0 */
    /* Disable Timer */
    PLIB_TMR_Stop(TMR_ID_1);
    /* Select clock source */
    PLIB_TMR_ClockSourceSelect(TMR_ID_1,
                               TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
    /* Select prescalar value */
    PLIB_TMR_PrescaleSelect(TMR_ID_1,
                           TMR_PRESCALE_VALUE_64);
    /* Enable 16 bit mode */
    PLIB_TMR_Mode16BitEnable(TMR_ID_1);
    /* Clear counter */
    PLIB_TMR_Counter16BitClear(TMR_ID_1);
    /*Set period */
    PLIB_TMR_Period16BitSet(TMR_ID_1, 62499);
}

```

```

    /* Setup Interrupt */
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T1,
                                INT_PRIORITY_LEVEL3);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T1,
                                INT_SUBPRIORITY_LEVEL0);
}

```

↳ La source de l'interruption n'est pas activée. Cette action est réalisée par la fonction \_DRV\_TMR0\_Resume elle-même appelée par la fonction DRV\_TMR0\_Start

```

static void _DRV_TMR0_Resume(bool resume)
{
    if (resume)
    {
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                INT_SOURCE_TIMER_1);
        PLIB_INT_SourceEnable(INT_ID_0,
                                INT_SOURCE_TIMER_1);
        PLIB_TMR_Start(TMR_ID_1);
    }
}

bool DRV_TMR0_Start(void)
{
    /* Start Timer*/
    _DRV_TMR0_Resume(true);
    DRV_TMR0_Running = true;

    return true;
}

```

Le timer est stoppé au début de la fonction. Pour lancer le timer, il sera nécessaire dans l'initialisation de l'application d'ajouter un appel à la fonction DRV\_TMR0\_Start.

### 5.2.3. INITIALISATION DU TIMER 2 (DRV\_TMR1)

La fonction DRV\_TMR1\_Initialize effectue l'initialisation du timer 2

```

void DRV_TMR1_Initialize(void)
{
    /* Initialize Timer Instance1 */
    /* Disable Timer */
    PLIB_TMR_Stop(TMR_ID_2);
    /* Select clock source */
    PLIB_TMR_ClockSourceSelect(TMR_ID_2,
                                TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
    /* Select prescalar value */
    PLIB_TMR_PrescaleSelect(TMR_ID_2,
                                TMR_PRESCALE_VALUE_1);
    /* Enable 16 bit mode */
    PLIB_TMR_Mode16BitEnable(TMR_ID_2);
    /* Clear counter */
}

```

```

    PLIB_TMR_Counter16BitClear(TMR_ID_2);
    /*Set period */
    PLIB_TMR_Period16BitSet(TMR_ID_2, 7999);

    /* Setup Interrupt */
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T2,
                                INT_PRIORITY_LEVEL1);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T2,
                                INT_SUBPRIORITY_LEVEL0);
}

```

¶ Comme nous n'avons pas besoin de l'interruption du timer 2, l'activation de la source d'interruption doit être mise en commentaire dans la fonction \_DRV\_TMR1\_Resume.

```

static void _DRV_TMR1_Resume(bool resume)
{
    if (resume)
    {
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                INT_SOURCE_TIMER_2);
        // PLIB_INT_SourceEnable(INT_ID_0,
                                INT_SOURCE_TIMER_2);
        PLIB_TMR_Start(TMR_ID_2);
    }
}

bool DRV_TMR1_Start(void)
{
    /* Start Timer*/
    _DRV_TMR1_Resume(true);
    DRV_TMR1_Running = true;

    return true;
}

```

Le timer 2 est stoppé au début de la fonction. Pour lancer le timer, il sera nécessaire d'ajouter un appel à la fonction DRV\_TMR1\_Start dans l'initialisation de l'application.

#### 5.2.4. INITIALISATION DU TIMER 3 (DRV\_TMR2)

La fonction DRV\_TMR2\_Initialize effectue l'initialisation du timer 3.

```

void DRV_TMR2_Initialize(void)
{
    /* Initialize Timer Instance2 */
    /* Disable Timer */
    PLIB_TMR_Stop(TMR_ID_3);
    /* Select clock source */
    PLIB_TMR_ClockSourceSelect(TMR_ID_3,
                                TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
    /* Select prescalar value */
    PLIB_TMR_PrescaleSelect(TMR_ID_3,
                                TMR_PRESCALE_VALUE_16);
    /* Enable 16 bit mode */
}

```

```

PLIB_TMR_Mode16BitEnable(TMR_ID_3);
/* Clear counter */
PLIB_TMR_Counter16BitClear(TMR_ID_3);
/*Set period */
PLIB_TMR_Period16BitSet(TMR_ID_3, 49999);

/* Setup Interrupt */
PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T3,
                             INT_PRIORITY_LEVEL1);
PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T3,
                                 INT_SUBPRIORITY_LEVEL0);
}

```

¶ Comme nous n'avons pas besoin de l'interruption du timer 3, l'activation de la source d'interruption doit être mise en commentaire dans la fonction **\_DRV\_TMR2\_Resume**.

```

static void _DRV_TMR2_Resume(bool resume)
{
    if (resume)
    {
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                   INT_SOURCE_TIMER_3);
        // PLIB_INT_SourceEnable(INT_ID_0,
        //                         INT_SOURCE_TIMER_3);
        PLIB_TMR_Start(TMR_ID_3);
    }
}

bool DRV_TMR2_Start(void)
{
    /* Start Timer*/
    _DRV_TMR2_Resume(true);
    DRV_TMR2_Running = true;

    return true;
}

```

Le timer 3 est stoppé au début de la fonction. Pour lancer le timer, il sera nécessaire d'ajouter un appel à la fonction **DRV\_TMR2\_Start** dans l'initialisation de l'application.

### 5.2.5. INITIALISATION DU TIMER 4 (DRV\_TMR3)

La fonction DRV\_TMR3\_Initialize effectue l'initialisation du timer 4, ce qui va configurer la paire de timers 4 & 5.

```
void DRV_TMR3_Initialize(void)
{
    /* Initialize Timer Instance3 */
    /* Disable Timer */
    PLIB_TMR_Stop(TMR_ID_4);
    /* Select clock source */
    PLIB_TMR_ClockSourceSelect(TMR_ID_4,
                                TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK);
    /* Select prescalar value */
    PLIB_TMR_PrescaleSelect(TMR_ID_4,
                            TMR_PRESCALE_VALUE_1);
    /* Enable 32 bit mode */
    PLIB_TMR_Mode32BitEnable(TMR_ID_4);
    /* Clear counter */
    PLIB_TMR_Counter32BitClear(TMR_ID_4);
    /*Set period */
    PLIB_TMR_Period32BitSet(TMR_ID_4, 39999999);

    /* Setup Interrupt */
    PLIB_INT_VectorPrioritySet(INT_ID_0, INT_VECTOR_T5,
                               INT_PRIORITY_LEVEL2);
    PLIB_INT_VectorSubPrioritySet(INT_ID_0, INT_VECTOR_T5,
                               INT_SUBPRIORITY_LEVEL0);
}
```

Cette configuration utilise les fonctions 32 bits pour la configuration du timer. Il est à remarquer qu'au niveau des interruptions **c'est le timer 5 (poids fort) qui est source de l'interruption**. La source est activée dans la fonction \_DRV\_TMR3\_Resume.

```
static void _DRV_TMR3_Resume(bool resume)
{
    if (resume)
    {
        PLIB_INT_SourceFlagClear(INT_ID_0,
                                INT_SOURCE_TIMER_5);
        PLIB_INT_SourceEnable(INT_ID_0,
                            INT_SOURCE_TIMER_5);
        PLIB_TMR_Start(TMR_ID_4);
    }
}

bool DRV_TMR3_Start(void)
{
    /* Start Timer*/
    _DRV_TMR3_Resume(true);
    DRV_TMR3_Running = true;
    return true;
}
```

Le timer 4 (poids faible) est stoppé au début de la fonction. Pour lancer le timer, il sera nécessaire d'ajouter un appel à la fonction DRV\_TMR3\_Start dans l'initialisation de l'application.

### 5.2.6. INITIALISATION DE OC2 (DRV\_OC0)

La fonction DRV\_OC0\_Initialize utilise les fonctions de PLIB\_OC pour configurer l'output compare en PWM. Elle se trouve dans le fichier drv\_oc\_static.c.

```
void DRV_OC0_Initialize(void)
{
    /* Setup OC0 Instance */
    // CHR : A ajouter
    PLIB_OC_Disable(OC_ID_2);

    PLIB_OC_ModeSelect(OC_ID_2,
                        OC_COMPARE_PWM_EDGE_ALIGNED_MODE);
    PLIB_OC_BufferSizeSelect(OC_ID_2,
                            OC_BUFFER_SIZE_16BIT);
    PLIB_OC_TimerSelect(OC_ID_2, OC_TIMER_16BIT_TMR2);
    PLIB_OC_Buffer16BitSet(OC_ID_2, 0);
    PLIB_OC_PulseWidth16BitSet(OC_ID_2, 3999);
}
```

¶ Par précaution et par cohérence, il faut ajouter comme 1<sup>ère</sup> action :

```
PLIB_OC_Disable(OC_ID_2);
```

Cela nous permet de mieux comprendre la nécessité d'activer l'OC en appelant dans l'initialisation de l'application la fonction DRV\_OC0\_Enable ou la fonction DRV\_OC0\_Start.

```
void DRV_OC0_Enable(void)
{
    PLIB_OC_Enable(OC_ID_2);
}

void DRV_OC0_Start(void)
{
    PLIB_OC_Enable(OC_ID_2);
}
```

La suite d'appels des fonctions élémentaires de la PLIB\_OC est relativement facile à comprendre. Par la suite, on peut modifier le mode, changer le timer qui sert de base de temps. On utilisera la fonction **PLIB\_OC\_PulseWidth16BitSet** pour modifier le rapport cyclique du PWM.

### 5.2.7. INITIALISATION DE OC3 (DRV\_OC1)

La fonction DRV\_OC1\_Initialize utilise les fonctions de PLIB\_OC pour configurer l'output compare pour générer une impulsion cyclique. Elle se trouve dans le fichier drv\_oc\_static.c.

```
void DRV_OC1_Initialize(void)
{
    /* Setup OC1 Instance */
    // CHR : A ajouter
    PLIB_OC_Disable(OC_ID_3);

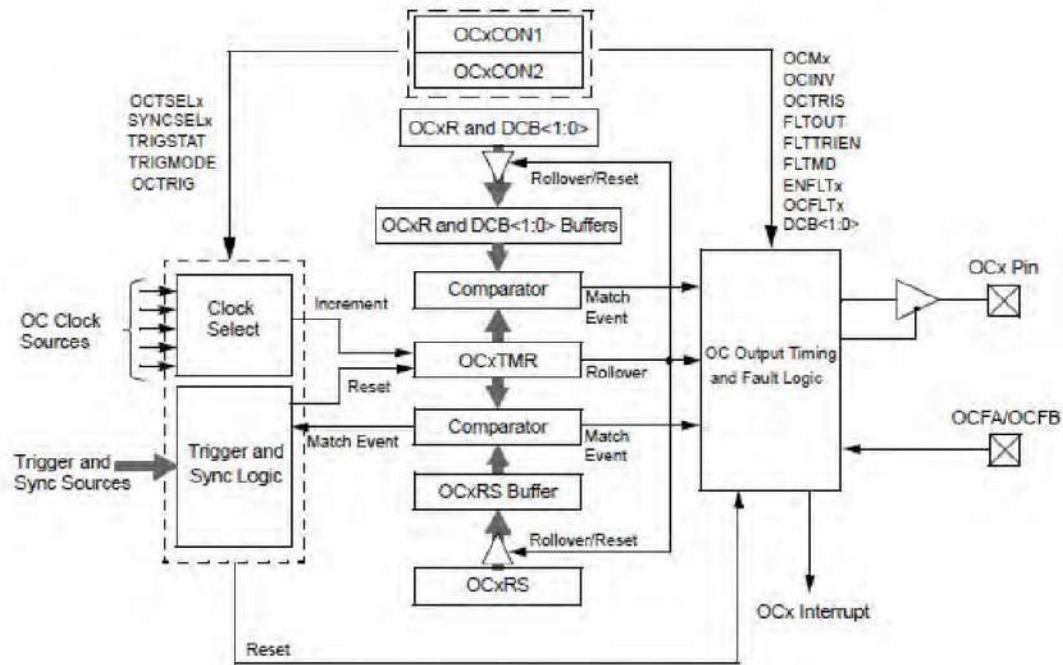
    PLIB_OC_ModeSelect(OC_ID_3,
                        OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE);
    PLIB_OC_BufferSizeSelect(OC_ID_3,
                            OC_BUFFER_SIZE_16BIT);
    PLIB_OC_TimerSelect(OC_ID_3, OC_TIMER_16BIT_TMR3);
    PLIB_OC_Buffer16BitSet(OC_ID_3, 0);
    PLIB_OC_PulseWidth16BitSet(OC_ID_3, 4999);
}
```

♪ A nouveau, par précaution et par cohérence il faut ajouter comme 1<sup>ère</sup> action :

```
PLIB_OC_Disable(OC_ID_3);
```

Par la suite, on utilisera les fonctions PLIB\_OC\_PulseWidth16BitSet et PLIB\_OC\_Buffer16BitSet pour modifier le rapport cyclique de l'impulsion répétitive.

### 5.2.8. PRINCIPE DE FONCTIONNEMENT DES OC



Le schéma de principe ci-dessus nous montre qu'il y a 2 comparateurs gérant la sortie. Dans les modes DUAL\_COMPARE, une des valeurs de référence est utilisée pour obtenir le flanc montant du signal et l'autre le flanc descendant.

- Avec **PLIB\_OC\_Buffer16BitSet(OC\_ID\_2, 0)**, on établit la valeur de référence du comparateur pour le flanc montant. Dans cet exemple, 0 correspond au début de la période du timer.
- Avec **PLIB\_OC\_PulseWidth16BitSet(OC\_ID\_2, 3999)**, on établit la valeur de référence du comparateur pour le flanc descendant. Dans cet exemple, 3999 correspond à la moitié de la période du timer.

Pour modifier le rapport cyclique du signal, on modifie la valeur de référence pour le flanc descendant en utilisant la fonction **PLIB\_OC\_PulseWidth16BitSet**.

## 5.3. RÉALISATION DU TEST DES TIMERS & OC

Nous allons ajouter le code minimal permettant de tester la période des timers, ainsi que des OC.

### 5.3.1. AJOUT DANS INITIALISATION DE L'APPLICATION

Ajout des appels aux fonctions pour démarrer les timers et les OC que l'on place dans APP\_Initialize. On ajoute aussi l'initialisation de l'afficheur LCD et l'affichage d'un message.

```
void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    appData.state = APP_STATE_INIT;

    // Init du LCD
    lcd_init();
    lcd_bl_on();
    // Affichage d'un message
    printf_lcd("App chap5_TimerPwm ");
    lcd_gotoxy(1,2);
    printf_lcd("C. Huber 30.11.2016");

    DRV_TMR0_Start();
    DRV_TMR1_Start();
    DRV_TMR2_Start();
    DRV_TMR3_Start();
    DRV_OC0_Start();
    DRV_OC1_Start();
}
```

### 5.3.2. AJOUT ACTION LED DANS RÉPONSES AUX INTERRUPTIONS

Les routines de réponses aux interruptions se trouvent dans le fichier system\_interrupt.c

#### 5.3.2.1. ACTION LED DANS INTERRUPTION TIMER 1

Dans l'ISR du timer 1 (période 50 ms), on ajoute le toggle de la LED\_1.

```
void __ISR(_TIMER_1_VECTOR, ip13AUTO)
          _IntHandlerDrvTmrInstance0(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
    BSP_LEDToggle(BSP_LED_1);
}
```

Avec une inversion toutes les 50 ms, on obtient un signal d'une période de 100 ms.

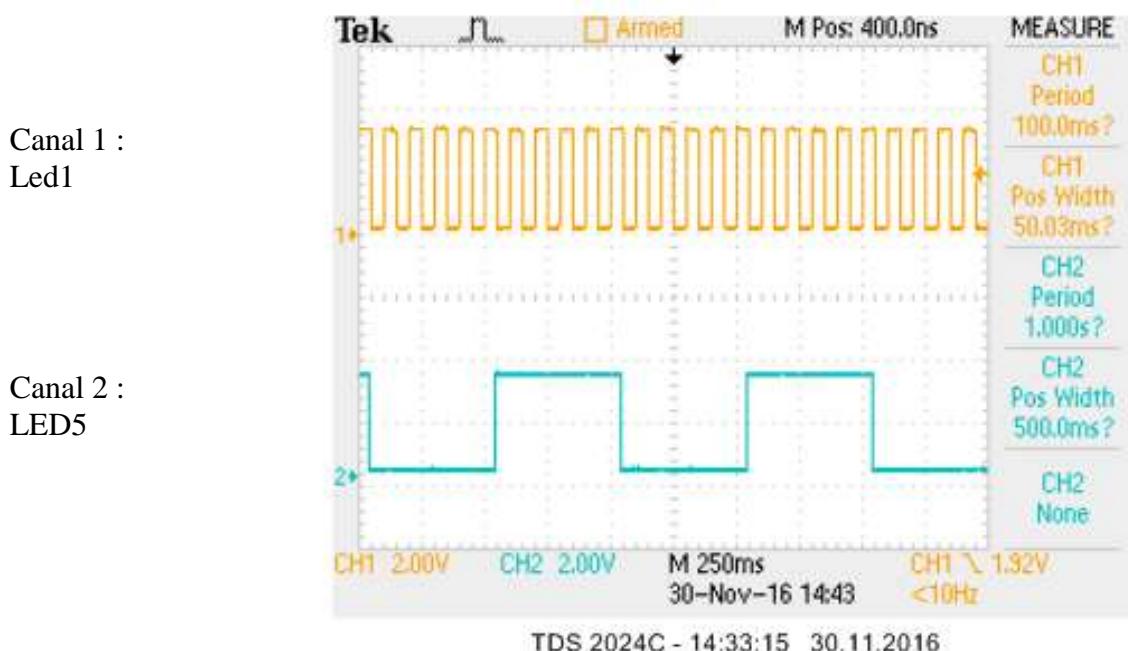
### 5.3.2.2. ACTION LED DANS INTERRUPTION TIMER 5

Dans l'ISR du timer 5 (période 500 ms), on ajoute le toggle de la LED\_5.

```
void __ISR(_TIMER_5_VECTOR, ip12AUTO)
           _IntHandlerDrvTmrInstance3(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_5);
    BSP_LEDToggle(BSP_LED_5);
}
```

Avec une inversion toutes les 500 ms, on obtient un signal d'une période de 1000 ms.

### 5.3.3. OBSERVATION DES CYCLES SUR LES LED



On obtient bien une période de 100 ms pour la LED1 et une période de 1000 ms pour la LED5.

### 5.3.4. TEST FONCTIONNEMENT PWM SUR OC2

☺ Il n'y a rien à faire d'autre que de démarrer les modules initialisés :

```
DRV_TMR1_Start();
DRV_OC0_Start();
```

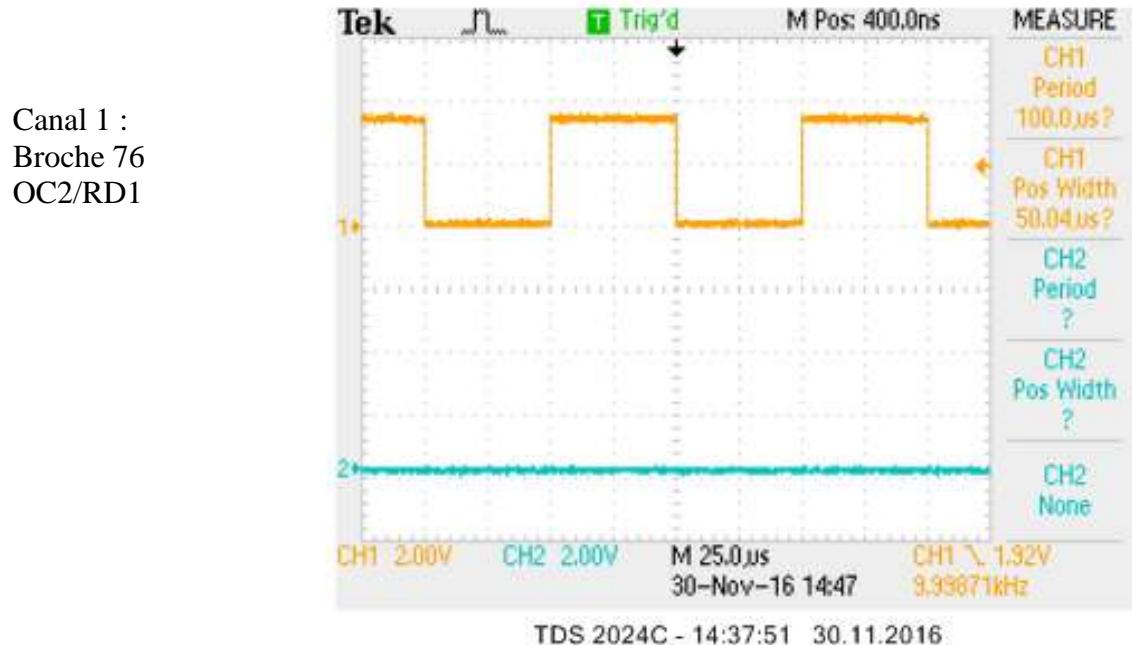
Pour l'OC2 en mode PWM et avec l'utilisation du timer 2 :

```
PLIB_OC_ModeSelect(OC_ID_2,
                    OC_COMPARE_PWM_EDGE_ALIGNED_MODE);
PLIB_OC_TimerSelect(OC_ID_2, OC_TIMER_16BIT_TMR2);
```

La configuration des 2 références de comparaison est :

```
PLIB_OC_Buffer16BitSet(OC_ID_2, 0);
PLIB_OC_PulseWidth16BitSet(OC_ID_2, 3999);
```

La période du timer 2 étant de 8000 ticks pour 100 µs, nous devons obtenir un signal d'une période de 100 µs et d'un rapport cyclique de 50 %.



Ce que la mesure à l'oscilloscope nous permet de vérifier.

Si on écrit PLIB\_OC\_Buffer16BitSet(OC\_ID\_2, 2000), au lieu de 0, cela n'a aucun effet sur le signal (ceci est valable pour le mode PWM).

### 5.3.5. TEST FONCTIONNEMENT IMPULSION OC3

☺ A nouveau, il n'y a rien à faire d'autre que de démarrer les modules initialisés :

```
DRV_TMR2_Start();  
DRV_OC1_Start();
```

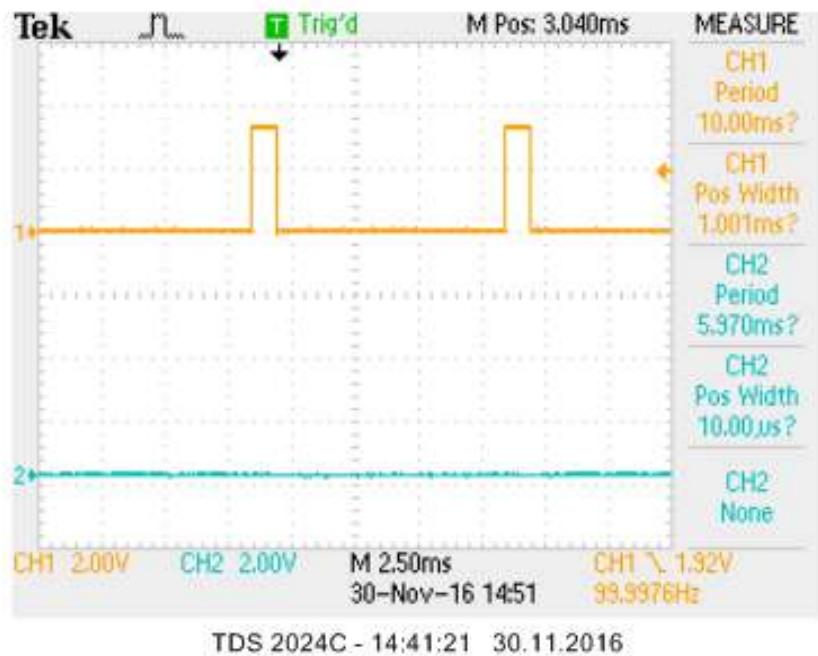
Pour l'OC3 en mode DUAL\_COMPARE\_CONTINUOUS\_PULSE et avec l'utilisation du timer 3 :

```
PLIB_OC_ModeSelect(OC_ID_3,  
                    OC_DUAL_COMPARE_CONTINUOUS_PULSE_MODE);  
PLIB_OC_TimerSelect(OC_ID_3, OC_TIMER_16BIT_TMR3);
```

La configuration des 2 références de comparaison est :

```
PLIB_OC_Buffer16BitSet(OC_ID_3, 0);  
PLIB_OC_PulseWidth16BitSet(OC_ID_3, 4999);
```

La période du timer 3 étant de 49'999, nous devons obtenir un signal d'une période de 10 ms, avec une durée du temps haut de 1 ms soit 10% de la période du timer 3.

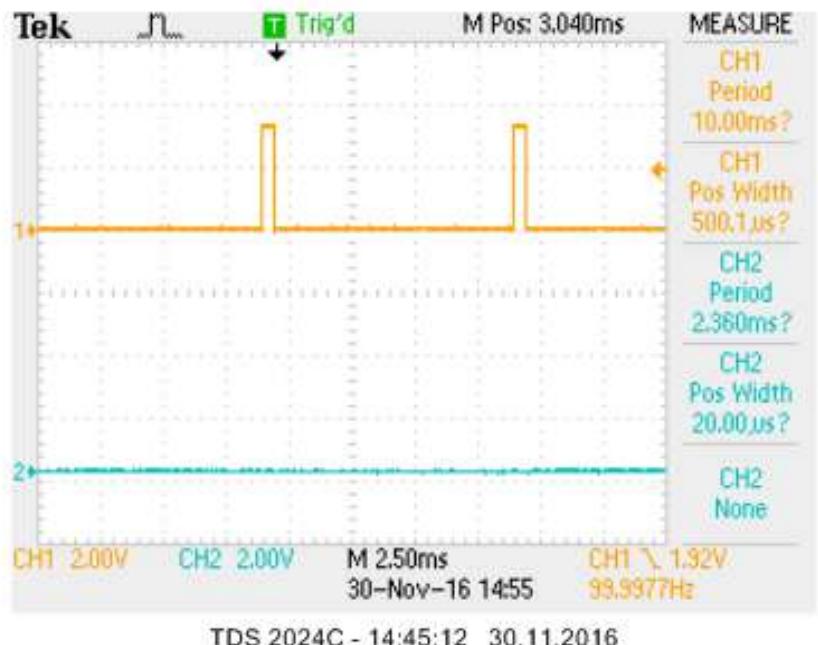


Ce que la mesure à l'oscilloscope nous permet de vérifier.

Si on écrit PLIB\_OC\_Buffer16BitSet(OC\_ID\_3, 2499), au lieu de 0, le signal est modifié.

On obtient une impulsion de seulement 500 μs :

Canal 1 :  
Broche 77  
OC3/RD2



Le flanc montant est retardé de 500 μs d'où un temps haut de 500 μs seulement.

## 5.4. COMPLÉMENT APPLICATION DE TEST DES TIMERS & OC

Pour tester les éléments mis en place par le MHC, nous allons modifier la machine d'état de l'application pour obtenir un traitement cyclique toutes les 50 ms.

Dans ce traitement cyclique, nous allons effectuer la lecture du pot0 pour varier le PWM de 0 à 100%. La lecture du pot1 va nous permettre de varier la largeur d'impulsion de 0,8 ms à 2,2 ms. Ceci dans le but de piloter un servomoteur de modélisme.

Dans la réponse à l'interruption du timer 32 bits, nous conservons l'inversion de la LED5 pour pouvoir vérifier la période. Dans celle du timer 1, nous conservons l'inversion de la LED1 et nous ajoutons l'activation de l'application à chaque 50 ms.

### 5.4.1. ADAPTATION INTERRUPTION TIMER 1

Complément de ISR du timer 1 au niveau du fichier system\_interrupt.c, dans le but d'établir l'état de l'application à APP\_STATE\_SERVICE\_TASKS toutes les 50 ms. Toggle de la LED\_1 à chaque interruption.

```
void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
          _IntHandlerDrvTmrInstance0(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
    BSP_LEDToggle(BSP_LED_1);
    // Etablit état d'exécution
    APP_UpdateState(APP_STATE_SERVICE_TASKS);
}
```

### 5.4.2. MACHINE D'ÉTAT DE L'APPLICATION

Ajout de l'état APP\_STATE\_WAIT ainsi que le prototype de la fonction:

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

Ceci dans app.h. Implémentation de la fonction dans app.c.

Nous ajoutons encore dans la structure :

```
typedef struct
{
    /* The application's current state */
    APP_STATES state;
    /* TODO: Define any additional data used by the
    application. */
    S_ADCResults AdcRes;
    int16_t PulseWidthOC2;
    int16_t PulseWidthOC3;
} APP_DATA;
```

Il est nécessaire d'ajouter dans app.h

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

### 5.4.3. DESCRIPTION DE APP\_TASK

Voici le contenu de la fonction APP\_Tasks au niveau du fichier app.c.

Remarque: avec l'introduction de la machine d'état, on déplace l'activation des timers et OC dans la section INIT du switch. D'où :

```
void APP_Tasks ( void )
{
    /* Check the application's current state. */
    switch ( appData.state )
    {
        /* Application's initial state. */
        case APP_STATE_INIT:
        {
            lcd_init();
            lcd_b1_on();

            // Init AD mode scan
            BSP_InitADC10();

            printf_lcd("App chap5_TimerPwm ");
            lcd_gotoxy(1,2);
            printf_lcd("C. Huber 30.11.2016");
            DRV_TMR0_Start();
            DRV_TMR1_Start();
            DRV_TMR2_Start();
            DRV_TMR3_Start();
            DRV_OC0_Start();
            DRV_OC1_Start();

            appData.state = APP_STATE_WAIT;
            break;
        }

        case APP_STATE_WAIT:
        {
            break;
        }

        case APP_STATE_SERVICE_TASKS:
        {
            // Lecture des 2 pots
            appData.AdRes = BSP_ReadAllADC();
            lcd_gotoxy(1,3);
            printf_lcd("Ch0 %4d Ch1 %4d ",
                      appData.AdRes.Chan0,
                      appData.AdRes.Chan1);
        }
    }
}
```

```

        // Modulation PWM OC2
        appData.PulseWidthOC2 =
            (DRV_TMR1_PeriodValueGet() *
             appData.AdcRes.Chan0 / 1024);
        PLIB_OC_PulseWidth16BitSet(OC_ID_2,
                                    appData.PulseWidthOC2);

        // Modulation PWM OC3
        // 1 ms correspond à 5000 (pér. 10 ms = 50'000)
        // 0.8 ms => 3999 (0 à 3999)
        // 2.2 - 0.8 = 1.4 => 7000
        appData.PulseWidthOC3 = 3999 +
            ((7000 * appData.AdcRes.Chan1) / 1024);
        PLIB_OC_PulseWidth16BitSet(OC_ID_3,
                                    appData.PulseWidthOC3);

        lcd_gotoxy(1,4);
        printf_lcd("OC2 %5d OC2 %5d",
                   appData.PulseWidthOC2,
                   appData.PulseWidthOC3);
        appData.state = APP_STATE_WAIT;
        break;
    }

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

```

#### 5.4.3.1. DÉTAILS MODULATION DES PWM

La valeur brute du canal 0 de l'AD sert à calculer une proportion de la période du timer 2 (7'999). Cette valeur est fournie à la fonction PLIB\_OC\_PulseWidth16BitSet.

☺ Pour éviter d'utiliser la valeur numérique de la période, il est possible d'utiliser la fonction DRV\_TMR1\_PeriodValueGet.

```

uint32_t DRV_TMR1_PeriodValueGet(void)
{
    /* Get 16-bit counter value*/
    return (uint32_t) PLIB_TMR_Period16BitGet(TMR_ID_2);
}

```

☝ Il est aussi possible d'utiliser directement PLIB\_TMR\_Period16BitGet, mais dans ce cas il sera nécessaire d'inclure :

```
#include "peripheral/tmr/plib_tmr.h"
```

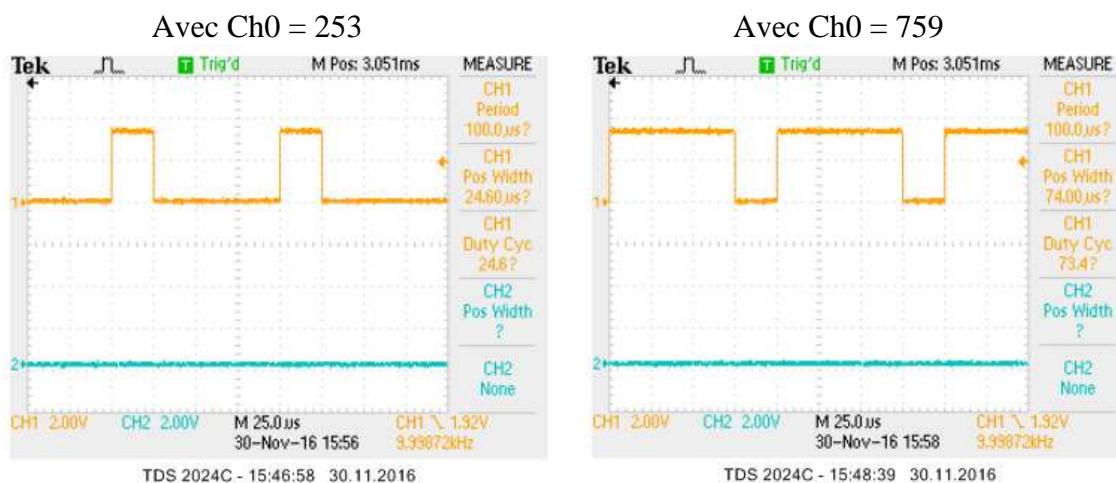
```
// Modulation PWM OC2
appData.PulseWidthOC2 = ( DRV_TMR1_PeriodValueGet() * appData.AdcRes.Chan0 / 1024 );
PLIB_OC_PulseWidth16BitSet(OC_ID_2,
                            appData.PulseWidthOC2);
```

☺ Les calculs étant effectués en 32 bits, on ne rencontre pas de problème de dépassement lors des multiplications, même si AdcRes.Chan0 est 16 bits.

☝ L'appel à PLIB\_OC\_PulseWidth16BitSet nécessite d'inclure :

```
#include "peripheral/oc/plib_oc.h"
```

#### 5.4.3.2. RÉSULTAT MODULATION DU PWM



- Avec Ch0 = 253, cela correspond environ à 25%. La valeur PulseWidthOC2 est de  $7999 * 253 / 1024 = 1976$ .
- Avec Ch0 = 759, cela correspond environ à 75%, la valeur PulseWidthOC2 est de  $7999 * 759 / 1024 = 5928$ .

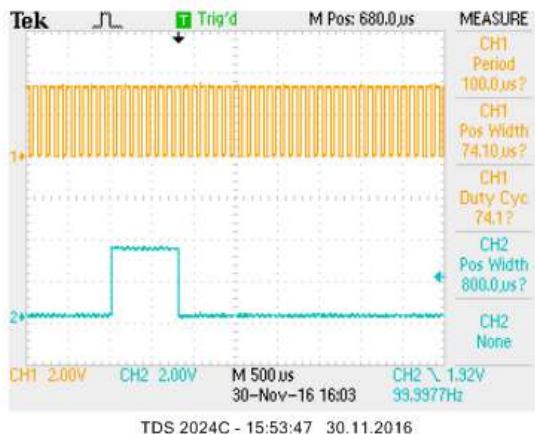
#### 5.4.3.3. DÉTAILS MODULATION DE L'IMPULSION

La période du timer 3 de 10 ms correspond à une valeur de 49'999. Pour obtenir une impulsion qui varie de 0,8 ms à 2,2 ms, on ajoute à la valeur qui correspond à 0,8 ms (3999) une proportion de la valeur qui correspond à la plage de variation de 1,4 ms (7000). La valeur brute du canal 1 de l'AD est utilisée pour établir cette proportion. Puis on applique cette valeur à la fonction PLIB\_OC\_PulseWidth16BitSet.

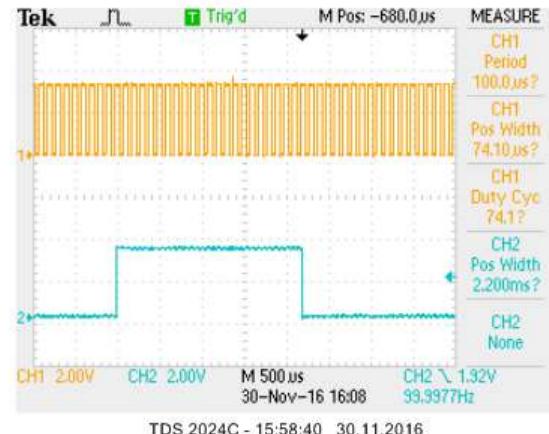
```
// Modulation PWM OC3
// 1 ms correspond à 5000 (pér. 10 ms = 50'000)
// 0.8 ms => 3999 (0 a 3999)
// 2.2 - 0.8 = 1.4 => 7000
PulseWidthOC3 = 3999 + ((7000 * AdcRes.Chan1) / 1024);
PLIB_OC_PulseWidth16BitSet(OC_ID_3, PulseWidthOC3);
```

#### 5.4.3.4. RÉSULTAT MODULATION DE L'IMPULSION

Avec Ch1 = 0



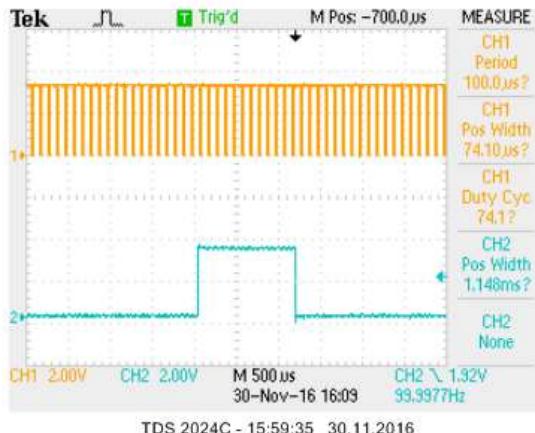
Avec Ch1 = 1023



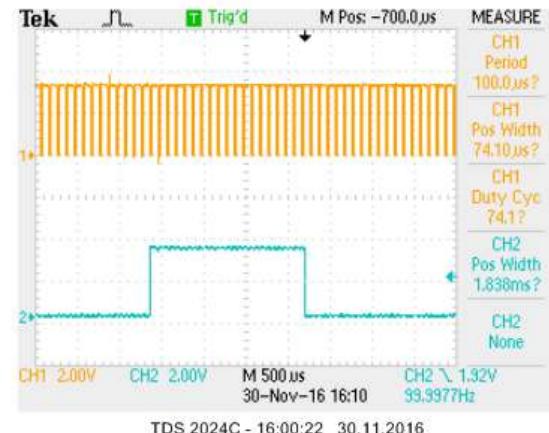
Avec Ch1 = 0, on doit obtenir une impulsion de  $0,8 + (1,4 * 0) / 1024 = 0,8$  ms, soit une valeur de 3999 pour OC2

Avec Ch1 = 1023, on doit obtenir une impulsion de  $0,8 + (1,4 * 1023) / 1024 = 2,199$  ms, soit  $3999 + (7000 * 1023) / 1024 = 10992$

Avec Ch1 = 254



Avec Ch1 = 758



Avec Ch1 = 254, on doit obtenir une impulsion de  $0,8 + (1,4 * 254) / 1024 = 1,147$  ms

Avec Ch1 = 758, on doit obtenir une impulsion de  $0,8 + (1,4 * 758) / 1024 = 1,836$  ms

Mis à part la difficulté à obtenir une valeur stable sur l'ADC, on constate que l'on peut facilement varier les 2 PWM et que l'on obtient de manière assez proche les valeurs prévues.

## 5.5. CONCLUSION

Cette approche basée sur le MHC, qui correspond à une approche de type "recette de cuisine", nous permet déjà de mettre en œuvre les timers et de générer des signaux PWM. C'est au niveau de la théorie que seront étudiés plus en détail les mécanismes et principes de fonctionnement.

## 5.6. HISTORIQUE DES VERSIONS

### 5.6.1. V1.0 JUIN 2013

Création du document.

### 5.6.2. V1.1 MARS 2014

Quelques retouches mineures. (Changement numérotation des chapitres de la doc du XC32 et avec le kit version B on utilise un PIC32MX795F512L).

### 5.6.3. V1.5 NOVEMBRE 2014

Remaniement en profondeur pour étudier les nouvelles PLIB de Harmony 1.0 en utilisant le MHC (Microchip Harmony Configurator).

### 5.6.4. V1.6 NOVEMBRE 2015

Adaptation aux changements de détails (en particulier pour les drivers) introduits par Harmony 1.06. Restructuration des exemples.

### 5.6.5. V1.7 NOVEMBRE 2016

Adaptation aux changements de détails (en particulier pour les drivers) introduits par Harmony 1.08.

### 5.6.6. V1.8 NOVEMBRE 2017

Reprise et relecture par SCA.

### 5.6.7. V1.81 DÉCEMBRE 2018

Relecture et corrections modes OC en relation avec Harmony 2.05 et calculs valeurs OC.

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

# **Chapitre 7**

## **Utilisation de l'USART**

### **❖ T.P. PIC32MX**

**Christian HUBER (CHR)**  
**Serge CASTOLDI (SCA)**  
**Version 1.82 novembre 2021**



## CONTENU DU CHAPITRE 7

<b>7. Utilisation de l'USART</b>	<b>7-1</b>
<b>    7.1. Utilisation de la PLIB_USART</b>	<b>7-1</b>
<b>    7.2. Configuration de l'USART</b>	<b>7-3</b>
7.2.1. Schéma bloc de principe de l'UART	7-3
7.2.1. Liste des UART à disposition (PIC32MX795F512L)	7-3
7.2.2. Schéma détaillé de la transmission	7-4
7.2.3. Schéma détaillé de la réception	7-5
7.2.4. Réalisation de la liaison RS232 sur le kit	7-6
7.2.4.1. Câblage sur le PIC32MX795F512L	7-6
7.2.4.2. Adaptation aux niveaux RS232	7-6
<b>    7.3. Initialisation de l'USART avec MHC</b>	<b>7-7</b>
7.3.1. Handshake Mode	7-7
7.3.2. Operation Mode	7-8
7.3.3. Line Control Mode	7-8
7.3.4. Configuration des modes d'interruption	7-9
7.3.5. La fonction PLIB_USART_InitializeOperation	7-9
7.3.5.1. USART_RECEIVE_INTR_MODE	7-9
7.3.5.2. USART_TRANSMIT_INTR_MODE	7-10
7.3.5.3. USART_OPERATION_MODE	7-10
7.3.6. Code C généré par le MHC	7-12
7.3.7. Modifications Nécessaires	7-13
<b>    7.4. Fonctions de l'émetteur</b>	<b>7-14</b>
7.4.1. La fonction PLIB_USART_TransmitterByteSend	7-14
7.4.2. La fonction PLIB_USART_TransmitterBufferIsFull	7-14
<b>    7.5. Fonctions du récepteur</b>	<b>7-15</b>
7.5.1. La fonction PLIB_USART_ReceiverByteReceive	7-15
7.5.2. La fonction PLIB_USART_ReceiverDataIsAvailable	7-15
7.5.2.1. PLIB_USART_ReceiverDataIsAvailable, exemple	7-15
7.5.3. Fonction PLIB_USART_ReceiverOverrunErrorClear	7-16
<b>    7.6. Fonctions générales de l'USART</b>	<b>7-16</b>
7.6.1. La fonction PLIB_USART_Disable	7-16
7.6.2. La fonction PLIB_USART_Enable	7-17
7.6.3. La fonction PLIB_USART_ErrorGet	7-17
7.6.4. Exemple gestion des erreurs	7-17
<b>    7.7. Réalisation ISR de l'USART</b>	<b>7-18</b>
7.7.1. Diagramme de principe	7-18
7.7.2. Concept émission - réception avec FIFO	7-19
7.7.3. ISR générée par le MHC	7-19
7.7.3.1. La fonction DRV_USART_TasksTransmit	7-20
7.7.3.2. La fonction DRV_USART_TasksReceive	7-20
7.7.3.3. La fonction DRV_USART_TasksError	7-21

7.7.4.	ISR USART, réalisation pratique _____	7-22
7.7.1.	Remarque sur la réalisation pratique_____	7-25
7.7.1.1.	Traitement de la réception_____	7-25
7.7.1.2.	Traitement de l'émission _____	7-25
7.7.2.	Test du mécanisme de réception _____	7-25
7.7.3.	Test du mécanisme d'émission _____	7-26
<b>7.8.</b>	<b>Historique des versions</b> _____	<b>7-27</b>
7.8.1.	V1.0 Avril 2014 _____	7-27
7.8.2.	V1.5 Janvier 2015 _____	7-27
7.8.3.	V1.6 Janvier 2016 _____	7-27
7.8.4.	V1.7 Janvier 2017 _____	7-27
7.8.1.	V1.8 novembre 2017 _____	7-27
7.8.2.	V1.81 janvier 2019 _____	7-27
7.8.3.	V1.82 novembre 2012 _____	7-27

## 7. UTILISATION DE L'USART

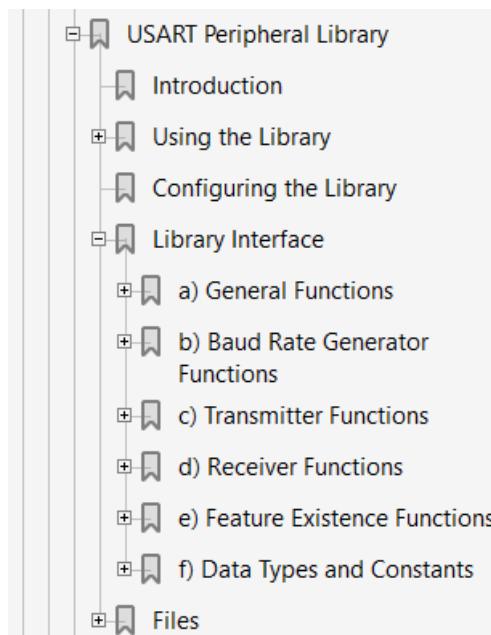
Dans ce chapitre, nous allons étudier comment utiliser l'USART du PIC32MX dans le but de réaliser une communication RS232 avec le kit PIC32MX795F512L.

Les documents de référence sont :

- La documentation "PIC32 Family Reference Manual" :  
Section 21 : UART
- Pour les détails spécifiques au PIC32MX795F512L, il faut se référer au document "PIC32MX5XX/6XX/7XX Family Data Sheet" :  
Section 19 : UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER (UART)
- La documentation d'Harmony, qui se trouve dans <Répertoire Harmony>\v<n>\doc :  
Section MPLAB Harmony Framework Reference > Peripheral Libraries Help,  
sous-section USART Peripheral Library

### 7.1. UTILISATION DE LA PLIB\_USART

Voici l'organisation de la documentation de l'USART Peripheral Library dans la documentation Harmony :



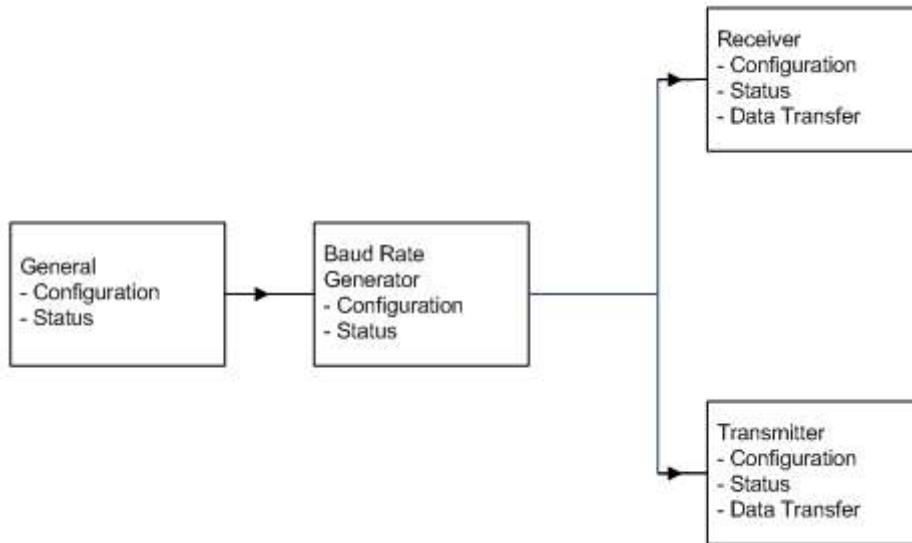
On s'aperçoit que les fonctions sont catégorisées selon leur utilité :

#### Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the USART module.

Library Interface Section	Description
General Functions	Provides setup and configuration interface routines for <ul style="list-style-type: none"> <li>• IrDA</li> <li>• Hardware Flow Control</li> <li>• Operation in power saving modes.</li> <li>• and other general setup</li> </ul>
Baud Rate Generator Functions	Provides setup and configuration interface routines together with status routines for Baud Rate Generator (BRG).
Transmitter Functions	Provides setup, data transfer, error and the status interface routines for the transmitter.
Receiver Functions	Provides setup, data transfer, error and the status interface routines for the receiver.
Feature Existence Functions	Provides interface routines to determine existence of features.

Le modèle d'utilisation est le suivant :



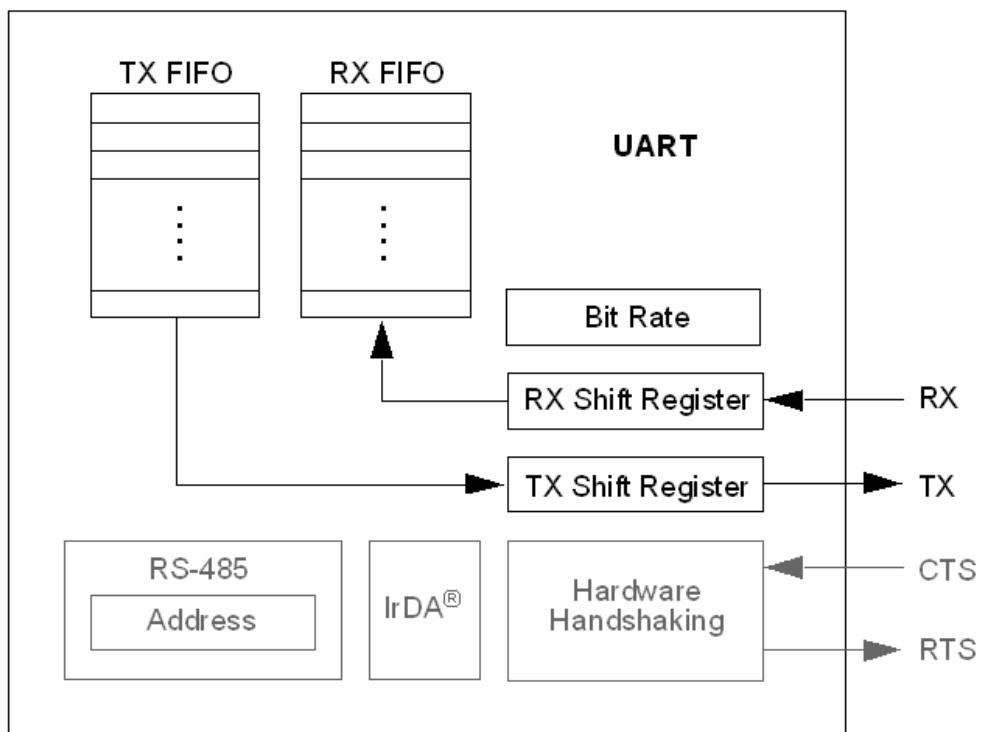
Selon toute logique, les opérations sont donc, dans l'ordre :

1. Configuration générale de l'USART.
2. Configuration du baudrate.
3. Configuration, puis utilisation des parties réception et émission.

## 7.2. CONFIGURATION DE L'USART

Pour bien configurer l'USART, il faut comprendre son architecture.

### 7.2.1. SCHÉMA BLOC DE PRINCIPE DE L'UART



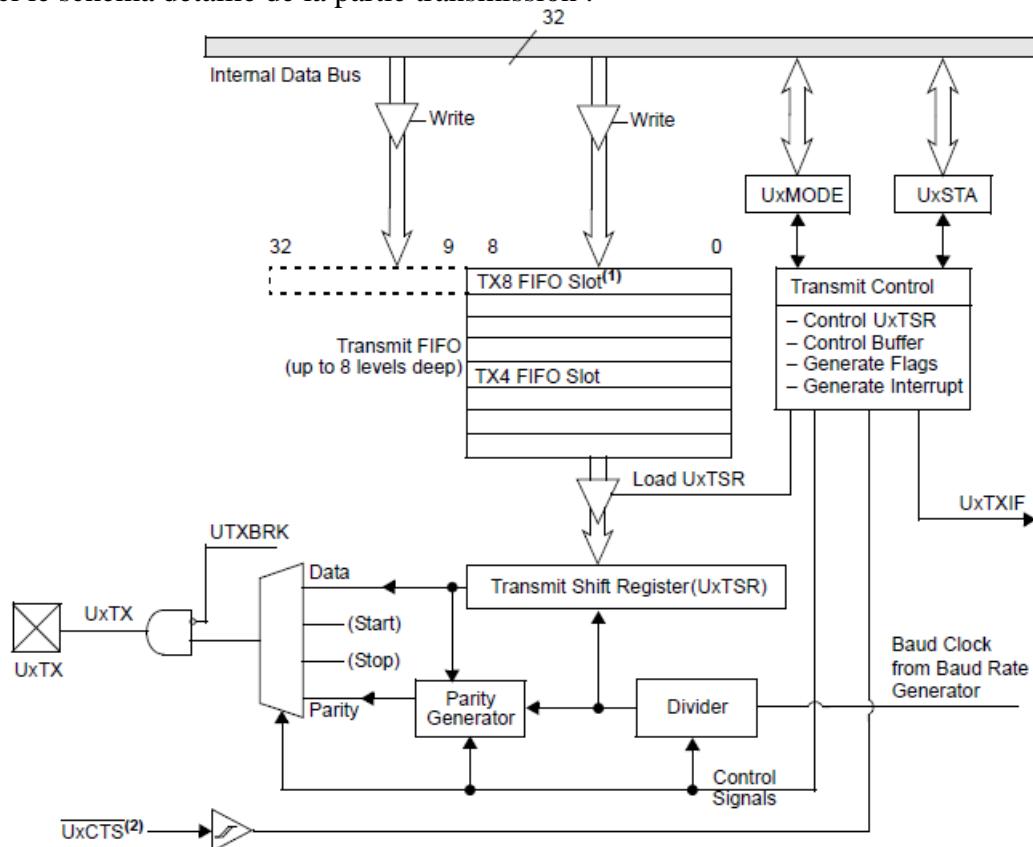
### 7.2.1. LISTE DES UART À DISPOSITION (PIC32MX795F512L)

On dispose de 6 UARTs. Les UARTS 1, 2 et 3 disposent de signaux de handshake (/CTS et /RTS) gérés matériellement.

Pin Name	Pin Number <sup>(1)</sup>			Pin Type	Buffer Type	Description
	64-Pin QFN/TQFP	100-Pin TQFP	121-Pin XBGA			
U1CTS	43	47	L9	I	ST	UART1 clear to send
U1RTS	49	48	K9	O	—	UART1 ready to send
U1RX	50	52	K11	I	ST	UART1 receive
U1TX	51	53	J10	O	—	UART1 transmit
U3CTS	8	14	F3	I	ST	UART3 clear to send
U3RTS	4	10	E3	O	—	UART3 ready to send
U3RX	5	11	F4	I	ST	UART3 receive
U3TX	6	12	F2	O	—	UART3 transmit
U2CTS	21	40	K6	I	ST	UART2 clear to send
U2RTS	29	39	L6	O	—	UART2 ready to send
U2RX	31	49	L10	I	ST	UART2 receive
U2TX	32	50	L11	O	—	UART2 transmit
U4RX	43	47	L9	I	ST	UART4 receive
U4TX	49	48	K9	O	—	UART4 transmit
U6RX	8	14	F3	I	ST	UART6 receive
U6TX	4	10	E3	O	—	UART6 transmit
U5RX	21	40	K6	I	ST	UART5 receive
U5TX	29	39	L6	O	—	UART5 transmit

### 7.2.2. SCHÉMA DÉTAILLÉ DE LA TRANSMISSION

Voici le schéma détaillé de la partie transmission :



Note 1: Refer to the specific device data sheet for availability of 8-level-deep FIFO.

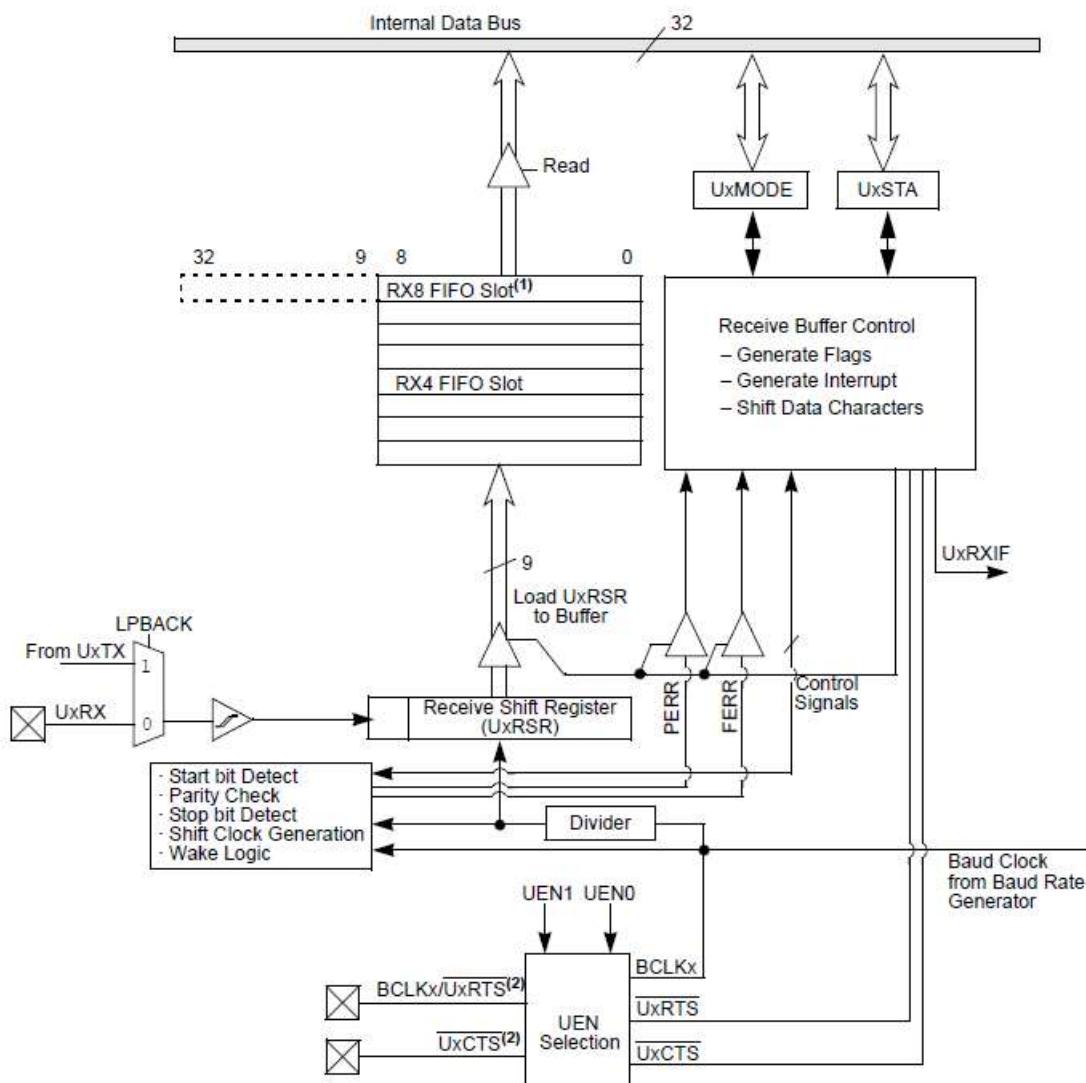
2: Refer to the specific device data sheet for availability of UxCTS pin.

Le PIC32MX795F512L possède un FIFO matériel de 8 éléments.

On constate également la présence de la broche d'entrée /UxCTS qui permet l'autorisation/blocage de l'émission par hardware.

### 7.2.3. SCHÉMA DÉTAILLÉ DE LA RÉCEPTION

Voici le schéma détaillé de la réception :



Note 1: Refer to the specific device data sheet for availability of 8-level-deep FIFO.

2: Refer to the specific device data sheet for availability of UxRTS and UxCTS pins.

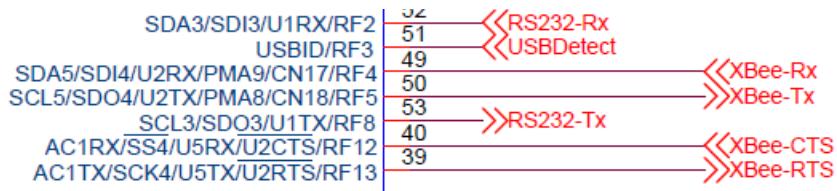
Le PIC32MX795F512L possède un FIFO matériel de 8 éléments.

On constate la présence des broches /UxRTS et /UxCTS qui permettent le handshaking hardware.

## 7.2.4. RÉALISATION DE LA LIAISON RS232 SUR LE KIT

### 7.2.4.1. CABLAGE SUR LE PIC32MX795F512L

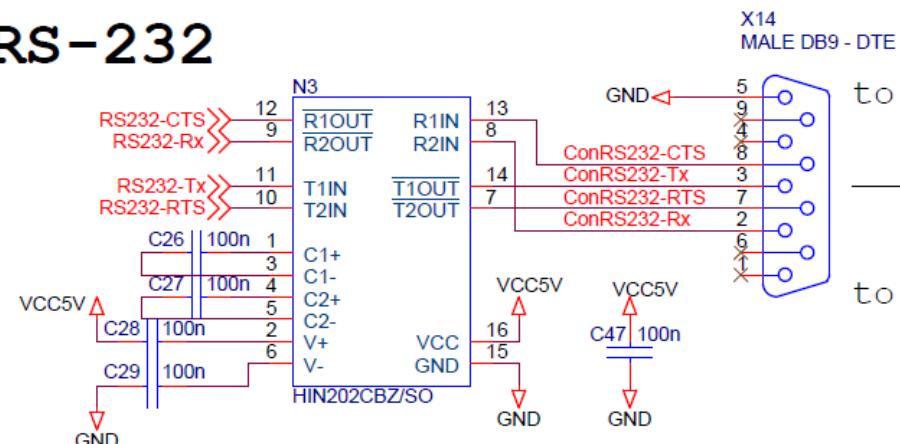
Comme on peut l'observer sur l'extrait du câblage du PIC32MX795F512L, c'est l'UART 1 qui est câblé sur le kit :



Les signaux /CTS et /RTS sont câblés sur les broches permettant un traitement automatique (par le PIC) du handshaking.

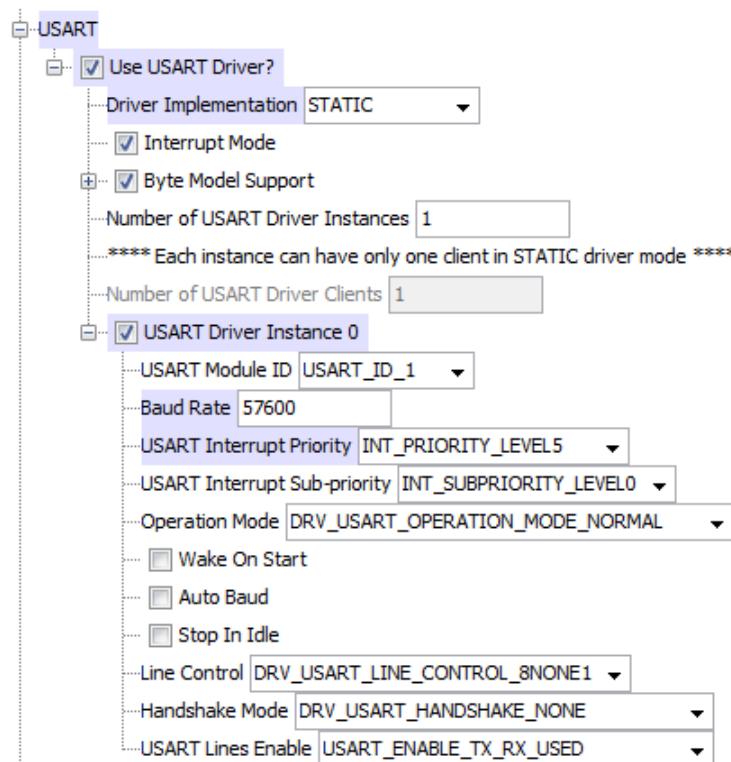


### 7.2.4.2. ADAPTATION AUX NIVEAUX RS232



### 7.3. INITIALISATION DE L'USART AVEC MHC

Pour découvrir les fonctions de configuration, nous utilisons le Microchip Harmony Configurator et nous introduisons un driver USART avec les choix suivants :



Cette configuration a été effectuée avec les versions de logiciels suivants :

- MPLABX 4.15
- Harmony 2.05

Ceci correspond à utiliser l'USART 1 avec un baudrate de 57600 [Bd].

On ne peut pas choisir le comportement des interruptions de réception et d'émission. Il faudra le faire manuellement.

Les éléments pour lesquels le choix n'est pas évident sont développés ci-après.

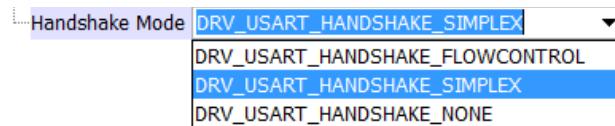
#### 7.3.1. HANDSHAKE MODE

Pour le Handshake mode, on dispose de 2 possibilités :

Members	Description
USART_HANDSHAKE_MODE_FLOW_CONTROL	Enables flow control
USART_HANDSHAKE_MODE_SIMPLEX	Enables simplex mode communication, no flow control

Choix du mode SIMPLEX pour ne pas activer l'automatisme de gestion et permettre une gestion par logiciel pour découvrir le principe.

Remarque : Au niveau du MHC, le menu déroulant a 3 éléments :



Si on choisit NONE, il n'y a pas génération de la fonction de configuration :

Par contre si on choisit SIMPLEX on obtient la génération de la fonction PLIB\_USART\_HandshakeModeSelect.

```
PLIB_USART_HandshakeModeSelect(USART_ID_1,  
DRV_USART_HANDSHAKE_SIMPLEX);
```

### 7.3.2. OPERATION MODE

Pour l'operation mode, on dispose de 4 possibilités :

Members	Description
DRV_USART_OPERATION_MODE_IRDA	USART works in IRDA mode
DRV_USART_OPERATION_MODE_NORMAL	This is the normal point to point communication mode where the USART communicates directly with another USART by connecting its Transmit signal to the external USART's Receiver signal and vice versa. An external transceiver may be connected to obtain RS-232 signal levels. This type of connection is typically full duplex.
DRV_USART_OPERATION_MODE_ADDRESSED	This is a multi-point bus mode where the USART can communicate with many other USARTS on a bus using an address-based protocol such as RS-485. This mode is typically half duplex and the physical layer may require a transceiver. In this mode every USART on the bus is assigned an address and the number of data bits is 9 bits
DRV_USART_OPERATION_MODE_LOOPBACK	Loopback mode internally connects the Transmit signal to the Receiver signal, looping data transmission back into this USART's own input. It is useful primarily as a test mode.

Utilisation du mode normal.

Au niveau du code généré, on obtient :

```
/* Initialize the USART based on configuration settings */  
PLIB_USART_InitializeModeGeneral(USART_ID_1,  
    false, /* Auto baud */  
    false, /* LoopBack mode */  
    false, /* Auto wakeup on start */  
    false, /* IRDA mode */  
    false); /* Stop In Idle mode */
```

### 7.3.3. LINE CONTROL MODE

Pour le Line Control Mode on dispose de 8 possibilités :

Members	Description
USART_8N1	8 Data Bits, No Parity, one Stop Bit
USART_8E1	8 Data Bits, Even Parity, 1 stop bit
USART_8O1	8 Data Bits, odd Parity, 1 stop bit
USART_8N2	8 Data Bits, No Parity, two Stop Bits
USART_8E2	8 Data Bits, Even Parity, 2 stop bits
USART_8O2	8 Data Bits, odd Parity, 2 stop bits
USART_9N1	9 Data Bits, No Parity, 1 stop bit
USART_9N2	9 Data Bits, No Parity, 2 stop bits

Notre choix est 8N1 pour 8 bits data, pas de parité et un seul bit de stop.

Au niveau du code généré, on obtient :

```
/* Set the line control mode */
PLIB_USART_LineControlModeSelect(USART_ID_1,
    DRV_USART_LINE_CONTROL_8NONE1);
```

### 7.3.4. CONFIGURATION DES MODES D'INTERRUPTION

⌚ Le MHC ne permet pas un réglage des interruptions de l'USART en fonction du niveau des buffers. Toutefois, le PIC32MX795F512L offre cette possibilité.

On obtient par défaut la configuration suivante avec l'appel de la fonction **PLIB\_USART\_InitializeOperation** :

```
PLIB_USART_InitializeOperation(USART_ID_1,
    USART_RECEIVE_FIFO_ONE_CHAR,
    USART_TRANSMIT_FIFO_EMPTY,
    USART_ENABLE_TX_RX_USED);
```

On aura une interruption de réception à chaque caractère reçu, et une interruption d'émission dès que le buffer d'envoi est vide. En cas de nécessité, ceci pourra être modifié manuellement (voir ci-dessous).

### 7.3.5. LA FONCTION **PLIB\_USART\_INITIALIZEOPERATION**

Cette fonction configure trois aspects :

- Le mode de l'interruption de réception
- Le mode de l'interruption d'émission
- Le mode d'utilisation de l'USART

Voici le prototype de cette fonction :

```
void PLIB_USART_InitializeOperation(USART_MODULE_ID index,
    USART_RECEIVE_INTR_MODE receiveInterruptMode,
    USART_TRANSMIT_INTR_MODE transmitInterruptMode,
    USART_OPERATION_MODE operationMode);
```

#### 7.3.5.1. USART\_RECEIVE\_INTR\_MODE

Permet de configurer le comportement de l'interruption de réception en relation avec la situation du FIFO hardware de réception.

Nous disposons de 3 possibilités :

Members	Description
USART_RECEIVE_FIFO_HALF_FULL	Interrupt when receive buffer is half full
USART_RECEIVE_FIFO_3B4FULL	Interrupt when receive buffer is 3/4 full
USART_RECEIVE_FIFO_ONE_CHAR	Interrupt when a character is received

Comme la taille du FIFO hardware de réception est de 8, avec HALF\_FULL l'interruption se produit lorsque l'on a reçu 4 caractères. Avec 3B4FULL, l'interruption se produit lorsque l'on a reçu 6 caractères et avec ONE\_CHAR l'interruption se produit dès que l'on a reçu un caractère.

- Les choix 3B4FULL et HALF\_FULL peuvent être judicieux en cas de réception continue de trames. Le CPU n'est alors pas interrompu à chaque caractère.
- Le mode ONE\_CHAR exploite peu le FIFO hardware, mais permet de traiter au fur et à mesure de l'arrivée des caractères.

### 7.3.5.2. USART\_TRANSMIT\_INTR\_MODE

Permet de configurer le comportement de l'interruption d'émission en relation avec la situation du FIFO hardware d'émission.

Nous disposons de 3 possibilités :

Members	Description
USART_TRANSMIT_FIFO_EMPTY	Interrupt when the transmit buffer becomes empty
USART_TRANSMIT_FIFO_IDLE	Interrupt when all characters are transmitted
USART_TRANSMIT_FIFO_NOT_FULL	Interrupt when at least one location is empty in the transmit buffer

- Avec FIFO\_EMPTY on obtient l'interruption lorsque le FIFO d'émission est vide, c'est notre choix.
- La situation FIFO\_IDLE correspond à attendre que le FIFO soit vide et que le dernier caractère ait été transmis.
- Avec la situation FIFO\_NOT\_FULL, on obtient l'interruption dès qu'il y a une place dans le FIFO d'émission. Cela signifie que l'interruption se produit après chaque caractère transmis.

¶ C'est dans le traitement de l'interruption que l'on étudiera le principe d'utilisation.

### 7.3.5.3. USART\_OPERATION\_MODE

Permet de configurer l'utilisation des éléments de l'USART.

Nous disposons de 4 possibilités :

Members	Description
USART_ENABLE_TX_RX_BCLK_USED	TX, RX and BCLK pins are used by USART module
USART_ENABLE_TX_RX_CTS_RTS_USED	TX, RX, CTS and RTS pins are used by USART module
USART_ENABLE_TX_RX_RTS_USED	TX, RX and RTS pins are used by USART module
USART_ENABLE_TX_RX_USED	TX and RX pins are used by USART module

Il y a une relation avec le choix du mode de handshake :

Avec HANDSHAKE\_SIMPLEX, on obtient :

```
PLIB_USART_InitializeOperation(USART_ID_1,
    USART_RECEIVE_FIFO_ONE_CHAR,
    USART_TRANSMIT_FIFO_IDLE,
    USART_ENABLE_TX_RX_USED);
```

OU en fonction des essais successifs :

```
USART_ENABLE_TX_RX_CTS_RTS_USED);
```

Avec HANDSHAKE\_FLOWCONTROL, on obtient :

```
PLIB_USART_InitializeOperation(USART_ID_1,
    USART_RECEIVE_FIFO_ONE_CHAR,
    USART_TRANSMIT_FIFO_IDLE,
    USART_ENABLE_TX_RX_CTS_RTS_USED);
```

Avec HANDSHAKE\_NONE, on obtient la même situation que simplex:

```
PLIB_USART_InitializeOperation(USART_ID_1,  
                               USART_RECEIVE_FIFO_ONE_CHAR,  
                               USART_TRANSMIT_FIFO_IDLE,  
                               USART_ENABLE_TX_RX_USED);
```

Par contre, la fonction PLIB\_USART\_HandshakeModeSelect n'est pas appelée.

⚠ Il peut arriver, lors du passage d'un mode à l'autre, qu'on finisse par obtenir USART\_ENABLE\_TX\_RX\_CTS\_RTS\_USED avec le mode SIMPLEX, ce qui crée un conflit.

Il faut donc utiliser NONE si on ne veut pas utiliser automatiquement RTS et CTS.

### 7.3.6. CODE C GÉNÉRÉ PAR LE MHC

La fonction **DRV\_USART0\_Initialize()** est appelée par la fonction **SYS\_Initialize()** du fichier `system_init.c`. Elle est implémentée dans le fichier `drv_usart_static.c`.

Situation avec **HANDSHAKE\_NONE** :

```

SYS_MODULE_OBJ DRV_USART0_Initialize(void)
{
    uint32_t clockSource;

    /* Disable the USART module to configure it*/
    PLIB_USART_Disable (USART_ID_1);

    /* Initialize the USART based on configuration settings */
    PLIB_USART_InitializeModeGeneral(USART_ID_1,
        false, /*Auto baud*/
        false, /*LoopBack mode*/
        false, /*Auto wakeup on start*/
        false, /*IRDA mode*/
        false); /*Stop In Idle mode*/

    /* Set the line control mode */
    PLIB_USART_LineControlModeSelect(USART_ID_1,
        DRV_USART_LINE_CONTROL_8NONE1);

    /* We set the receive interrupt mode to receive an
    interrupt whenever FIFO is not empty */
    PLIB_USART_InitializeOperation(USART_ID_1,
        USART_RECEIVE_FIFO_ONE_CHAR,
        USART_TRANSMIT_FIFO_IDLE,
        USART_ENABLE_TX_RX_USED);

    /* Get the USART clock source value*/
    clockSource =
        SYS_CLK_PeripheralFrequencyGet (CLK_BUS_PERIPHERAL_1);

    /* Set the baud rate and enable the USART */
    PLIB_USART_BaudSetAndEnable(USART_ID_1, clockSource,
        57600); /*Desired Baud rate value*/

    /* Clear the interrupts to be on the safer side*/
    SYS_INT_SourceStatusClear(INT_SOURCE_USART_1_TRANSMIT);
    SYS_INT_SourceStatusClear(INT_SOURCE_USART_1_RECEIVE);
    SYS_INT_SourceStatusClear(INT_SOURCE_USART_1_ERROR);

    /* Enable the error interrupt source */
    SYS_INT_SourceEnable(INT_SOURCE_USART_1_ERROR);

    /* Enable the Receive interrupt source */
    SYS_INT_SourceEnable(INT_SOURCE_USART_1_RECEIVE);

    /* Return the driver instance value*/
    return (SYS_MODULE_OBJ)DRV_USART_INDEX_0;
}

```

On constate la correspondance entre les éléments configurés et les fonctions de configurations de la PLIB\_USART.

Au niveau de la configuration de l'interruption, on constate qu'il y a 3 sources d'interruption pour le même vecteur.

☞ Il sera donc nécessaire de traiter dans la même interruption la réception et la transmission, ainsi que la situation d'erreur.

### 7.3.7. MODIFICATIONS NÉCESSAIRES

Pour obtenir une configuration adaptée au traitement des interruptions de l'USART, il faut effectuer la modification suivante au niveau du fichier drv\_usart\_static.c.

Situation après génération :

```
PLIB_USART_InitializeOperation(USART_ID_1,
                               USART_RECEIVE_FIFO_ONE_CHAR,
                               USART_TRANSMIT_FIFO_IDLE,
                               USART_ENABLE_TX_RX_USED);
```

Situation après modification :

```
PLIB_USART_InitializeOperation(USART_ID_1,
                               USART_RECEIVE_FIFO_ONE_CHAR,
                               USART_TRANSMIT_FIFO_EMPTY,
                               USART_ENABLE_TX_RX_USED);
```

☞ On conserve la configuration par défaut USART\_RECEIVE\_FIFO\_ONE\_CHAR.

Liste des modes pour TX interrupt :

```
typedef enum {
    USART_TRANSMIT_FIFO_NOT_FULL = 0x00,
    USART_TRANSMIT_FIFO_IDLE = 0x01,
    USART_TRANSMIT_FIFO_EMPTY = 0x02
} USART_TRANSMIT_INTR_MODE;
```

Liste des modes pour RX interrupt :

```
typedef enum {
    USART_RECEIVE_FIFO_3B4FULL = 0x02,
    USART_RECEIVE_FIFO_HALF_FULL = 0x01,
    USART_RECEIVE_FIFO_ONE_CHAR = 0x00
} USART_RECEIVE_INTR_MODE;
```

## 7.4. FONCTIONS DE L'EMETTEUR

On dispose d'un certain nombre de fonctions liées à l'émission.

	Name	Description
•	<b>PLIB_USART_Transmitter9BitsSend</b>	Data to be transmitted in the byte mode with the 9th bit.
•	<b>PLIB_USART_TransmitterBreakSend</b>	Transmits the break character.
•	<b>PLIB_USART_TransmitterBreakSendIsComplete</b>	Returns the status of the break transmission
•	<b>PLIB_USART_TransmitterBufferIsFull</b>	Gets the transmit buffer full status.
•	<b>PLIB_USART_TransmitterByteSend</b>	Data to be transmitted in the Byte mode.
•	<b>PLIB_USART_TransmitterDisable</b>	Disables the specific USART module transmitter.
•	<b>PLIB_USART_TransmitterEnable</b>	Enables the specific USART module transmitter.
•	<b>PLIB_USART_TransmitterIdleIsLowDisable</b>	Disables the Transmit Idle Low state.
•	<b>PLIB_USART_TransmitterIdleIsLowEnable</b>	Enables the Transmit Idle Low state.
•	<b>PLIB_USART_TransmitterInterruptModeSelect</b>	Sets the USART transmitter interrupt mode.
•	<b>PLIB_USART_TransmitterIsEmpty</b>	Gets the transmit shift register empty status.
•	<b>PLIB_USART_TransmitterAddressGet</b>	Returns the address of the USART TX register

Voici la description des fonctions les plus utiles.

### 7.4.1. LA FONCTION PLIB\_USART\_TRANSMITTERBYTESEND

La fonction **PLIB\_USART\_TransmitterByteSend** permet de transmettre un octet.

```
void PLIB_USART_TransmitterByteSend(USART_MODULE_ID index, int8_t data);
```

### 7.4.2. LA FONCTION PLIB\_USART\_TRANSMITTERBUFFERISFULL

La fonction **PLIB\_USART\_TransmitterBufferIsFull** permet de connaitre la situation du tampon d'émission.

```
bool PLIB_USART_TransmitterBufferIsFull(USART_MODULE_ID index);
```

Si true, le buffer est plein. Avec false le buffer n'est pas plein, il y a au moins la place pour un caractère.

## 7.5. FONCTIONS DU RECEPTEUR

On dispose d'un certain nombre de fonctions liées à la réception.

	Name	Description
≡	<b>PLIB_USART_ReceiverAddressAutoDetectDisable</b>	Disables the automatic Address Detect mode.
≡	<b>PLIB_USART_ReceiverAddressAutoDetectEnable</b>	Setup the automatic Address Detect mode.
≡	<b>PLIB_USART_ReceiverAddressDetectDisable</b>	Enables the Address Detect mode.
≡	<b>PLIB_USART_ReceiverAddressDetectEnable</b>	Enables the Address Detect mode.
≡	<b>PLIB_USART_ReceiverAddressIsReceived</b>	Checks and return if the data received is an address.
≡	<b>PLIB_USART_ReceiverByteReceive</b>	Data to be received in the Byte mode.
≡	<b>PLIB_USART_ReceiverDataIsAvailable</b>	Identifies if the receive data is available for the specified USART module.
≡	<b>PLIB_USART_ReceiverDisable</b>	Disables the USART receiver.
≡	<b>PLIB_USART_ReceiverEnable</b>	Enables the USART receiver.
≡	<b>PLIB_USART_ReceiverFramingErrorHasOccurred</b>	Gets the framing error status.
≡	<b>PLIB_USART_ReceiverIdleStateLowDisable</b>	Disables receive polarity inversion.
≡	<b>PLIB_USART_ReceiverIdleStateLowEnable</b>	Enables receive polarity inversion.
≡	<b>PLIB_USART_ReceiverInterruptModeSelect</b>	Sets the USART receiver FIFO level.
≡	<b>PLIB_USART_ReceiverIsIdle</b>	Identifies if the receiver is idle.
≡	<b>PLIB_USART_ReceiverOverrunErrorClear</b>	Clears a USART v verrun error.
≡	<b>PLIB_USART_ReceiverOverrunHasOccurred</b>	Identifies if there was a receiver overrun error.
≡	<b>PLIB_USART_ReceiverParityErrorHasOccurred</b>	Gets the parity error status.
≡	<b>PLIB_USART_ReceiverAddressGet</b>	Returns the address of the USART RX register
≡	<b>PLIB_USART_Receiver9BitsReceive</b>	Data to be received in the byte mode with the 9th bit.

Voici la description des fonctions les plus utiles.

### 7.5.1. LA FONCTION PLIB\_USART\_RECEIVERBYTE RECEIVE

La fonction **PLIB\_USART\_ReceiverByteReceive** permet de lire un byte des données reçu par l'USART et stocké dans le tampon de réception.

```
int8_t PLIB_USART_ReceiverByteReceive(USART_MODULE_ID index);
```

↳ Il faut au préalable s'assurer qu'un byte est disponible.

### 7.5.2. LA FONCTION PLIB\_USART\_RECEIVERDATAISAVAILABLE

La fonction **PLIB\_USART\_ReceiverDataIsAvailable** permet de savoir si une donnée est disponible.

```
bool PLIB_USART_ReceiverDataIsAvailable(USART_MODULE_ID index);
```

#### 7.5.2.1. PLIB\_USART\_RECEIVERDATAISAVAILABLE, EXEMPLE

Cet exemple repris de la réponse à l'interruption de réception montre comment exploiter le tampon de réception.

```
int8_t c;

while (PLIB_USART_ReceiverDataIsAvailable(USART_ID_1))
{
    c = PLIB_USART_ReceiverByteReceive(USART_ID_1);
    PutCharInFifo (&descrFifoRX, c);
}
```

### 7.5.3. FONCTION PLIB\_USART\_RECEIVEROVERRUNERRORCLEAR

La fonction **PLIB\_USART\_ReceiverOverrunErrorClear** permet de supprimer une erreur d'overrun. L'appel de cette fonction efface les données reçues.

```
void PLIB_USART_ReceiverOverrunErrorClear(USART_MODULE_ID index);
```

## 7.6. FONCTIONS GENERALES DE L'USART

Voici les nombreuses fonctions générales (ne s'appliquent pas toutes pour le suivant le modèle de microcontrôleur) :

Name	Description
<b>PLIB_USART_Disable</b>	Disables the specific USART module
<b>PLIB_USART_Enable</b>	Enables the specific USART module.
<b>PLIB_USART_HandshakeModeSelect</b>	Sets the data flow configuration.
<b>PLIB_USART_IrDADisable</b>	Disables the IrDA encoder and decoder.
<b>PLIB_USART_IrDAEnable</b>	Enables the IrDA encoder and decoder.
<b>PLIB_USART_LineControlModeSelect</b>	Sets the data flow configuration.
<b>PLIB_USART_LoopbackDisable</b>	Disables Loopback mode.
<b>PLIB_USART_LoopbackEnable</b>	Enables Loopback mode.
<b>PLIB_USART_OperationModeSelect</b>	Configures the operation mode of the USART module.
<b>PLIB_USART_StopInIdleDisable</b>	Disables the Stop in Idle mode (the USART module continues operation when the device is in Idle mode).
<b>PLIB_USART_StopInIdleEnable</b>	Discontinues operation when the device enters Idle mode.
<b>PLIB_USART_WakeOnStartDisable</b>	Disables the wake-up on start bit detection feature during Sleep mode.
<b>PLIB_USART_WakeOnStartEnable</b>	Enables the wake-up on start bit detection feature during Sleep mode.
<b>PLIB_USART_WakeOnStartIsEnabled</b>	Gets the state of the sync break event completion.
<b>PLIB_USART_ErrorGet</b>	Return the status of all errors in the specified USART module.
<b>PLIB_USART_InitializeModeGeneral</b>	Enables or disables general features of the USART module.
<b>PLIB_USART_InitializeOperation</b>	Configures the Receive and Transmit FIFO interrupt levels and the hardware lines to be used by the module.
<b>PLIB_USART_AddressGet</b>	Gets the address for the Address Detect mode.
<b>PLIB_USART_AddressMaskGet</b>	Gets the address mask for the Address Detect mode.
<b>PLIB_USART_AddressMaskSet</b>	Sets the address mask for the Address Detect mode.
<b>PLIB_USART_AddressSet</b>	Sets the address for the Address Detect mode.
<b>PLIB_USART_ModuleIsBusy</b>	Returns the USART module's running status.
<b>PLIB_USART_RunInOverflowDisable</b>	Disables the Run in overflow condition mode.
<b>PLIB_USART_RunInOverflowEnable</b>	Enables the USART module to continue to operate when an overflow error condition has occurred.
<b>PLIB_USART_RunInOverflowIsEnabled</b>	Gets the status of the Run in Overflow condition.
<b>PLIB_USART_RunInSleepModeDisable</b>	Turns off the USART module's BRG clock during Sleep mode.
<b>PLIB_USART_RunInSleepModeEnable</b>	Allows the USART module's BRG clock to run when the device enters Sleep mode.
<b>PLIB_USART_RunInSleepModelsEnabled</b>	Gets the status of Run in Sleep mode.

Voici la description des fonctions les plus utiles.

### 7.6.1. LA FONCTION PLIB\_USART\_DISABLE

La fonction **PLIB\_USART\_Disable** permet de désactiver l'USART. Cela a pour effet de vider les tampons d'émission et de réception.

```
void PLIB_USART_Disable(USART_MODULE_ID index);
```

### 7.6.2. LA FONCTION PLIB\_USART\_ENABLE

La fonction **PLIB\_USART\_Enable** permet d'activer l'USART. Cela a pour effet que les broches RX et TX sont gérées par l'USART.

```
void PLIB_USART_Enable(USART_MODULE_ID index);
```

### 7.6.3. LA FONCTION PLIB\_USART\_ERRORGET

La fonction **PLIB\_USART\_ErrorGet** permet d'obtenir la situation des bits d'erreurs.

```
USART_ERROR PLIB_USART_ErrorsGet(USART_MODULE_ID index);
```

Le type USART\_ERROR est défini ainsi :

```
typedef enum {
    USART_ERROR_NONE = 0x00,
    USART_ERROR_RECEIVER_OVERRUN = 0x01,
    USART_ERROR_FRAMING = 0x02,
    USART_ERROR_PARITY = 0x04
} USART_ERROR;
```

### 7.6.4. EXEMPLE GESTION DES ERREURS

L'exemple repris de la réponse à l'interruption de réception montre le traitement des erreurs.

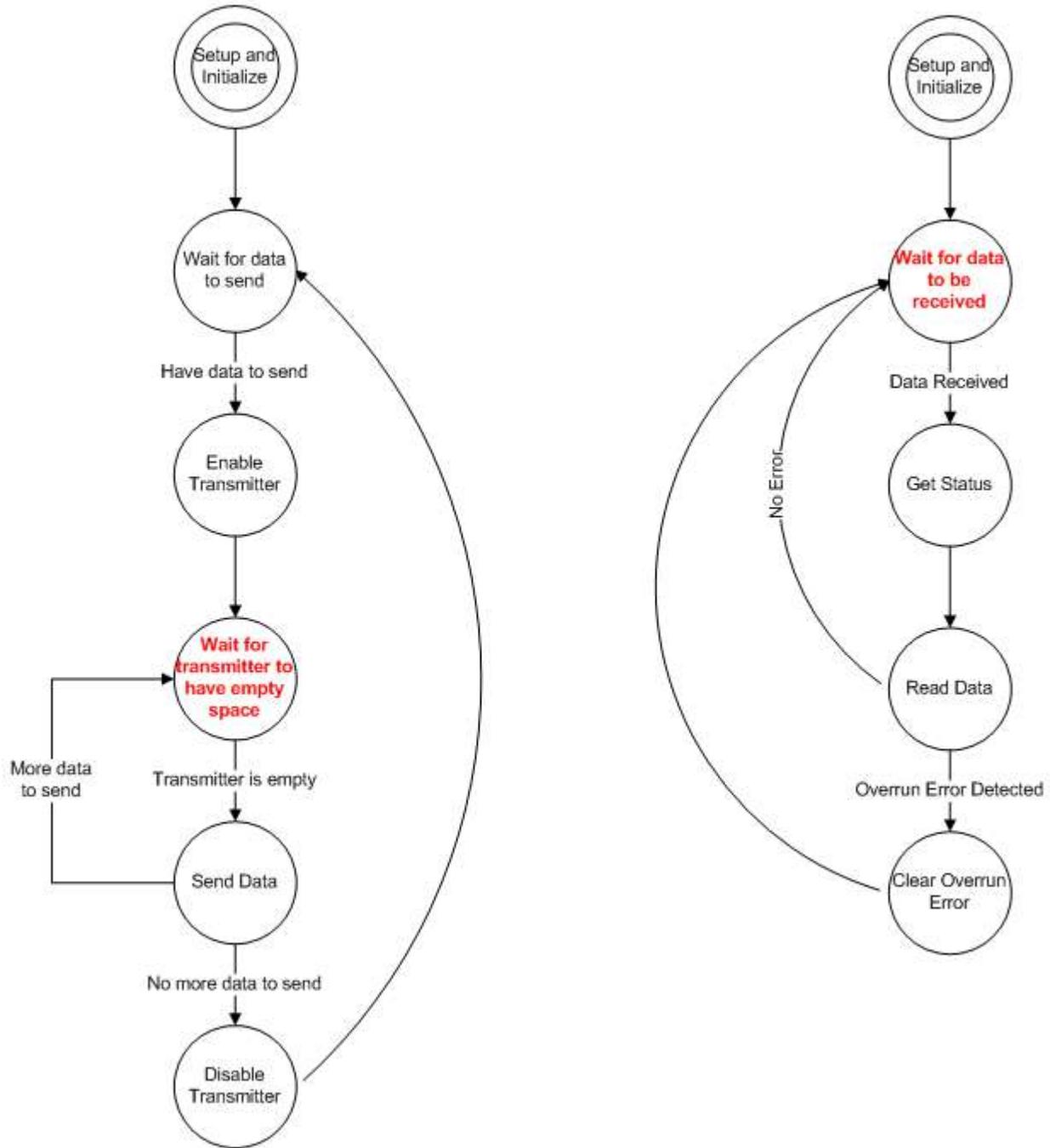
```
USART_ERROR UsartStatus = PLIB_USART_ErrorsGet(USART_ID_1);

if ( (UsartStatus & (USART_ERROR_PARITY |
        USART_ERROR_FRAMING |
        USART_ERROR_RECEIVER_OVERRUN)) == 0) {
    // OK traitement de réception possible
} else {
    // Suppression des erreurs
    // La lecture des erreurs les efface sauf pour overrun
    if ( (UsartStatus & USART_ERROR_RECEIVER_OVERRUN) ==
        USART_ERROR_RECEIVER_OVERRUN) {
        PLIB_USART_ReceiverOverrunErrorClear(USART_ID_1);
    }
}
```

## 7.7. REALISATION ISR DE L'USART

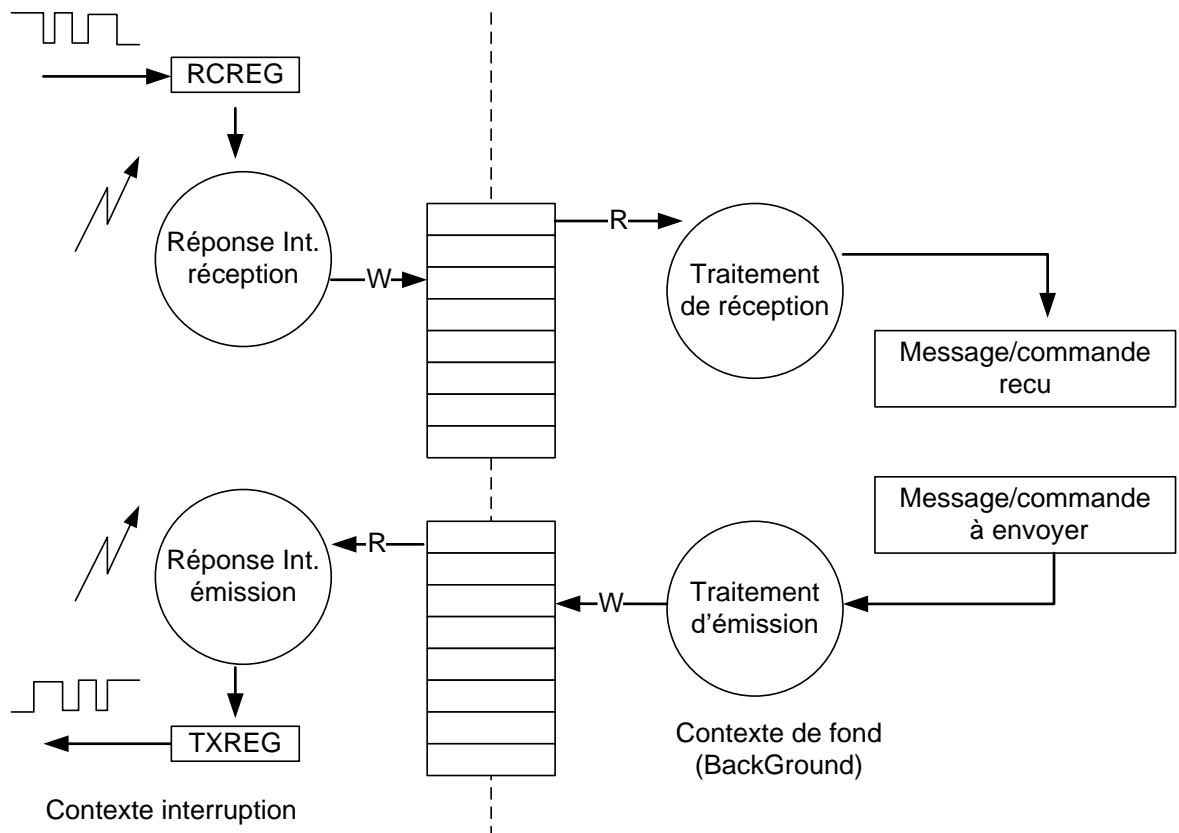
### 7.7.1. DIAGRAMME DE PRINCIPE

Dans la documentation Harmony, on trouve les 2 diagrammes d'état suivants :



### 7.7.2. CONCEPT ÉMISSION - RÉCEPTION AVEC FIFO

Le diagramme ci-dessous illustre le principe du traitement de la réception et de l'émission sous interruption en utilisant des FIFO software.



### 7.7.3. ISR GÉNÉRÉE PAR LE MHC

Dans la réponse à l'interruption USART (partagée entre émission, réception et erreur), Harmony génère 3 appels à des fonctions qui font partie du driver :

```
void __ISR(_UART_1_VECTOR, ipl5AUTO)
          _IntHandlerDrvUsartInstance0(void)
{
    DRV_USART_TasksTransmit(sysObj.drvUsart0);
    DRV_USART_TasksReceive(sysObj.drvUsart0);
    DRV_USART_TasksError(sysObj.drvUsart0);
}
```

### 7.7.3.1. LA FONCTION DRV\_USART\_TASKSTXMIT

Comme on peut le constater, cette fonction teste s'il s'agit d'une interruption de transmission, mais à part la mise à zéro du flag elle ne fait rien ! C'est à l'utilisateur d'implémenter son traitement.

```
void DRV_USART0_Taskstxmit(void)
{
    /* This is the USART Driver Transmit tasks routine.
       In this function, the driver checks if a transmit
       interrupt is active and performs respective action*/

    /* Reading the transmit interrupt flag */
    if(SYS_INT_SourceStatusGet
        (INT_SOURCE_USART_1_TRANSMIT))
    {
        /* Disable the interrupt,
           to avoid calling ISR continuously*/
        SYS_INT_SourceDisable(INT_SOURCE_USART_1_TRANSMIT);

        /* Clear up the interrupt flag */
        SYS_INT_SourceStatusClear
            (INT_SOURCE_USART_1_TRANSMIT);
    }
}
```

### 7.7.3.2. LA FONCTION DRV\_USART\_TASKSRECEIVE

Comme on peut le constater, cette fonction teste s'il s'agit d'une interruption de réception, mais à part la mise à zéro du flag elle ne fait rien ! C'est à l'utilisateur d'implémenter son traitement.

```
void DRV_USART0_Tasksrceive(void)
{
    /* This is the USART Driver Receive tasks routine.
       If the receive interrupt flag is set,
       the tasks routines are executed.
    */

    /* Reading the receive interrupt flag */
    if(SYS_INT_SourceStatusGet(INT_SOURCE_USART_1_RECEIVE))
    {

        /* Clear up the interrupt flag */
        SYS_INT_SourceStatusClear
            (INT_SOURCE_USART_1_RECEIVE);
    }
}
```

### 7.7.3.3. LA FONCTION DRV\_USART\_TASKSError

Cette fonction teste s'il s'agit d'une interruption d'erreur, elle effectue un traitement partiel des erreurs.

```
void DRV_USART0_TasksError(void)
{
    /* This is the USART Driver Error tasks routine.
     * In this function, the driver checks if an error
     * interrupt has occurred.
     * If so the error condition is cleared. */

    /* Reading the error interrupt flag */
    if(SYS_INT_SourceStatusGet(INT_SOURCE_USART_1_ERROR))
    {
        /* This means an error has occurred */
        if(PLIB_USART_ReceiverOverrunHasOccurred
            (USART_ID_1))
        {
            PLIB_USART_ReceiverOverrunErrorClear
                (USART_ID_1);
        }

        /* Clear up the error interrupt flag */
        SYS_INT_SourceStatusClear
            (INT_SOURCE_USART_1_ERROR);
    }
}
```

☝ Nous n'utiliserons pas ces fonctions mais nous reprenons certains éléments dans l'élaboration d'une routine de réponse unique.

#### 7.7.4. ISR USART, RÉALISATION PRATIQUE

Voici un exemple pratique de réalisation de la routine de réponse à l'interruption de l'USART. Pour bien comprendre les traitements, il faut rappeler que nous avons configuré la relation entre les FIFOs hardware et les interruptions de la manière suivante :

```
PLIB_USART_TransmitterInterruptModeSelect(USART_ID_1,
                                         USART_TRANSMIT_FIFO_EMPTY);
PLIB_USART_ReceiverInterruptModeSelect(USART_ID_1,
                                         USART_RECEIVE_FIFO_ONE_CHAR);
```

☞ La réponse est prévue pour supporter un changement du **ReceiverInterruptModeSelect** à HALF\_FULL ou 3B4FULL, d'où une boucle dans la réception.

```
void __ISR(_UART_1_VECTOR, ip15AUTO)
          _IntHandlerDrvUsartInstance0(void)
{
    uint8_t freeSize, TXsize;
    int8_t c;
    int8_t i_cts = 0;
    BOOL TxBuffFull;

    USART_ERROR UsartStatus;

    // Marque début interruption avec Led3
    LED3_W = 1;

    // Is this an RX interrupt ?
    if ( PLIB_INT_SourceFlagGet(INT_ID_0,
                                INT_SOURCE_USART_1_RECEIVE) &&
        PLIB_INT_SourceIsEnabled(INT_ID_0,
                                INT_SOURCE_USART_1_RECEIVE) ) {

        // Teste si erreur parité, framing ou overrun
        UsartStatus = PLIB_USART_ErrorsGet(USART_ID_1);

        //pas d'erreur ?
        if ( (UsartStatus & (USART_ERROR_PARITY |
                               USART_ERROR_FRAMING |
                               USART_ERROR_RECEIVER_OVERRUN)) == 0) {
            // transfert dans le FIFO software
            // de tous les char reçus
            while (PLIB_USART_ReceiverDataIsAvailable
                               (USART_ID_1))
            {
                c = PLIB_USART_ReceiverByteReceive
                               (USART_ID_1);
                PutCharInFifo ( &descrFifoRX, c);
            }
        }
    }
}
```

Voir note 1  
ci-dessous

```
    LED4_W = !LED4_R; // Toggle Led4
    // buffer is empty, clear interrupt flag
    PLIB_INT_SourceFlagClear(INT_ID_0,
                                INT_SOURCE_USART_1_RECEIVE);
} else {
    // Suppression des erreurs
    // La lecture des erreurs les efface
    // sauf pour overrun
    if ( (UsartStatus &
          USART_ERROR_RECEIVER_OVERRUN) ==
        USART_ERROR_RECEIVER_OVERRUN) {
        PLIB_USART_ReceiverOverrunErrorClear
            (USART_ID_1);
    }
}

freeSize = GetWriteSpace ( &descrFifoRX );
// A cause du cas un int pour 6 char % de 8 = 6
if (freeSize <= 6) {
    // Contrôle de flux : demande stop émission
    RS232 RTS = 1;
}
}
} // end if RX

// Is this an TX interrupt ?
if ( PLIB_INT_SourceFlagGet(INT_ID_0,
                             INT_SOURCE_USART_1_TRANSMIT) &&
     PLIB_INT_SourceIsEnabled(INT_ID_0,
                               INT_SOURCE_USART_1_TRANSMIT) ) {

    TXsize = GetReadSize (&descrFifoTX);
    // i_cts = input(RS232_CTS);
    // On vérifie 3 conditions :
    // Si CTS = 0 (autorisation d'émettre)
    // Si il y a un caractère à émettre
    // Si le txreg est bien disponible
    i_cts = RS232_CTS;
    // Il est possible de déposer un caractère
    // tant que le tampon n'est pas plein
    TxBuffFull =
        PLIB_USART_TransmitterBufferIsFull(USART_ID_1);
    if ( (i_cts == 0) && ( TXsize > 0 ) &&
        TxBuffFull == false ) {

        do {
            GetCharFromFifo(&descrFifoTX, &c);
            PLIB_USART_TransmitterByteSend
                (USART_ID_1, c);
            i_cts = RS232_CTS;
            TXsize = GetReadSize (&descrFifoTX);
        }
    }
}
```

```

TxBuffFull =
    PLIB_USART_TransmitterBufferIsFull
        (USART_ID_1);

} while ( (i_cts == 0) && ( TXsize > 0 ) &&
          TxBuffFull==false );

LED5_W = !LED5_R; // Toggle Led5

// Clear the TX interrupt Flag
// (Seulement après TX)
PLIB_INT_SourceFlagClear(INT_ID_0,
                          INT_SOURCE_USART_1_TRANSMIT);
if (TXsize == 0) {
    // disable TX interrupt
    // (pour éviter une int inutile)
    PLIB_INT_SourceDisable(INT_ID_0,
                           INT_SOURCE_USART_1_TRANSMIT);
}
} else {
    // disable TX interrupt
    PLIB_INT_SourceDisable(INT_ID_0,
                           INT_SOURCE_USART_1_TRANSMIT);
}

// Marque fin interruption avec Led3
LED3_W = 0;
} // end __ISR Usart 1

```

#### Note 1 :

Dans la partie réception de l'ISR, l'utilisation de la condition `PLIB_USART_ReceiverDataIsAvailable()` vérifie que des données reçues soient disponibles. Mais aucune vérification de la place disponible dans FIFO software n'est faite avant d'y copier ces données.

Cette manière de procéder peut être acceptable si, comme ici, un contrôle de flux est implémenté, ou si le FIFO est prévu avec suffisamment de place pour stocker le flux de données entrant dans le pire cas.

### 7.7.1. REMARQUE SUR LA RÉALISATION PRATIQUE

En testant le flag d'interruption et le flag d'autorisation, il est possible de déterminer la source de l'interruption (réception, émission ou erreur).

#### 7.7.1.1. TRAITEMENT DE LA RECEPTION

Lors de la réception, on utilise une boucle pour vider le tampon dans le but de supporter les différents modes. Le contenu du tampon est transféré dans le FIFO de réception software.

Donc nous utilisons une boucle qui prend un caractère tant que disponible. Il faut ajouter le traitement d'une éventuelle erreur.

☞ Le flag d'interruption de réception ne doit être mis à zéro que lorsque le tampon de réception est vide.

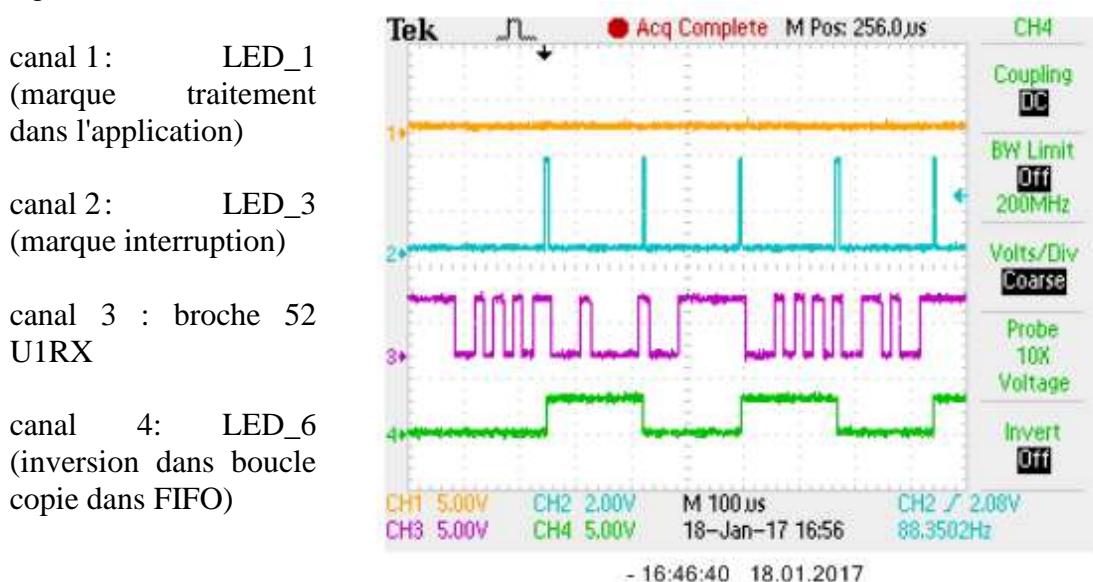
#### 7.7.1.2. TRAITEMENT DE L'EMISSION

Le traitement d'émission est un peu particulier car l'interruption de transmission se produit lorsqu'il est possible d'émettre. Il est donc nécessaire de bloquer cette interruption s'il n'y a plus rien à transmettre. C'est l'application qui est responsable de d'activer l'interruption lorsqu'il y a un message dans le FIFO software de transmission.

Réalisation d'une boucle qui essaie de remplir au maximum le tampon d'émission.

### 7.7.2. TEST DU MÉCANISME DE RÉCEPTION

Voici l'observation du comportement de l'interruption de l'USART en relation avec le signal RX :



On observe qu'il y a 5 interruptions car le message est de 5 caractères. L'interruption se produit après la réception d'un caractère. Le canal 4 nous signale qu'il y a bien copie dans le FIFO à chaque interruption.

### 7.7.3. TEST DU MÉCANISME D'ÉMISSION

Voici l'observation du comportement de l'interruption de l'USART en relation avec le signal TX :

canal 1: LED\_1  
(marque traitement dans l'application)

canal 2: LED\_3  
(marque interruption)

canal 3 : broche 53 U1TX

canal 4: LED\_7  
(inversion dans boucle écriture dans tampon émission)



On constate que l'interruption (unique) a lieu à la fin du traitement de l'application (action SendMessage)

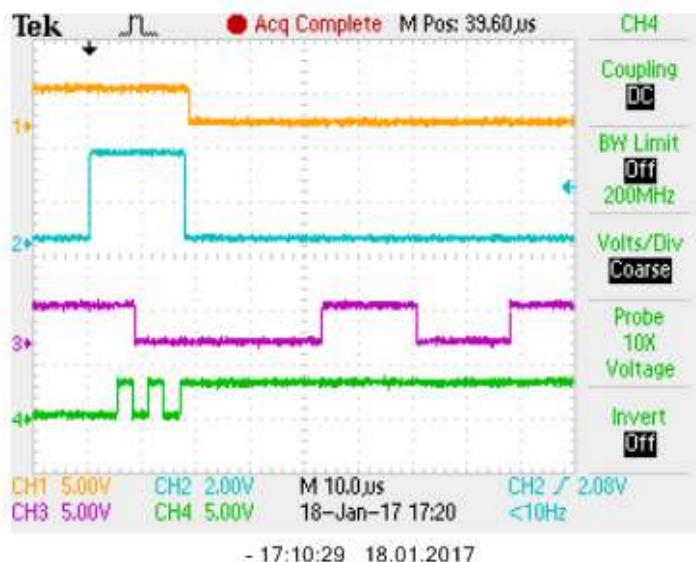
En changeant l'échelle, on peut observer la copie des 5 caractères du message dans le tampon d'émission (5 flancs sur le signal).

canal 1: LED\_1  
(marque traitement dans l'application)

canal 2: LED\_3  
(marque interruption)

canal 3 : broche 53 U1TX

canal 4: LED\_7  
(inversion dans boucle écriture dans tampon émission)



On voit ici l'efficacité du mécanisme d'émission : Avec une seule interruption on remplit le tampon d'émission et les 5 caractères sont émis par l'USART.

## 7.8. HISTORIQUE DES VERSIONS

### 7.8.1. V1.0 AVRIL 2014

Création du document après une laborieuse mise au point de l'application utilisée pour tester.

### 7.8.2. V1.5 JANVIER 2015

Adaptation du document aux nouvelles PLIB\_USART et PLIB\_INT introduite avec Harmony 1.00.

### 7.8.3. V1.6 JANVIER 2016

Adaptation du document aux détails du code généré lié au MPLABX 3.10 et Harmony 1.06.

### 7.8.4. V1.7 JANVIER 2017

Adaptation du document aux détails du code généré lié au MPLABX 3.40 et Harmony 1.08. Remarque : forte évolution du driver USART.

### 7.8.1. V1.8 NOVEMBRE 2017

Reprise et relecture par SCA.

### 7.8.2. V1.81 JANVIER 2019

Mise à jour figure configuration USART via MHC.

### 7.8.3. V1.82 NOVEMBRE 2021

Précision code ISR réception.

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

# **Chapitre 8**

**Gestion du bus I2C,  
fonctions à machine d'état**

**❖ T.P. PIC32MX**

**Christian HUBER (CHR)  
Serge CASTOLDI (SCA)  
Version v1.71 février 2022**



## CONTENU DU CHAPITRE 8

<b>8. Gestion par machine d'état du bus I2C PIC32MX</b>	<b>8-1</b>
<b>8.1. Réalisation du bus I2C avec le PIC32MX95F512L</b>	<b>8-1</b>
8.1.1. Composant I2C du Kit PIC32MX795F512L	8-1
8.1.1.1. Sonde de température LM92	8-1
8.1.1.2. Convertisseur 1-wire DS2482S-100	8-2
8.1.1.3. RTC, MAC adresse et EEPROM	8-2
8.1.2. Les fonctions de la PLIB_I2C	8-3
8.1.2.1. Vue d'ensemble des fonctions de la PLIB_I2C	8-3
<b>8.2. Projet avec driver I2C</b>	<b>8-5</b>
8.2.1. Création du projet	8-5
8.2.2. Choix des Eléments	8-5
8.2.2.1. Choix BSP	8-5
8.2.2.2. Device configuration	8-6
8.2.2.3. Timer driver	8-6
8.2.2.4. I2C driver	8-7
8.2.3. Observation des résultats	8-7
8.2.3.1. Organisation	8-7
<b>8.3. Réalisation de fonctions I2C à machine d'état</b>	<b>8-8</b>
8.3.1. Action au niveau de l'application	8-8
8.3.1.1. Ajout donnée application pour LM92 dans struct APP_DATA	8-8
8.3.1.2. traitement dans case APP_STATE_INIT	8-9
8.3.1.3. traitement dans case APP_STATE_SERVICE_TASKS	8-9
8.3.2. Action au niveau de l'interruption du Timer1	8-10
8.3.3. Initialisation dans le fichier system_init.c	8-10
8.3.4. Contenu du fichier Mc32gestI2cLM92_SM.h	8-11
8.3.5. Contenu du fichier Mc32gestI2cLM92_SM.c	8-12
8.3.5.1. La fonction Lm92SmInit	8-12
8.3.5.2. La fonction I2C_LM92_SM_Init	8-12
8.3.5.3. La fonction I2C_LM92_SM_Restart	8-13
8.3.5.4. La fonction I2C_LM92_SM_IsReady	8-13
8.3.5.5. La fonction I2C_LM92_SM_Execute	8-13
8.3.5.6. La fonction I2C_LM92_SM_GetTemp	8-17
8.3.5.7. La fonction I2C_LM92_SM_GetTempMilli	8-17
8.3.5.8. La fonction I2C_LM92_SM_GetRawTemp	8-17
8.3.6. Contenu du fichier Mc32_I2cUtil_SM.h	8-18
8.3.7. Contenu du fichier Mc32_I2cUtil_SM.c	8-19
8.3.7.1. La fonction I2C_SM_init	8-19
8.3.7.2. La fonction I2C_SM_begin	8-19
8.3.7.3. La fonction I2C_SM_isReady	8-19
8.3.7.4. La fonction I2C_SM_start	8-20
8.3.7.5. La fonction I2C_SM_reStart	8-21
8.3.7.6. La fonction I2C_SM_write	8-22
8.3.7.7. La fonction I2C_SM_read	8-23
8.3.7.8. La fonction I2C_SM_stop	8-24
8.3.8. Contenu du fichier drv_i2c_static.h	8-25

8.3.9.	Contenu du fichier drv_i2c_static.c	8-26
8.3.9.1.	La fonction DRV_I2C0_Initialize créée	8-27
8.3.9.2.	La fonction DRV_I2C0_Initialize modifiée	8-27
8.3.9.3.	La fonction DRV_I2C0_DeInitialize	8-27
8.3.9.4.	La fonction DRV_I2C0_SetUpByteRead	8-28
8.3.9.5.	La fonction DRV_I2C0_WaitForReadByteAvailable	8-28
8.3.9.6.	La fonction DRV_I2C0_ByteRead	8-28
8.3.9.7.	La fonction DRV_I2C0_ByteWrite	8-29
8.3.9.8.	La fonction DRV_I2C0_WaitForByteWriteToComplete	8-29
8.3.9.9.	La fonction DRV_I2C0_WriteByteAcknowledged	8-29
8.3.9.10.	La fonction DRV_I2C0_BaudRateSet	8-30
8.3.9.11.	La fonction DRV_I2C0_MasterBusIdle	8-30
8.3.9.12.	La fonction DRV_I2C0_MasterStart	8-31
8.3.9.13.	La fonction DRV_I2C0_WaitForStartComplete	8-31
8.3.9.14.	La fonction DRV_I2C0_MasterRestart	8-31
8.3.9.15.	La fonction DRV_I2C0_MasterStop	8-32
8.3.9.16.	La fonction DRV_I2C0_WaitForStopComplete	8-32
8.3.9.17.	La fonction DRV_I2C0_MasterACKSend	8-32
8.3.9.18.	La fonction DRV_I2C0_MasterNACKSend	8-33
8.3.9.19.	La fonction DRV_I2C0_WaitForACKOrNACKComplete	8-33
8.3.10.	Contrôle de fonctionnement des fonctions SM	8-34
8.3.10.1.	Vue d'ensemble cycle 1 ms	8-34
8.3.10.2.	Vue d'ensemble cycle 100 us	8-34
8.3.10.3.	Détail début transaction cycle 100 us	8-35
8.3.11.	Conclusion sur les fonctions SM	8-35
<b>8.4.</b>	<b>Utilisation en machine d'état de plusieurs composants</b>	<b>8-36</b>
8.4.1.	Illustration du séquencement dans le temps	8-36
8.4.2.	Principe réalisation du séquencement dans le temps	8-36
<b>8.5.</b>	<b>Localisation des fichiers I2C</b>	<b>8-38</b>
<b>8.6.</b>	<b>Conclusion</b>	<b>8-39</b>
<b>8.7.</b>	<b>Historique des versions</b>	<b>8-39</b>
8.7.1.	Version 1.5 mai 2015	8-39
8.7.2.	Version 1.6 mai 2016	8-39
8.7.3.	Version 1.7 avril 2017	8-39

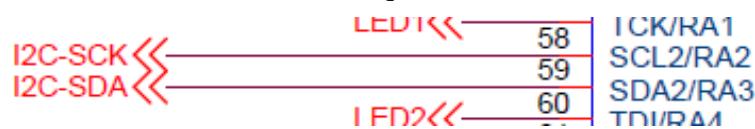
## 8. GESTION PAR MACHINE D'ÉTAT DU BUS I2C PIC32MX

Ce chapitre a pour objectif de découvrir le driver I2C fourni par Harmony.

L'objectif est de disposer de fonctions avec machine d'état, permettant un traitement cyclique et concurrent de composants I2C, ceci en utilisant au mieux les éléments fournis par Harmony.

### 8.1. RÉALISATION DU BUS I2C AVEC LE PIC32MX95F512L

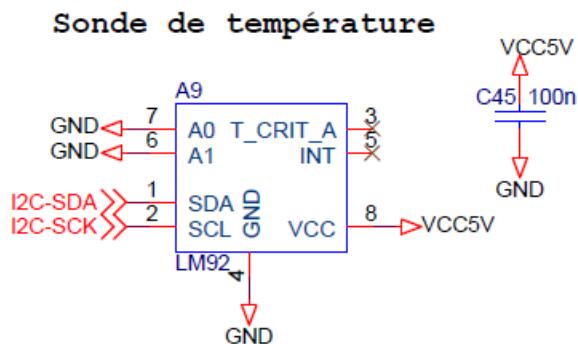
Le PIC32MX dispose de modules spécifiques pour la gestion du bus I2C. Dans le cadre du kit PIC32MX795F512L c'est le module I2C no 2 qui est utilisé.



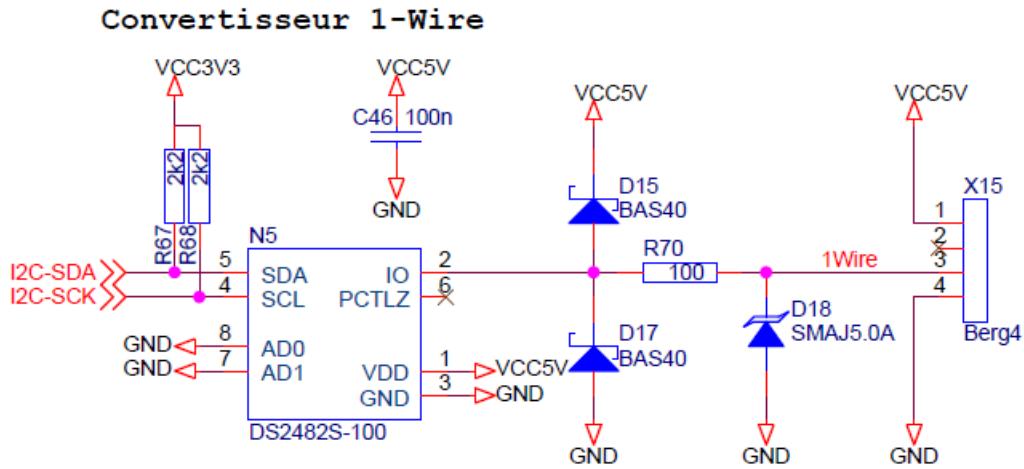
#### 8.1.1. COMPOSANT I2C DU KIT PIC32MX795F512L

Il y a trois composants I2C sur le Kit PIC32MX795F512L.

##### 8.1.1.1. SONDE DE TEMPÉRATURE LM92



### 8.1.1.2. CONVERTISSEUR 1-WIRE DS2482S-100

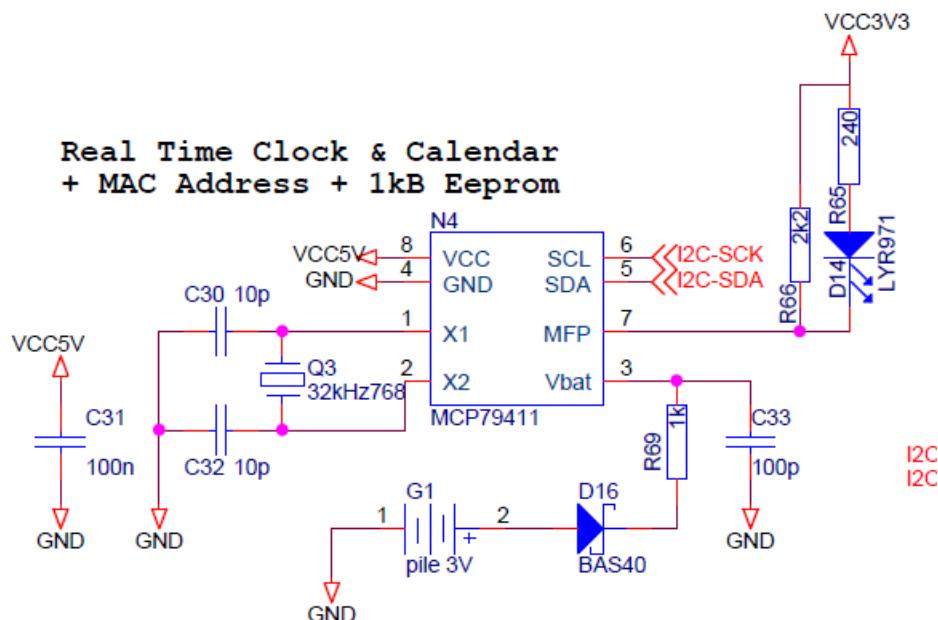


On remarque la paire de résistances de pullup pour le bus I2C.

### 8.1.1.3. RTC, MAC ADRESSE ET EEPROM

Le MCP79411 intègre :

- un Real Time Clock/Calendar,
- une adresse MAC préprogrammée d'usine,
- une EEPROM de 1 Kbits, soit 128 octets.

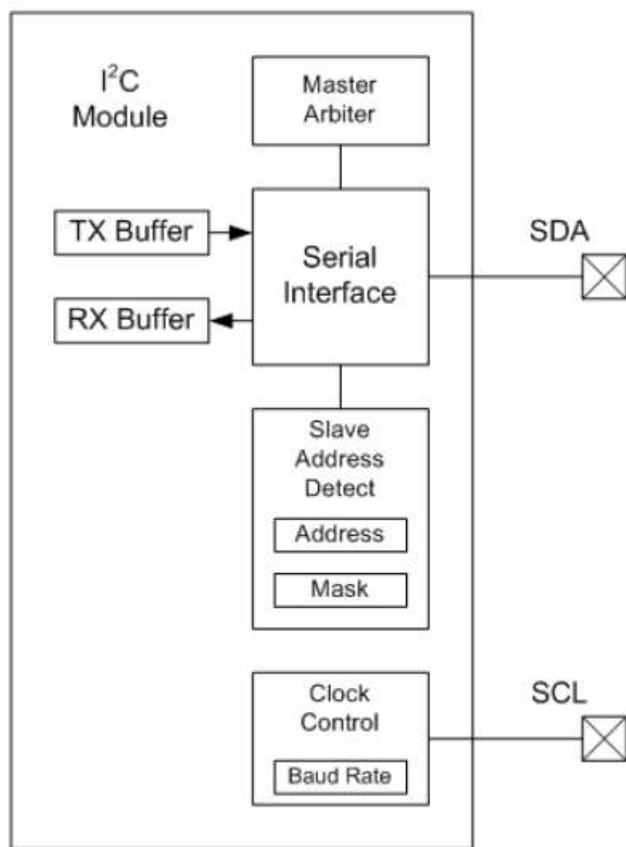


### 8.1.2. LES FONCTIONS DE LA PLIB\_I2C

La PLIB\_I2C fournit des fonctions de bas niveau qu'il faut combiner pour obtenir une action de base comme l'écriture ou la lecture d'un octet.

Cela nous permettra de découvrir les fonctions utilisées par le driver I2C fourni.

Dans l'aide de Harmony, section I2C Peripheral Library, on trouve le schéma de principe du module I2C.



#### 8.1.2.1. VUE D'ENSEMBLE DES FONCTIONS DE LA PLIB\_I2C

Voici une partie des fonctions de la plib\_i2c. Les fonctions d'existence et de gestion du slave ne sont pas présentées.

Baud Rate Generator Control Functions

	Name	Description
≡	<a href="#">PLIB_I2C_BaudRateGet</a>	Calculates the I2C module's current SCL clock frequency.
≡	<a href="#">PLIB_I2C_BaudRateSet</a>	Sets the desired baud rate.

**General Initialization Functions**

Name	Description
<a href="#">PLIB_I2C_Disable</a>	Disables the specified I2C module.
<a href="#">PLIB_I2C_Enable</a>	Enables the specified I2C module.
<a href="#">PLIB_I2C_GeneralCallDisable</a>	Disables the I2C module from recognizing the general call address.
<a href="#">PLIB_I2C_GeneralCallEnable</a>	Enables the I2C module to recognize the general call address.
<a href="#">PLIB_I2C_HighFrequencyDisable</a>	Disables the I2C module from using high frequency (400 kHz or 1 MHz) signaling.
<a href="#">PLIB_I2C_HighFrequencyEnable</a>	Enables the I2C module to use high frequency (400 kHz or 1 MHz) signaling.
<a href="#">PLIB_I2C_IPMIDisable</a>	Disables the I2C module's support for the IPMI specification
<a href="#">PLIB_I2C_IPMIEnable</a>	Enables the I2C module to support the Intelligent Platform Management Interface (IPMI) specification (see Remarks).
<a href="#">PLIB_I2C_ReservedAddressProtectDisable</a>	Disables the I2C module from protecting reserved addresses, allowing it to respond to them.
<a href="#">PLIB_I2C_ReservedAddressProtectEnable</a>	Enables the I2C module to protect (not respond to) reserved addresses.
<a href="#">PLIB_I2C_SlaveClockStretchingDisable</a>	Disables the I2C module from stretching the slave clock.
<a href="#">PLIB_I2C_SlaveClockStretchingEnable</a>	Enables the I2C module to stretch the slave clock.
<a href="#">PLIB_I2C_SMBDisable</a>	Disable the I2C module support for SMBus electrical signaling levels.
<a href="#">PLIB_I2C_SMBEnable</a>	Enables the I2C module to support System Management Bus (SMBus) electrical signaling levels.
<a href="#">PLIB_I2C_StopInIdleDisable</a>	Disables the Stop-in-Idle feature.
<a href="#">PLIB_I2C_StopInIdleEnable</a>	Enables the I2C module to stop when the processor enters Idle mode

**General Status Functions**

Name	Description
<a href="#">PLIB_I2C_ArbitrationLossClear</a>	Clears the arbitration loss status flag
<a href="#">PLIB_I2C_ArbitrationLossHasOccurred</a>	Identifies if bus arbitration has been lost.
<a href="#">PLIB_I2C_BusIsIdle</a>	Determines whether the I2C bus is idle or busy.
<a href="#">PLIB_I2C_StartClear</a>	Clears the start status flag
<a href="#">PLIB_I2C_StartWasDetected</a>	Identifies when a Start condition has been detected.
<a href="#">PLIB_I2C_StopClear</a>	Clears the stop status flag
<a href="#">PLIB_I2C_StopWasDetected</a>	Identifies when a Stop condition has been detected

**Master Control Functions**

Name	Description
<a href="#">PLIB_I2C_MasterReceiverClock1Byte</a>	Drives the bus clock to receive 1 byte of data from a slave device.
<a href="#">PLIB_I2C_MasterStart</a>	Sends an I2C Start condition on the I2C bus in Master mode.
<a href="#">PLIB_I2C_MasterStartRepeat</a>	Sends a repeated Start condition during an ongoing transfer in Master mode.
<a href="#">PLIB_I2C_MasterStop</a>	Sends an I2C Stop condition to terminate a transfer in Master mode.

**Receiver Control Functions**

Name	Description
<a href="#">PLIB_I2C_ReceivedByteAcknowledge</a>	Allows a receiver to acknowledge a that a byte of data has been received.
<a href="#">PLIB_I2C_ReceivedByteGet</a>	Gets a byte of data received from the I2C bus interface.
<a href="#">PLIB_I2C_ReceivedBytesAvailable</a>	Detects whether the receiver has data available.
<a href="#">PLIB_I2C_ReceiverByteAcknowledgeHasCompleted</a>	Determines if the previous acknowledge has completed.
<a href="#">PLIB_I2C_ReceiverOverflowClear</a>	Clears the receiver overflow status flag.
<a href="#">PLIB_I2C_ReceiverOverflowHasOccurred</a>	Identifies if a receiver overflow error has occurred.
<a href="#">PLIB_I2C_MasterReceiverReadyToAcknowledge</a>	Checks whether the hardware is ready to acknowledge.

**Transmitter Control Functions**

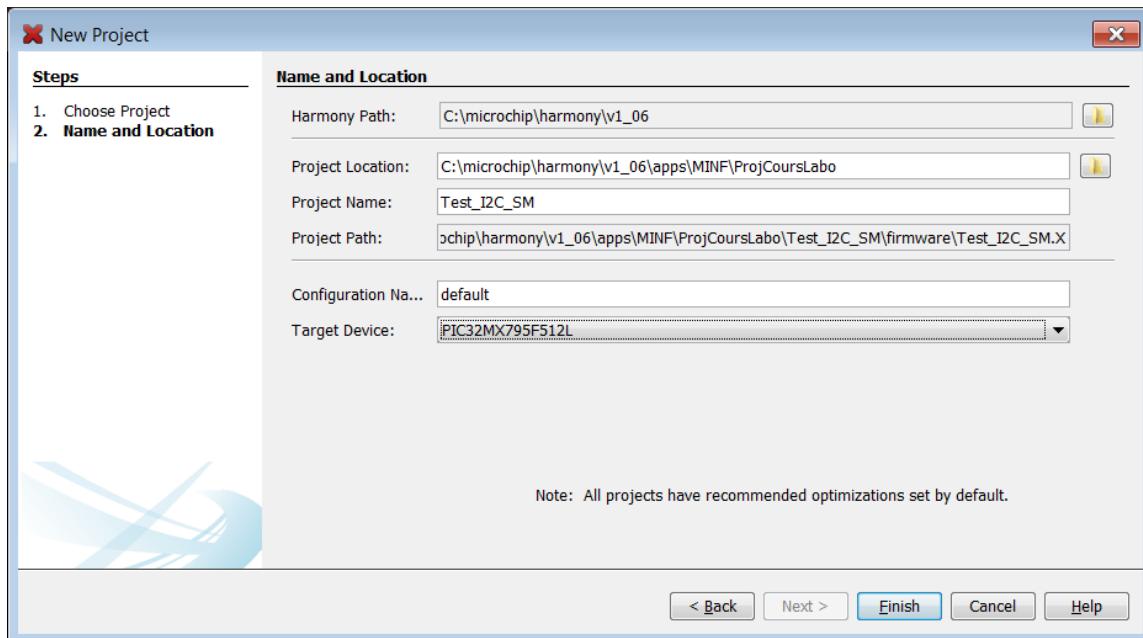
Name	Description
<a href="#">PLIB_I2C_TransmitterByteHasCompleted</a>	Detects whether the module has finished transmitting the most recent byte.
<a href="#">PLIB_I2C_TransmitterByteSend</a>	Sends a byte of data on the I2C bus.
<a href="#">PLIB_I2C_TransmitterByteWasAcknowledged</a>	Determines whether the most recently sent byte was acknowledged.
<a href="#">PLIB_I2C_TransmitterIsBusy</a>	Identifies if the transmitter of the specified I2C module is currently busy (unable to accept more data).
<a href="#">PLIB_I2C_TransmitterIsReady</a>	Detects if the transmitter is ready to accept data to transmit.
<a href="#">PLIB_I2C_TransmitterOverflowClear</a>	Clears the transmitter overflow status flag.
<a href="#">PLIB_I2C_TransmitterOverflowHasOccurred</a>	Identifies if a transmitter overflow error has occurred.

## 8.2. PROJET AVEC DRIVER I2C

L'exemple suivant a été réalisé avec les logiciels suivants :

- Harmony v1.08
- MPLABX IDE v3.40
- XC32 v1.42

### 8.2.1. CRÉATION DU PROJET



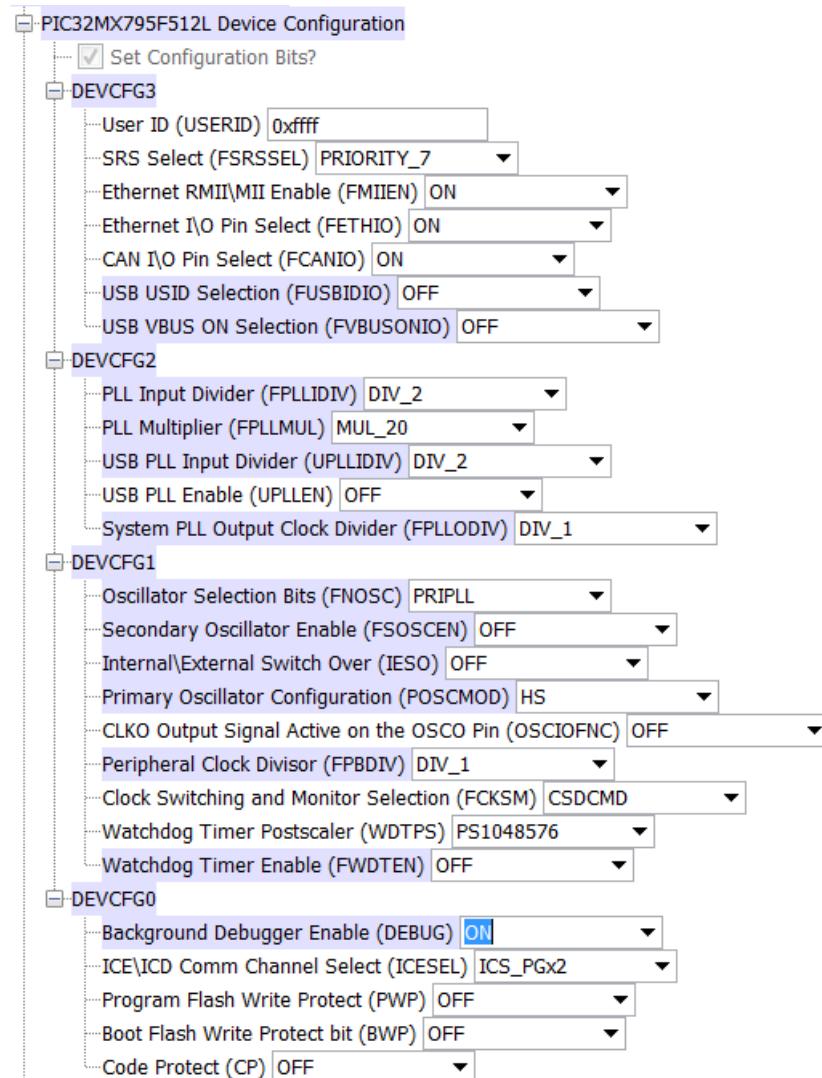
### 8.2.2. CHOIX DES ÉLÉMENTS

#### 8.2.2.1. CHOIX BSP



### 8.2.2.2. DEVICE CONFIGURATION

On retrouve les 4 sections déjà connues :

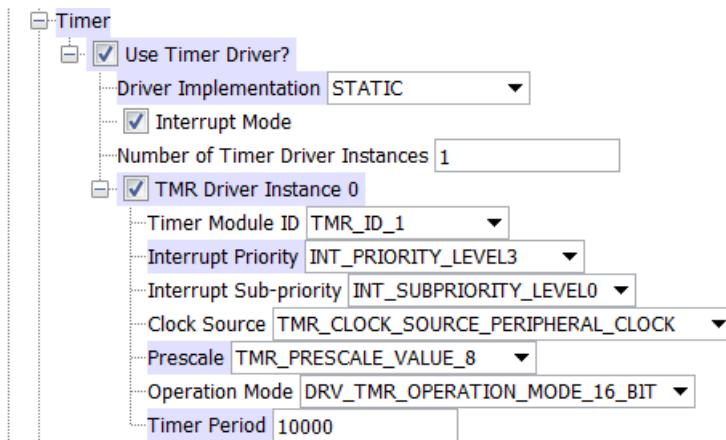


The screenshot shows the Device Configuration interface for a PIC32MX795F512L microcontroller. It displays four main configuration sections:

- DEVCFG3:** Includes fields for User ID (USERID), SRS Select (FSRSEL), Ethernet RMII\MI Enable (FMIEN), Ethernet I/O Pin Select (FETHIO), CAN I/O Pin Select (FCANIO), USB USID Selection (FUSBIDIO), and USB VBUS ON Selection (FVBUSONIO).
- DEVCFG2:** Includes fields for PLL Input Divider (FPLLIDIV), PLL Multiplier (FPLLMUL), USB PLL Input Divider (UPLLIDIV), USB PLL Enable (UPLLEN), and System PLL Output Clock Divider (FPLLODIV).
- DEVCFG1:** Includes fields for Oscillator Selection Bits (FNOSC), Secondary Oscillator Enable (FSOSCEN), Internal\External Switch Over (IESO), Primary Oscillator Configuration (POSCMOD), CLKO Output Signal Active on the OSCO Pin (OSCIOFNC), Peripheral Clock Divisor (FPBDIV), Clock Switching and Monitor Selection (FCKSM), Watchdog Timer Postscaler (WDTPS), and Watchdog Timer Enable (FWDTEN).
- DEVCFG0:** Includes fields for Background Debugger Enable (DEBUG), ICE\ICD Comm Channel Select (ICESEL), Program Flash Write Protect (PWP), Boot Flash Write Protect bit (BWP), and Code Protect (CP).

### 8.2.2.3. TIMER DRIVER

Configuration pour une période de 1ms et interruption priorité 3.

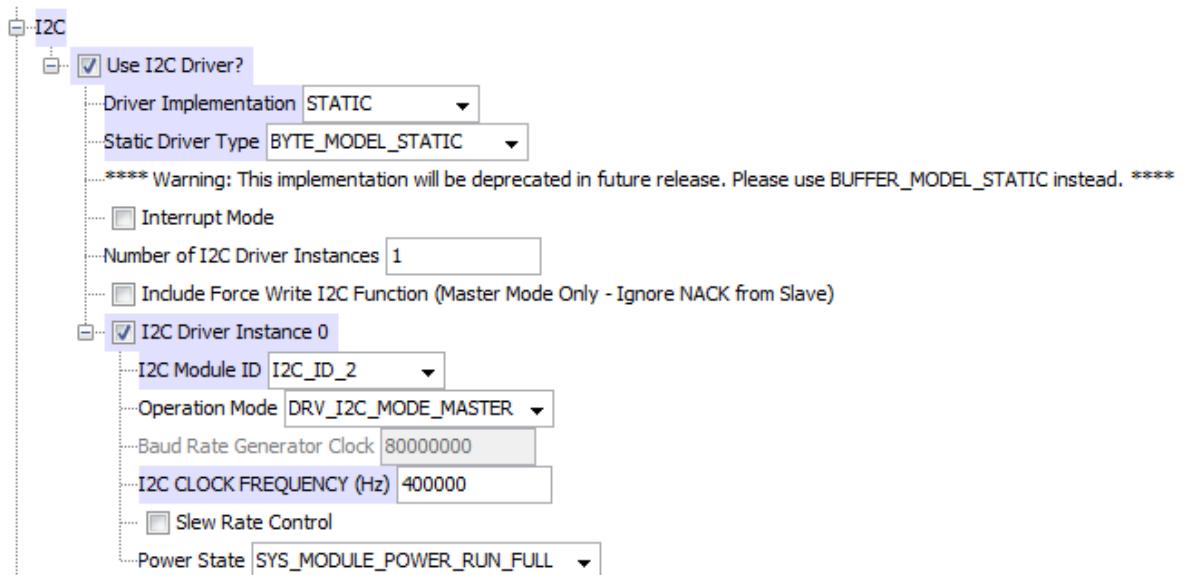


The screenshot shows the Timer configuration interface. Key settings include:

- Driver Implementation:** STATIC
- Interrupt Mode:** Enabled
- Number of Timer Driver Instances:** 1
- TMR Driver Instance 0:**
  - Timer Module ID:** TMR\_ID\_1
  - Interrupt Priority:** INT\_PRIORITY\_LEVEL3
  - Interrupt Sub-priority:** INT\_SUBPRIORITY\_LEVEL0
  - Clock Source:** TMR\_CLOCK\_SOURCE\_PERIPHERAL\_CLOCK
  - Prescale:** TMR\_PRESCALE\_VALUE\_8
  - Operation Mode:** DRV\_TMR\_OPERATION\_MODE\_16\_BIT
  - Timer Period:** 10000 (representing 1ms)

### 8.2.2.4. I2C DRIVER

Configuration en master, utilisation du module 2. Clock à 400 kHz.

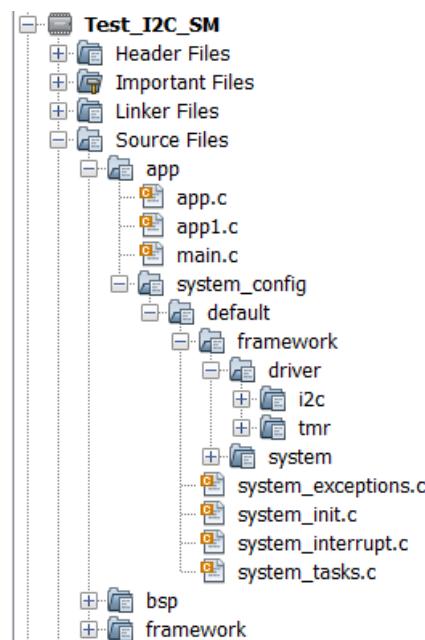


Ne pas cocher "slew rate control" afin d'éviter des problèmes d'incompatibilité avec le capteur de température LM92.

### 8.2.3. OBSERVATION DES RÉSULTATS

#### 8.2.3.1. ORGANISATION

Après la génération du code, on constate au niveau du system\_config, sous framework une section driver dans laquelle on trouve i2c et tmr.



### 8.3. RÉALISATION DE FONCTIONS I2C À MACHINE D'ÉTAT

Pour permettre une utilisation concurrente des fonctions I2C, par exemple pour gérer plusieurs composants et ceci au sein d'un cycle rapide, il est nécessaire de réaliser des fonctions comportant des machines d'état. Ce qui implique que la fonction est appelée cycliquement et que son traitement interne évolue.

Les fonctions à machine d'état sont des fonctions comportant un switch avec une variable d'état. Ces fonctions doivent être appelées cycliquement.

Une attente sera réalisée selon le principe suivant :

```
switch (MonEtat) {
    case 0 :
        // Attente Machin Ready
        if (IsMachinReady() ) {
            // Passe à l'étape suivante
            MonEtat = 1
        }
    break;
    case 1 :
    ...
}
```

Nous allons suivre par étape la réalisation d'un tel système pour obtenir la température du LM92.

#### 8.3.1. ACTION AU NIVEAU DE L'APPLICATION

Au niveau de l'application, il faut appeler la fonction d'initialisation dans le case APP\_STATE\_INIT. Dans le case APP\_STATE\_SERVICE\_TASKS, on utilise la fonction d'exploitation des résultats et on effectue l'affichage.

Il faut inclure dans **app.h** :

```
#include "Mc32gestI2cLM92_SM.h"
```

##### 8.3.1.1. AJOUT DONNÉE APPLICATION POUR LM92 DANS STRUCT APP\_DATA

L'ajout de champ dans la struct APP\_DATA permet de disposer des variables chaque fois que app.h est inclus.

```
typedef struct
{
    /* The application's current state */
    APP_STATES state;
    /* TODO: Define any additional data used by the
    application. */
    int16_t RawTemp;      // valeur brute registre température
    int32_t TempMilli;   // température en millième de degré
    float Lm92Temp;      // température en degré
} APP_DATA;
```

### 8.3.1.2. TRAITEMENT DANS CASE APP\_STATE\_INIT

Appel d'une fonction qui initialise le système à machine d'état ainsi que le bus I2C.  
**Lm92SmInit(true); // en fast**

### 8.3.1.3. TRAITEMENT DANS CASE APP\_STATE\_SERVICE\_TASKS

Dans la boucle du programme principal comportant un délai d'attente de 1000 ms, on teste si la température est disponible.

```
case APP_STATE_SERVICE_TASKS:
    // Activé toute les 1000 ms
    BSP_LEDToggle(BSP_LED_2);
    // Test si LM92 ready
    if (I2C_LM92_SM_IsReady(&DescrLm92)) {
        lcd_gotoxy(1,2);
        appData.RawTemp =
            I2C_LM92_SM_GetRawTemp(&DescrLm92);
        printf_lcd( "LM92 raw: %04X    ", appData.RawTemp );
        lcd_gotoxy(1,3);
        appData.TempMilli =
            I2C_LM92_SM_GetTempMilli(&DescrLm92);
        printf_lcd("LM92 Tmilli: %06d", appData.TempMilli);
        lcd_gotoxy(1,4);
        appData.Lm92Temp = I2C_LM92_SM_GetTemp(&DescrLm92);
        printf_lcd( "LM92 temp: %6.2f    ",
                    appData.Lm92Temp);
        // Quittance la lecture
        I2C_LM92_SM_Restart(&DescrLm92);
    }
    appData.state = APP_STATE_WAIT;
break;
```

Les informations sont placées dans le descripteur, ce qui permet de les exploiter lorsqu'on a le Ready. Le descripteur est fourni dans Mc32gestI2cLM92\_SM.h.

Pour relancer une conversion, on utilise la fonction **I2C\_LM92\_SM\_Restart** qui réarme le système.

### 8.3.2. ACTION AU NIVEAU DE L'INTERRUPTION DU TIMER1

Dans la réponse à l'interruption du Timer1, prévue pour un cycle de 1 ms, on effectue l'appel de la fonction d'exécution **I2C\_LM92\_SM\_Execute**. Le traitement évolue à l'intérieur de cette fonction. En plus on gère le déclenchement de l'application toutes les 1000 ms.

```
// Interruption timer1, Cycle 1 ms
void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
    _IntHandlerDrvTmrInstance0(void)
{
    static int count = 0;

    BSP_LEDOn(BSP_LED_0);

    // Appel cyclique traitement LM92
    I2C_LM92_SM_Execute(&DescrLm92);
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
    count++;
    if (count >= 1000 ) {
        count = 0;
        APP_UpdateState(APP_STATE_SERVICE_TASKS);
    }
    BSP_LEDOFF(BSP_LED_0);
}
```

### 8.3.3. INITIALISATION DANS LE FICHIER SYSTEM\_INIT.C

Dans la fonction **SYS\_Initialize** qui se trouve dans le fichier **system\_init.c**, on met en commentaire l'appel de la fonction **DRV\_I2C0\_Initialize()** afin de l'appeler plus tard dans la fonction **I2C\_SM\_init**.

Voici une partie de la fonction **SYS\_Initialize** :

```
/* Board Support Package Initialization */
BSP_Initialize();

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

// DRV_I2C0_Initialize(); // CHR utilisé plus tard
// dans I2C_SM_init

/* Initialize System Services */
SYS_INT_Initialize();

/* Initialize Middleware */
/* Enable Global Interrupts */
SYS_INT_Enable();

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

### 8.3.4. CONTENU DU FICHIER MC32GESTI2CLM92\_SM.H

Ce fichier contient les types énumérés définissant les valeurs des états et des séquences. Il contient aussi le typedef de la structure correspondant au descripteur, ainsi que le prototype des fonctions.

```
#include <stdint.h>
#include "Mc32_I2cUtil_SM.h"

// enumeration Etat principal
typedef enum { LM92_SM_Idle, LM92_SM_Busy, LM92_SM_ready }
E_LM92_state;

// enumeration Etapes de la séquence
typedef enum { LM92_I2CSEQ_Idle,
               LM92_I2CSEQ_Start,
               LM92_I2CSEQ_WriteAddrW,
               LM92_I2CSEQ_WritePtrTemp,
               LM92_I2CSEQ_ReStart,
               LM92_I2CSEQ_WriteAddrR,
               LM92_I2CSEQ_ReadMsb,
               LM92_I2CSEQ_ReadLsb,
               LM92_I2CSEQ_Stop,
} E_LM92_Sequence;
```

✍ Avec l'introduction du driver I2C, il n'est plus possible de modifier l'ID du module I2C d'où la mise en commentaire.

```
// Descripteur LM92 pour traitement par machine d'état
typedef struct {
    // I2C_MODULE i2cModuleId;           // Id du Module I2C
    E_LM92_state Lm92state;           // Etat principal
    E_LM92_Sequence Lm92Sequence;    // Etapes de la séquence
    S_Descr_I2C_SM I2cSmInfo;        // Descr. pour fonct. I2C_SM
    uint8 Lsb;
    uint8 Msb;
    sint16 RawTemp;                 // valeur brute registre température
    sint32 TempMilli;               // température en millième de degré
    float Temperature;              // température en degré
} S_Descr_LM92_SM;

// Descripteur gestion LM92 par State Machine
extern S_Descr_LM92_SM DescrLm92;      // descripteur LM92

// prototypes des fonctions
void Lm92SmInit(bool Fast); // pour appel externe

void I2C_LM92_SM_Init(S_Descr_LM92_SM *pDescr, bool Fast);
void I2C_LM92_SM_Execute(S_Descr_LM92_SM *pDescr);
void I2C_LM92_SM_Restart(S_Descr_LM92_SM *pDescr);
bool I2C_LM92_SM_IsReady(S_Descr_LM92_SM *pDescr);
```

```
float I2C_LM92_SM_GetTemp(S_Descr_LM92_SM *pDescr);
int32_t I2C_LM92_SM_GetTempMilli(S_Descr_LM92_SM *pDescr);
int16_t I2C_LM92_SM_GetRawTemp(S_Descr_LM92_SM *pDescr);
```

### 8.3.5. CONTENU DU FICHIER Mc32GESTI2cLM92\_SM.c

Ce fichier contient l'implémentation des fonctions.

```
#include "Mc32gestI2cLM92_SM.h"
#include "Mc32_I2cUtil_SM.h"
#include "system_config.h"      // pour bsp

// Compilation conditionnelle (Mettre en commentaire
// pour ne pas utiliser les leds)
#define USE_LED_MEASURE true

// Définition pour LM92
#define lm92_rd    0x91          // lm92 address for read
#define lm92_wr    0x90          // lm92 address for write
#define lm92_temp_ptr 0x00        // adr. pointeur température

// Définitions du bus (pour mesures)
// #define I2C-SCK SCL2/RA2      PORTAbits.RA2    pin 58
// #define I2C-SDA SDa2/RA3       PORTAbits.RA3    pin 59

// Descripteur gestion LM92 par State Machine
S_Descr_LM92_SM DescrLm92;      // descripteur LM92
```

#### 8.3.5.1. LA FONCTION LM92SMINIT

Cette fonction est une fonction interface qui permet un appel sans devoir fournir un descripteur.

```
void Lm92SmInit(bool Fast)
{
    I2C_LM92_SM_Init(&DescrLm92, true);
}
```

#### 8.3.5.2. LA FONCTION I2C\_LM92\_SM\_INIT

La fonction I2C\_LM92\_SM\_Init effectue l'initialisation du mécanisme SM du LM92 et de la communication I2C.

```
void I2C_LM92_SM_Init(S_Descr_LM92_SM *pDescr, bool Fast)
{
    pDescr->Lm92state = LM92_SM_Idle;
    pDescr->Lm92Sequence = LM92_I2CSEQ_Idle;
    pDescr->RawTemp = 0;
    pDescr->TempMilli = 0;
    pDescr->Temperature = 0.0;

    I2C_SM_init(Fast, &pDescr->I2cSmInfo );
}
```

### 8.3.5.3. LA FONCTION I2C\_LM92\_SM\_RESTART

Cette fonction sort de l'état Ready et passe en Idle, ce qui fait recommencer la séquence au début dans la fonction **I2C\_LM92\_SM\_Execute**.

```
void I2C_LM92_SM_Restart(S_Descr_LM92_SM *pDescr) {
    pDescr->Lm92state = LM92_SM_Idle;
    pDescr->Lm92Sequence = LM92_I2CSEQ_Idle;
}
```

### 8.3.5.4. LA FONCTION I2C\_LM92\_SM\_IsREADY

Cette fonction retourne true si la fonction d'exécution est dans l'état ready.

```
bool I2C_LM92_SM_IsReady(S_Descr_LM92_SM *pDescr) {
    bool answer = false;
    if (pDescr->Lm92state == LM92_SM_ready) {
        answer = true;
    }
    return answer;
}
```

### 8.3.5.5. LA FONCTION I2C\_LM92\_SM\_EXECUTE

Cette fonction doit être appelée cycliquement (interruption rapide ou boucle de traitement cyclique assez rapide).

Cette fonction effectue la lecture du registre de température du LM92 en la décomposant en étapes sans attente.

⌚ Utilisation pour mesure de BSP\_LED\_6 et BSP\_LED\_5.

```
void I2C_LM92_SM_Execute(S_Descr_LM92_SM *pDescr)
{
    //Déclaration des variables
    int16_t RawTemp;
    bool AckBit;

    switch ( pDescr->Lm92state ) {
        case LM92_SM_Idle :
            // Passe à Busy
            pDescr->Lm92state = LM92_SM_Busy;
            pDescr->Lm92Sequence = LM92_I2CSEQ_Start;
            #ifdef USE_LED_MEASURE
                BSP_LEDOn(BSP_LED_6); // Début activité
            #endif
            break;

        case LM92_SM_Busy :
            // Effectue la lecture par étapes
            switch (pDescr->Lm92Sequence) {
                case LM92_I2CSEQ_Start :
                    // i2c_start();
                    I2C_SM_start(&pDescr->I2cSmInfo);
                    if (I2C_SM_isReady
```

```

        (&pDescr->I2cSmInfo));
pDescr->Lm92Sequence =
LM92_I2CSEQ_WriteAddrW;
I2C_SM_begin(&pDescr->I2cSmInfo);
#ifdef USE_LED_MEASURE
    BSP_LEDToggle(BSP_LED_5);
#endif
}
break;

case LM92_I2CSEQ_WriteAddrW :
// i2c_write(lm92_wr); addr. + Write
I2C_SM_write(&pDescr->I2cSmInfo,
               lm92_wr, &AckBit);
if (I2C_SM_isReady
        (&pDescr->I2cSmInfo)) {
pDescr->Lm92Sequence =
LM92_I2CSEQ_WritePtrTemp;
I2C_SM_begin(&pDescr->I2cSmInfo);
#ifdef USE_LED_MEASURE
    BSP_LEDToggle(BSP_LED_5);
#endif
}
break;

case LM92_I2CSEQ_WritePtrTemp :
// i2c_write(lm92_temp_ptr);
I2C_SM_write(&pDescr->I2cSmInfo,
               lm92_temp_ptr, &AckBit);
if (I2C_SM_isReady
        (&pDescr->I2cSmInfo)) {
pDescr->Lm92Sequence =
LM92_I2CSEQ_ReStart;
I2C_SM_begin(&pDescr->I2cSmInfo);
#ifdef USE_LED_MEASURE
    BSP_LEDToggle(BSP_LED_5);
#endif
}
break;

case LM92_I2CSEQ_ReStart :
// i2c_reStart();
I2C_SM_reStart(&pDescr->I2cSmInfo);
if (I2C_SM_isReady
        (&pDescr->I2cSmInfo)) {
pDescr->Lm92Sequence =
LM92_I2CSEQ_WriteAddrR;
I2C_SM_begin(&pDescr->I2cSmInfo);
#ifdef USE_LED_MEASURE
    BSP_LEDToggle(BSP_LED_5);
#endif
}
}

```

```
        break;

    case LM92_I2CSEQ_WriteAddrR :
        // i2c_write(lm92_rd);      addr.+ Read
        I2C_SM_write(&pDescr->I2cSmInfo,
                      lm92_rd, &AckBit);
        if (I2C_SM_isReady
            (&pDescr->I2cSmInfo)) {
            pDescr->Lm92Sequence =
                LM92_I2CSEQ_ReadMsb;
            I2C_SM_begin(&pDescr->I2cSmInfo);
            #ifdef USE_LED_MEASURE
                BSP_LEDToggle(BSP_LED_5);
            #endif
        }
        break;

    case LM92_I2CSEQ_ReadMsb :
        // msb = i2c_read(1);      // ack
        I2C_SM_read(&pDescr->I2cSmInfo, true,
                     &pDescr->Msb);
        if (I2C_SM_isReady
            (&pDescr->I2cSmInfo)) {
            pDescr->Lm92Sequence =
                LM92_I2CSEQ_ReadLsb;
            I2C_SM_begin(&pDescr->I2cSmInfo);
            #ifdef USE_LED_MEASURE
                BSP_LEDToggle(BSP_LED_5);
            #endif
        }
        break;

    case LM92_I2CSEQ_ReadLsb :
        // lsb = i2c_read(0);      // no ack
        I2C_SM_read(&pDescr->I2cSmInfo, false,
                     &pDescr->Lsb);
        if (I2C_SM_isReady
            (&pDescr->I2cSmInfo)) {
            pDescr->Lm92Sequence =
                LM92_I2CSEQ_Stop;
            I2C_SM_begin(&pDescr->I2cSmInfo);
            #ifdef USE_LED_MEASURE
                BSP_LEDToggle(BSP_LED_5);
            #endif
        }
        break;
```

```

        case LM92_I2CSEQ_Stop :
            // i2c_stop();
            I2C_SM_stop(&pDescr->I2cSmInfo);
            if (I2C_SM_isReady
                (&pDescr->I2cSmInfo)) {
                // Effectue les calculs
                // des températures
                RawTemp = pDescr->Msb;
                RawTemp = RawTemp << 8;
                RawTemp = RawTemp | pDescr->Lsb;
                pDescr->RawTemp = RawTemp;
                RawTemp = RawTemp / 8;
                // bit poid faible = 0.0625 degré
                pDescr->TempMilli = RawTemp * 62.5;
                pDescr->Temperature =
                    RawTemp * 0.0625;
                pDescr->Lm92Sequence =
                    LM92_I2CSEQ_Idle;
                I2C_SM_begin(&pDescr->I2cSmInfo);
                pDescr->Lm92state = LM92_SM_ready;
                // marque fin séquence
                #ifdef USE_LED_MEASURE
                    BSP_LEDToggle(BSP_LED_6);
                #endif
            }
            break;

        case LM92_I2CSEQ_Idle :
            // ajout pour éviter Warning
            break;
    }

    case LM92_SM_ready :
        // Les résultats sont disponibles
        // Attente du Restart de l'utilisateur
        // après lecture des résultats
        break;
}
} // end I2C_LM92_SM_Execute

```

### 8.3.5.6. LA FONCTION I2C\_LM92\_SM\_GETTEMP

Retourne la valeur du champ Temperature.

```
float I2C_LM92_SM_GetTemp(S_Descr_LM92_SM *pDescr)
{
    return pDescr->Temperature;
}
```

### 8.3.5.7. LA FONCTION I2C\_LM92\_SM\_GETTEMPMILLI

Retourne la valeur du champ TempMilli (température au millième de degré).

```
int32_t I2C_LM92_SM_GetTempMilli(S_Descr_LM92_SM *pDescr) {
    return pDescr->TempMilli;
}
```

### 8.3.5.8. LA FONCTION I2C\_LM92\_SM\_GETRAWTTEMP

Retourne la valeur du champ RawTemp qui correspond au contenu du registre de température du LM92.

```
int16_t I2C_LM92_SM_GetRawTemp(S_Descr_LM92_SM *pDescr) {
    return pDescr->RawTemp;
}
```

### 8.3.6. CONTENU DU FICHIER Mc32\_I2CUTIL\_SM.H

Ce fichier contient les définitions nécessaires à la gestion de la machine d'état interne aux fonctions, ainsi que le prototype des fonctions.

Les fonctions reprennent le principe de décomposition d'une transaction I2C en actions de base (start, restart, read, write et stop), tout en utilisant les fonctions du driver I2C.

¶ Cela a pour effet que l'ID du module I2C n'est plus en paramètres pour les fonctions.

```
#include <stdbool.h>
#include <stdint.h>

// KIT 32MX795F512L Constants
#define KIT_I2C_BUS    I2C_ID_2      // n'est plus utilisée

typedef enum { I2C_XSM_Idle, I2C_XSM_Busy, I2C_XSM_Ready
} E_I2C_XSM;

typedef struct {
    E_I2C_XSM I2cMainSM;
    int I2cSeqSM;
    int DebugCode;
} S_Descr_I2C_SM;

bool I2C_SM_isReady( S_Descr_I2C_SM *pDSM );
void I2C_SM_begin( S_Descr_I2C_SM *pDSM );

void I2C_SM_init( bool Fast, S_Descr_I2C_SM *pDSM );
void I2C_SM_start(S_Descr_I2C_SM *pDSM);
void I2C_SM_reStart(S_Descr_I2C_SM *pDSM);
void I2C_SM_write(S_Descr_I2C_SM *pDSM, uint8 data,
                  bool *pAck );
void I2C_SM_read(S_Descr_I2C_SM *pDSM,
                  bool ackTodo, uint8 *pData );
void I2C_SM_stop(S_Descr_I2C_SM *pDSM);
```

### 8.3.7. CONTENU DU FICHIER Mc32\_I2cUtil\_SM.c

Ce fichier contient l'implémentation des fonctions I2C, mais avec une machine d'état interne permettant de remplacer les boucles d'attentes par des passages successifs à travers un test.

Ce set de fonctions doit permettre la réalisation du traitement d'autre composant I2C en s'inspirant du principe adopté pour le LM92.

Les fonctions sont réalisées en utilisant les fonctions du driver I2C obtenues du MHC de Harmony 1\_06

```
#include "Mc32_I2cUtil_SM.h"
#include "system_config/default/framework/driver/i2c/
                    drv_i2c_static.h"

#define I2C_CLOCK_FAST 400000
#define I2C_CLOCK_SLOW 100000
```

#### 8.3.7.1. LA FONCTION I2C\_SM\_INIT

Cette fonction initialise la machine d'état interne, et surtout configure et active le bus I2C. Cette fonction est prévue pour un appel unique. Elle appelle la fonction d'initialisation du driver I2C (version modifiée).

```
void I2C_SM_init( bool Fast, S_Descr_I2C_SM *pDSM )
{
    I2C_SM_begin(pDSM);
    // Appel la version modifiée de la fonction du driver
    DRV_I2C0_Initialize();
}
```

#### 8.3.7.2. LA FONCTION I2C\_SM\_BEGIN

Cette fonction permet le réarmement de la machine d'état interne aux fonctions.

```
void I2C_SM_begin( S_Descr_I2C_SM *pDSM )
{
    pDSM->I2cMainSM = I2C_XSM_Idle;
    pDSM->I2cSeqSM = 0;
    pDSM->DebugCode = 0;
}
```

#### 8.3.7.3. LA FONCTION I2C\_SM\_ISREADY

Cette fonction permet de savoir si la fonction a achevé sa séquence interne, c'est-à-dire si le traitement est effectué.

```
bool I2C_SM_isReady( S_Descr_I2C_SM *pDSM )
{
    bool answer = false;
    if (pDSM->I2cMainSM == I2C_XSM_Ready) {
        answer = true;
    }
    return answer;
}
```

### 8.3.7.4. LA FONCTION I2C\_SM\_START

Effectue le start I2C en séquence. Doit être appelée cycliquement.

```
void I2C_SM_start(S_Descr_I2C_SM *pDSM)
{
    int DebugCode = 0;

    switch (pDSM->I2cMainSM) {

        case I2C_XSM_Idle:
            // démarre le traitement
            pDSM->I2cMainSM = I2C_XSM_Busy;
            pDSM->I2cSeqSM = 0;
            break;

        case I2C_XSM_Busy :
            switch ( pDSM->I2cSeqSM) {
                case 0 :
                    // Wait for the bus to be idle,
                    // then start the transfer
                    if ( DRV_I2C0_MasterBusIdle() ) {
                        // OK alors Start
                        pDSM->I2cSeqSM = 1;
                        if( DRV_I2C0_MasterStart() == false)
                        {
                            pDSM->DebugCode = 1;
                        }
                    }
                break;

                case 1 :
                    // Wait for the signal to complete
                    if (DRV_I2C0_WaitForStartComplete()) {
                        pDSM->I2cSeqSM = 0;
                        pDSM->I2cMainSM = I2C_XSM_Ready;
                    }
                break;
            } // end switch SeqSM
        break;

        case I2C_XSM_Ready :
            // Attente relancement
        break;
    } // end switch
} // end I2C_SM_start
```

### 8.3.7.5. LA FONCTION I2C\_SM\_RESTART

Effectue le restart I2C en séquence. Doit être appelée cycliquement.

```
void I2C_SM_reStart( S_Descr_I2C_SM *pDSM)
{
    switch (pDSM->I2cMainSM) {

        case I2C_XSM_Idle:
            // démarre le traitement
            pDSM->I2cMainSM = I2C_XSM_Busy;
            pDSM->I2cSeqSM = 0;
            break;

        case I2C_XSM_Busy :
            switch ( pDSM->I2cSeqSM) {
                case 0 :
                    // Restart the transfer
                    if(DRV_I2C0_MasterRestart() == false)
                    {
                        pDSM->DebugCode = 2;
                    }
                    pDSM->I2cSeqSM = 1;
                    break;

                case 1 :
                    // Wait for the signal to complete
                    if ( DRV_I2C0_WaitForStartComplete())
                    {
                        pDSM->I2cSeqSM = 0;
                        pDSM->I2cMainSM = I2C_XSM_Ready;
                    }
                    break;
            } // end switch SeqSM
            break;

        case I2C_XSM_Ready :
            // Attente relancement
            break;
    } // end switch
} // end I2C_SM_reStart
```

### 8.3.7.6. LA FONCTION I2C\_SM\_WRITE

Effectue la transmission d'un octet sur le bus I2C en séquence. Doit être appelée cycliquement.

Cette fonction reçoit en paramètre par valeur l'octet à transmettre et par référence un boolean pour indiquer si ack ou pas.

```
void I2C_SM_write(S_Descr_I2C_SM *pDSM,
                     uint8_t data, bool *pAck)
{
    switch (pDSM->I2cMainSM) {
        case I2C_XSM_Idle:
            // demarre le traitement
            pDSM->I2cMainSM = I2C_XSM_Busy;
            pDSM->I2cSeqSM = 0;
            break;

        case I2C_XSM_Busy :
            switch ( pDSM->I2cSeqSM) {
                case 0 :
                    // Wait for the bus to be idle
                    // (nécessaire après un reStart)
                    if (DRV_I2C0_MasterBusIdle()) {
                        pDSM->I2cSeqSM = 1;
                    }
                break;

                case 1 :
                    // Wait for the transmitter to be ready
                    // CHR ?! cette étape n'existe pas
                    // dans le driver,
                    // mais ne semble pas nécessaire
                    // Transmit the byte
                    if (DRV_I2C0_ByteWrite(data) == false)
                    {
                        pDSM->DebugCode = 3;
                    }
                    pDSM->I2cSeqSM = 2;
                break;

                case 2 :
                    // Wait for byte write completion
                    if
                        (DRV_I2C0_WaitForByteWriteToComplete()) {
                            // on peut obtenir Ack
                            *pAck =
                                DRV_I2C0_WriteByteAcknowledged();
                            pDSM->I2cSeqSM = 0;
                            pDSM->I2cMainSM = I2C_XSM_Ready;
                        }
                    break;
            } // end switch SeqSM
    }
}
```

```

        break;
    case I2C_XSM_Ready :
        // Attente relancement
        break;
    } // end switch

} // end I2C_SM_write

```

### 8.3.7.7. LA FONCTION I2C\_SM\_READ

Effectue la réception d'un octet sur le bus I2C en séquence. Doit être appelée cycliquement.

Cette fonction a en paramètre par valeur, un booléen pour indiquer s'il faut effectuer ou non l'acquittement. Le retour de la valeur est réalisé par référence.

```

void I2C_SM_read(S_Descr_I2C_SM *pDSM,
                    bool ackTodo, uint8_t *pData )
{
    switch (pDSM->I2cMainSM ) {

        case I2C_XSM_Idle:
            // démarre le traitement
            pDSM->I2cMainSM = I2C_XSM_Busy;
            pDSM->I2cSeqSM = 0;
            break;

        case I2C_XSM_Busy :
            switch ( pDSM->I2cSeqSM) {
                case 0 :
                    // Check for receive overflow
                    // If OK Initiate clock to receive
                    if(DRV_I2C0_SetUpByteRead() == false)
                    {
                        pDSM->DebugCode = 4; // debug
                    }
                    pDSM->I2cSeqSM = 1;
                    break;

                case 1 :
                    // Wait until data available
                    if (DRV_I2C0_WaitForReadByteAvailable())
                    {
                        *pData = DRV_I2C0_ByteRead();
                        pDSM->I2cSeqSM = 2;
                    }
                    break;

                case 2 :
                    // Wait until ready to ack
                    if (ackTodo) {
                        DRV_I2C0_MasterACKSend();
                    } else {
                        DRV_I2C0_MasterNACKSend();
                    }
            }
    }
}

```

```

        // ??? attente dans boucle
        DRV_I2C0_WaitForACKOrNACKComplete();
        pDSM->I2cSeqSM = 0;
        pDSM->I2cMainSM = I2C_XSM_Ready;
        break;
    } // end switch SeqSM
break;

case I2C_XSM_Ready :
    // Attente relancement
    break;
} // end switch
} // end I2C_SM_read

```

### 8.3.7.8. LA FONCTION I2C\_SM\_STOP

Effectue la transaction stop sur le bus I2C en séquence, pour terminer la transaction I2C master. Doit être appelée cycliquement.

```

void I2C_SM_stop(S_Descr_I2C_SM *pDSM)
{
    switch (pDSM->I2cMainSM ) {

        case I2C_XSM_Idle:
            // demarre le traitement
            pDSM->I2cMainSM = I2C_XSM_Busy;
            pDSM->I2cSeqSM = 0;
        break;

        case I2C_XSM_Busy :
            switch ( pDSM->I2cSeqSM) {
                case 0 :
                    // Disable receiver and stop I2C
                    if ( DRV_I2C0_MasterStop() ) {
                        pDSM->I2cSeqSM = 1;
                    }
                    // Si pas OK on répète le stop
                break;

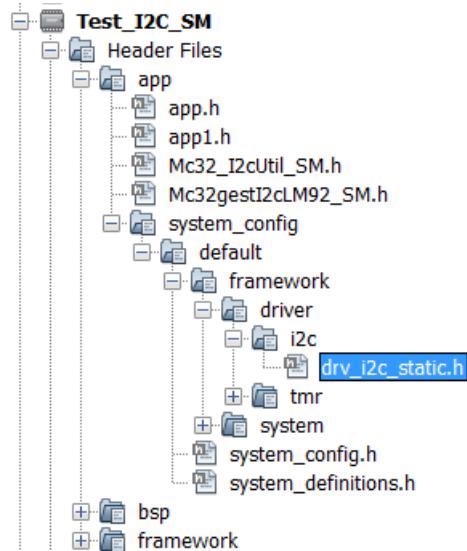
                case 1 :
                    // Wait for the signal to complete
                    // boucle d'attente dans la fonction
                    DRV_I2C0_WaitForStopComplete();
                    pDSM->I2cSeqSM = 0;
                    pDSM->I2cMainSM = I2C_XSM_Ready;
                    break;
            } // end switch SeqSM
        break;

        case I2C_XSM_Ready :
            // Attente relancement
            break;
    } // end switch
} // end I2C_SM_stop

```

### 8.3.8. CONTENU DU FICHIER DRV\_I2C\_STATIC.H

Voici le contenu du fichier drv\_i2c\_static.h (sans l'entête Microchip) que l'on trouve sous :



```

#ifndef _DRV_I2C_STATIC_H
#define _DRV_I2C_STATIC_H

#include <stdbool.h>
#include "system_config.h"
#include "peripheral/i2c/plib_i2c.h"
#include "system/clk/sys_clk.h"
#include "peripheral/ports/plib_ports.h"

// *****
// Section: Interface Headers for Instance 0 for the
// static driver
// *****
void DRV_I2C0_Initialize(void);
void DRV_I2C0_DeInitialize(void);

// *****
// Section: Instance 0 Byte transfer functions
// (Master/Slave)
// *****

bool DRV_I2C0_SetUpByteRead(void);
bool DRV_I2C0_WaitForReadByteAvailable(void);
uint8_t DRV_I2C0_ByteRead(void);
bool DRV_I2C0_ByteWrite(const uint8_t byte);
bool DRV_I2C0_WaitForByteWriteToComplete(void);
bool DRV_I2C0_WriteByteAcknowledged(void);
bool DRV_I2C0_ReceiverBufferIsEmpty(void);

```

```

// ****
// Section: Instance 0 I2C Master functions
// ****

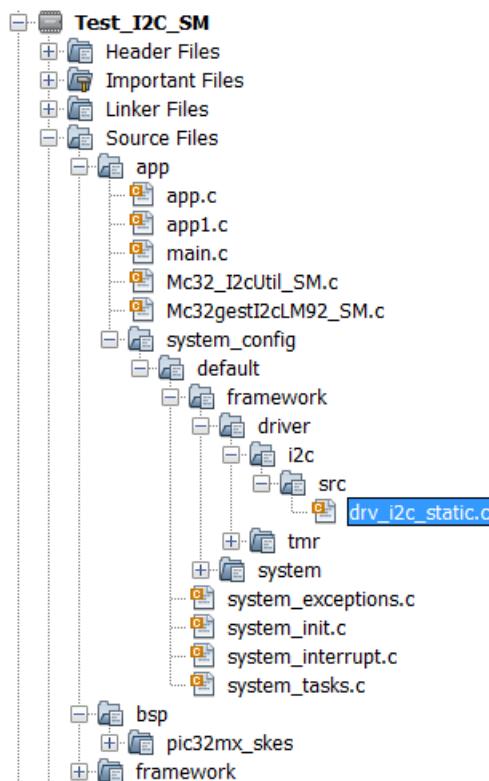
void DRV_I2C0_BaudRateSet(I2C_BAUD_RATE baudRate);
bool DRV_I2C0_MasterBusIdle(void);
bool DRV_I2C0_MasterStart(void);
bool DRV_I2C0_MasterRestart(void);
bool DRV_I2C0_WaitForStartComplete(void);
bool DRV_I2C0_MasterStop(void);
bool DRV_I2C0_WaitForStopComplete(void);
void DRV_I2C0_MasterACKSend(void);
void DRV_I2C0_MasterNACKSend(void);
bool DRV_I2C0_WaitForACKOrNACKComplete(void);

#endif // #ifndef _DRV_I2C_STATIC_H

```

### 8.3.9. CONTENU DU FICHIER DRV\_I2C\_STATIC.C

Voici le contenu du fichier drv\_i2c\_static.c (sans l'entête Microchip) que l'on trouve sous :



☺ De par le fait que le driver est généré dans la structure de notre projet, il est possible d'apporter des modifications au driver sans toucher à des sections système communes.

```
#include "framework/driver/i2c/driv_i2c_static.h"
```

### 8.3.9.1. LA FONCTION **DRV\_I2C0\_INITIALIZE** CRÉÉE

La fonction **DRV\_I2C0\_Initialize** ci-dessous correspond à la version générée par Harmony :

```
void DRV_I2C0_Initialize(void)
{
    /* Initialize I2C0 */
    PLIB_I2C_BaudRateSet(I2C_ID_2,
        SYS_CLK_PeripheralFrequencyGet(CLK_BUS_PERIPHERAL_1),
        400000);
    PLIB_I2C_StopInIdleDisable(I2C_ID_2);

    /* Low frequency is enabled (**NOTE** PLIB function
    logic inverted) */
    PLIB_I2C_HighFrequencyEnable(I2C_ID_2);

    /* Enable I2C0 */
    PLIB_I2C_Enable(I2C_ID_2);
}
```

### 8.3.9.2. LA FONCTION **DRV\_I2C0\_INITIALIZE** MODIFIÉE

La fonction **DRV\_I2C0\_Initialize** modifiée ci-dessous reprend le principe déjà utilisé dans la fonction d'initialisation de I2C en version classique. Ce code est plus robuste et sera donc utilisé.

```
void DRV_I2C0_Initialize(void)
{
    /* Initialize I2C0 */
    PLIB_I2C_Disable(I2C_ID_2); // Ajout CHR
    PLIB_I2C_HighFrequencyEnable(I2C_ID_2);
    // Toujours en Fast

    PLIB_I2C_BaudRateSet(I2C_ID_2,
        SYS_CLK_PeripheralFrequencyGet
            (CLK_BUS_PERIPHERAL_1), 400000);

    /* Enable I2C0 */
    PLIB_I2C_Enable(I2C_ID_2);
}
```

### 8.3.9.3. LA FONCTION **DRV\_I2C0\_DEINITIALIZE**

Voici la fonction **DRV\_I2C0\_DeInitialize** :

```
void DRV_I2C0_DeInitialize(void)
{
    /* Disable I2C0 */
    PLIB_I2C_Disable(I2C_ID_2);
}
```

### 8.3.9.4. LA FONCTION **DRV\_I2C0\_SetUpByteRead**

Voici la fonction **DRV\_I2C0\_SetUpByteRead**. Cette fonction teste la situation d'OverFlow et génère les coups d'horloge pour la lecture.

```
bool DRV_I2C0_SetUpByteRead(void)
{
    /* Check for receive overflow */
    if ( PLIB_I2C_ReceiverOverflowHasOccurred(I2C_ID_2) )
    {
        PLIB_I2C_ReceiverOverflowClear(I2C_ID_2);
        return false;
    }

    /* Initiate clock to receive */
    PLIB_I2C_MasterReceiverClock1Byte(I2C_ID_2);
    return true;
}
```

### 8.3.9.5. LA FONCTION **DRV\_I2C0\_WaitForReadByteAvailable**

Voici la fonction **DRV\_I2C0\_WaitForReadByteAvailable**. Cette fonction teste la disponibilité d'un octet.

```
bool DRV_I2C0_WaitForReadByteAvailable(void)
{
    /* Wait for Receive Buffer Full */
    if(PLIB_I2C_ReceivedByteIsAvailable(I2C_ID_2))
        return true;

    return false;
}
```

### 8.3.9.6. LA FONCTION **DRV\_I2C0\_ByteRead**

Voici la fonction **DRV\_I2C0\_ByteRead**. Cette fonction effectue la lecture d'un octet dans le tampon de réception.

```
uint8_t DRV_I2C0_ByteRead(void)
{
    /* Return received value */
    return (PLIB_I2C_ReceivedByteGet(I2C_ID_2));
}
```

### 8.3.9.7. LA FONCTION DRV\_I2C0\_BYTEWRITE

Voici la fonction DRV\_I2C0\_ByteWrite. Cette fonction effectue des tests et envoie un octet si tout est OK.

```
bool DRV_I2C0_ByteWrite(const uint8_t byte)
{
    /* if no IWCOL errors exist, then transmit byte */
    if ( !(PLIB_I2C_TransmitterIsBusy(I2C_ID_2)) &&
        (PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_2)) )
    {
        PLIB_I2C_TransmitterByteSend(I2C_ID_2, byte);
    }

    /* check if writing to I2CxTRN caused a transmitter
       overflow */
    if (PLIB_I2C_TransmitterOverflowHasOccurred(I2C_ID_2))
        return false;

    return true;
}
```

### 8.3.9.8. LA FONCTION DRV\_I2C0\_WAITFORBYTEWRITETOCOMPLETE

Voici la fonction DRV\_I2C0\_WaitForByteWriteToComplete. Elle indique si une écriture est terminée.

```
bool DRV_I2C0_WaitForByteWriteToComplete(void)
{
    /* if TBF == 0 and TRSTAT == 0 then write complete */

    if ( !(PLIB_I2C_TransmitterIsBusy(I2C_ID_2)) &&
        (PLIB_I2C_TransmitterByteHasCompleted(I2C_ID_2)) )
        return true;

    return false;
}
```

### 8.3.9.9. LA FONCTION DRV\_I2C0\_WRITEBYTEACKNOWLEDGED

Voici la fonction DRV\_I2C0\_WriteByteAcknowledged. Cette fonction teste si l'ack a été effectué.

```
bool DRV_I2C0_WriteByteAcknowledged(void)
{
    /* Check to see if transmit ACKed = true or NACKed =
       false */
    if (PLIB_I2C_TransmitterByteWasAcknowledged(I2C_ID_2))
        return true;

    return false;
}
```

### 8.3.9.10. LA FONCTION DRV\_I2C0\_BAUDRATESET

Voici la fonction DRV\_I2C0\_BaudRateSet :

```
void DRV_I2C0_BaudRateSet(I2C_BAUD_RATE baudRate)
{
    /* Disable I2C0 */
    PLIB_I2C_Disable(I2C_ID_2);

    /* Change baud rate */
    PLIB_I2C_BaudRateSet(I2C_ID_2,
        SYS_CLK_PeripheralFrequencyGet(CLK_BUS_PERIPHERAL_1),
        baudRate);

    /* Low frequency is enabled (**NOTE** PLIB function
    inverted) */
    PLIB_I2C_HighFrequencyEnable(I2C_ID_2);

    /* Enable I2C0 */
    PLIB_I2C_Enable(I2C_ID_2);
}
```

### 8.3.9.11. LA FONCTION DRV\_I2C0\_MASTERBUSIDLE

Voici la fonction DRV\_I2C0\_MasterBusIdle qui permet de tester si le bus I2C est au repos :

```
bool DRV_I2C0_MasterBusIdle(void)
{
    if (PLIB_I2C_BusIsIdle(I2C_ID_2))
        return true;
    else
        return false;
}
```

### 8.3.9.12. LA FONCTION DRV\_I2C0\_MASTERSTART

Voici la fonction DRV\_I2C0\_MasterStart qui effectue une série de tests avant d'exécuter le Start :

```
bool DRV_I2C0_MasterStart(void)
{
    /* if bus is not idle return with false */
    if (!(PLIB_I2C_BusIsIdle(I2C_ID_2)))
        return false;

    /* return false is Bus Collision exists */
    if (PLIB_I2C_ArbitrationLossHasOccurred(I2C_ID_2))
    {
        return false;
    }

    /* Issue start */
    PLIB_I2C_MasterStart(I2C_ID_2);

    return true;
}
```

### 8.3.9.13. LA FONCTION DRV\_I2C0\_WAITFORSTARTCOMPLETE

Voici la fonction DRV\_I2C0\_WaitForStartComplete qui indique si le start est terminé :

```
bool DRV_I2C0_WaitForStartComplete(void)
{
    /* Wait for start/restart sequence to finish (hardware
       clear) */

    if ( (PLIB_I2C_BusIsIdle(I2C_ID_2)) &&
        (PLIB_I2C_StartWasDetected(I2C_ID_2)) )
        return true;

    return false;
}
```

### 8.3.9.14. LA FONCTION DRV\_I2C0\_MASTERRESTART

Voici la fonction DRV\_I2C0\_MasterRestart qui effectue une série de tests avant d'exécuter le MasterStartRepeat.

```
bool DRV_I2C0_MasterRestart (void)
{
    /* if bus is not idle return with false */
    if (!(PLIB_I2C_BusIsIdle(I2C_ID_2)))
        return false;

    /* return false is Bus Collision exists */
    if (PLIB_I2C_ArbitrationLossHasOccurred(I2C_ID_2))
    {
        return false;
    }
```

```

    }

    /* Issue restart */
    PLIB_I2C_MasterStartRepeat(I2C_ID_2);

    return true;
}

```

### 8.3.9.15. LA FONCTION DRV\_I2C0\_MASTERSTOP

Voici la fonction DRV\_I2C0\_MasterStop, qui effectue une série de tests avant d'exécuter le MasterStop :

```

bool DRV_I2C0_MasterStop(void)
{
    /* if bus is not idle return with false */
    if (!(PLIB_I2C_BusIsIdle(I2C_ID_2)))
        return false;

    /* Issue stop */
    PLIB_I2C_MasterStop(I2C_ID_2);

    return true;
}

```

### 8.3.9.16. LA FONCTION DRV\_I2C0\_WAITFORSTOPCOMPLETE

Voici la fonction DRV\_I2C0\_WaitForStopComplete qui indique si le stop est terminé :

```

bool DRV_I2C0_WaitForStopComplete(void)
{
    if ( (PLIB_I2C_BusIsIdle(I2C_ID_2)) &&
        (PLIB_I2C_StopWasDetected(I2C_ID_2)) )
        return true;

    return false;
}

```

### 8.3.9.17. LA FONCTION DRV\_I2C0\_MASTERACKSEND

Voici la fonction DRV\_I2C0\_MasterACKSend qui effectue l'acknowledge si le master est prêt.

```

void DRV_I2C0_MasterACKSend(void)
{
    /* Check if receive is ready to ack */
    if ( PLIB_I2C_MasterReceiverReadyToAcknowledge
                    (I2C_ID_2) )
    {
        PLIB_I2C_ReceivedByteAcknowledge (I2C_ID_2, true);
    }
}

```

### 8.3.9.18. LA FONCTION DRV\_I2C0\_MASTERNACKSEND

Voici la fonction DRV\_I2C0\_MasterNACKSend qui effectue le NACK si le master est prêt.

```
void DRV_I2C0_MasterNACKSend(void)
{
    /* Check if receive is ready to nack */
    if ( PLIB_I2C_MasterReceiverReadyToAcknowledge
        (I2C_ID_2) )
    {
        PLIB_I2C_ReceivedByteAcknowledge (I2C_ID_2, false);
    }
}
```

### 8.3.9.19. LA FONCTION DRV\_I2C0\_WAITFORACKORNACKCOMPLETE

Voici la fonction DRV\_I2C0\_WaitForACKOrNACKComplete qui indique si l'ACK, respectivement le NACK, est terminé :

```
bool DRV_I2C0_WaitForACKOrNACKComplete(void)
{
    /* Check for ACK/NACK to complete */

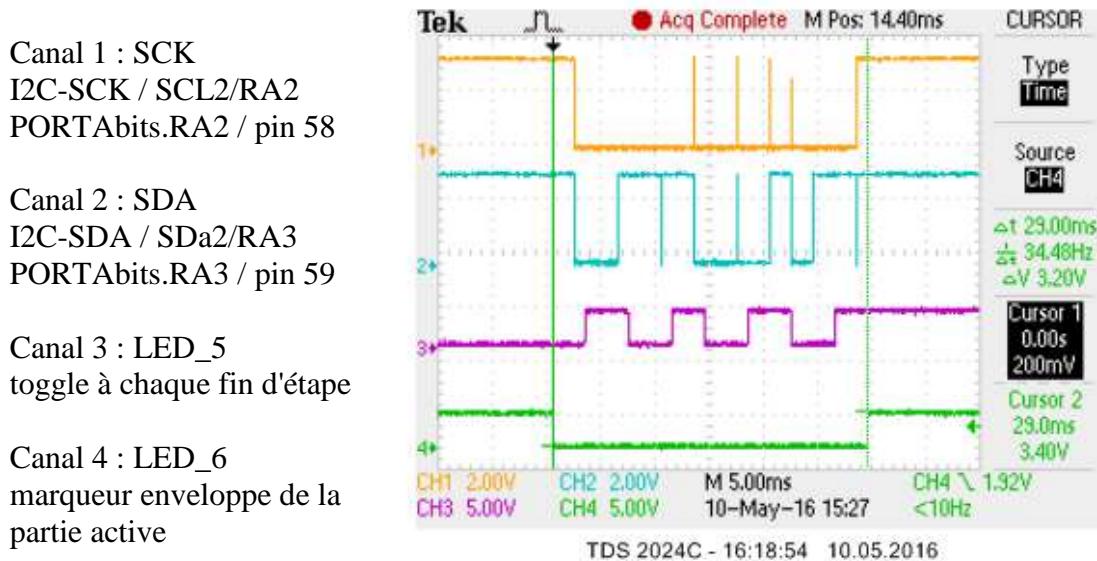
    if(PLIB_I2C_ReceiverByteAcknowledgeHasCompleted(I2C_ID_2))
        return true;

    return false;
}
```

### 8.3.10. CONTRÔLE DE FONCTIONNEMENT DES FONCTIONS SM

#### 8.3.10.1. VUE D'ENSEMBLE CYCLE 1 MS

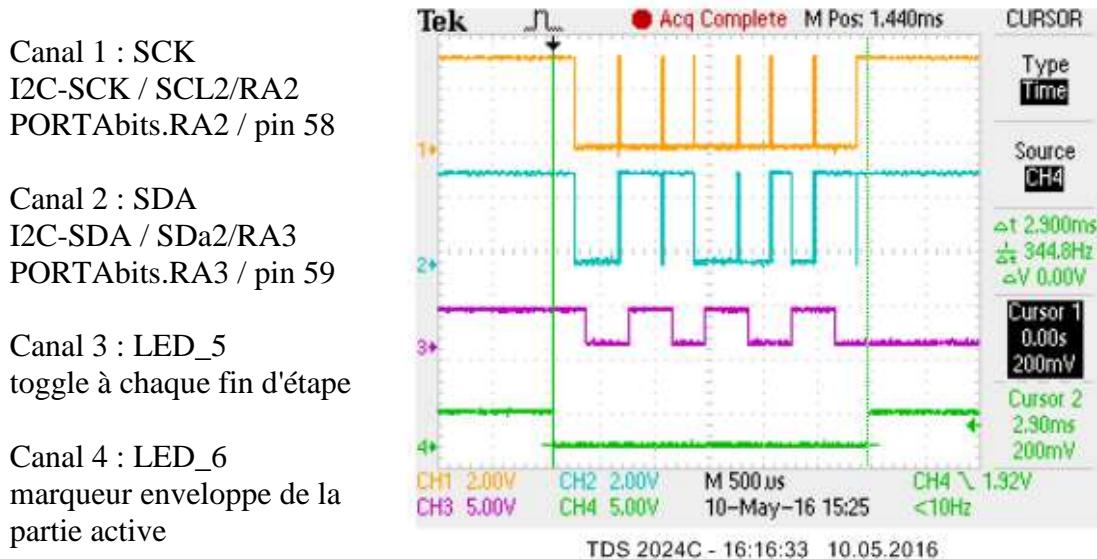
Voici une mesure de l'ensemble de la transaction avec un cycle de 1 ms :



On obtient un temps d'exécution de 29 ms, ce qui est important et rend difficile l'observation des signaux I2C.

#### 8.3.10.2. VUE D'ENSEMBLE CYCLE 100 US

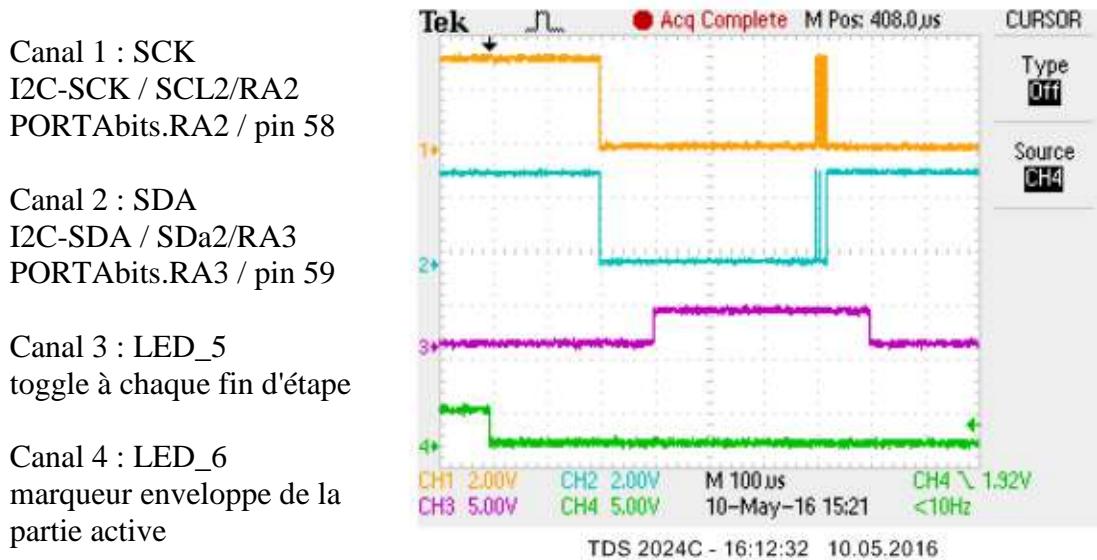
Voici une mesure de l'ensemble de la transaction avec un cycle de 100 us après adaptation du Timer et de la réponse à l'interruption.



On parvient à mieux observer les actions I2C avec un cycle de 100 us. La durée d'exécution est de 2.9 ms.

### 8.3.10.3. DÉTAIL DÉBUT TRANSACTION CYCLE 100 US

Voici une mesure de détail du début de la transaction :



Les transitions de la LED\_5 (canal 3) montrent qu'il y a quittance du i2c:start (flanc montant de LED\_5 et quittance du 1<sup>er</sup> i2c\_write, lors du flanc descendant.

### 8.3.11. CONCLUSION SUR LES FONCTIONS SM

Comme on peut le constater, l'introduction de machine d'état à deux niveaux rend un peu plus lourde l'écriture des fonctions mais permet d'utiliser la même systématique.

La découpe en fonction I2C de base comme start, stop, write et read permet d'avoir une correspondance avec les séquences décrites dans les datasheets des composants I2C. Cela évite aussi une séquence entièrement dédiée et comportant de 15 à 20 états.

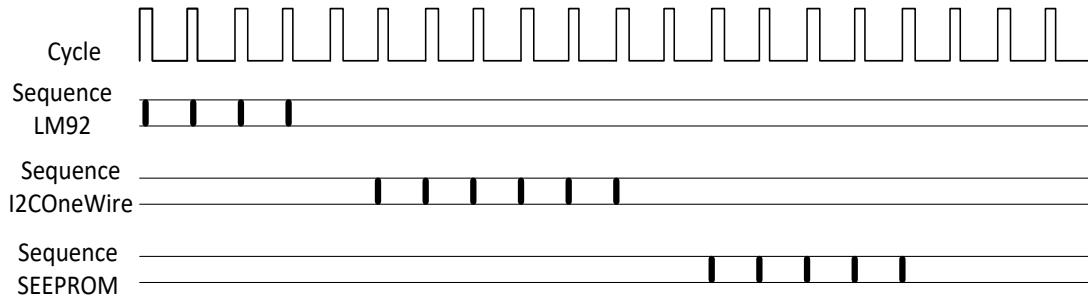
Avec l'utilisation du driver IC2 fourni par le MHC, on obtient l'adaptation au hardware lors des choix au niveau du configurateur.

## 8.4. UTILISATION EN MACHINE D'ÉTAT DE PLUSIEURS COMPOSANTS

Si on veut gérer sur le même bus I2C plusieurs composants, il est nécessaire d'effectuer les séquences l'une après l'autre.

### 8.4.1. ILLUSTRATION DU SÉQUENCIEMENT DANS LE TEMPS

Pour illustrer cela, supposons que nous disposons du LM92, du I2C-OneWire et de la EEPROM. Dans le temps, le traitement doit être réalisé de la manière suivante :



### 8.4.2. PRINCIPE RÉALISATION DU SÉQUENCIEMENT DANS LE TEMPS

Si on appelle l'une après l'autre les fonctions de réalisation de séquence, on obtiendra un mélange des actions élémentaires I2C, ce qui ne fonctionnera pas du tout.

Pour réaliser le séquencement dans le temps il faut à nouveau introduire un switch avec une variable qui représente le capteur à exécuter.

Par exemple :

```
typedef enum { IC_LM92, IC_DS2482, IC_EEPROM
} E_Composants_I2C;

// Variable globale
E_Composants_I2C ComposantEnCours = IC_LM92;
```

Dans le traitement cyclique :

```
Switch (ComposantEnCours) {
    case IC_LM92 :
        // Appel cyclique traitement LM92
        I2C_LM92_SM_Execute(&DescrLm92);
        if (I2C_LM92_SM_IsReady(&DescrLm92)) {
            ComposantEnCours = IC_DS2482;
        }
        break;

    case IC_DS2482:
        // Appel cyclique traitement DS2482
        I2C_DS2482_SM_Execute(&DescrDs2482);
        if (I2C_DS2482_SM_IsReady(&DescrDs2482)) {
            ComposantEnCours = IC_EEPROM;
        }
        break;
}
```

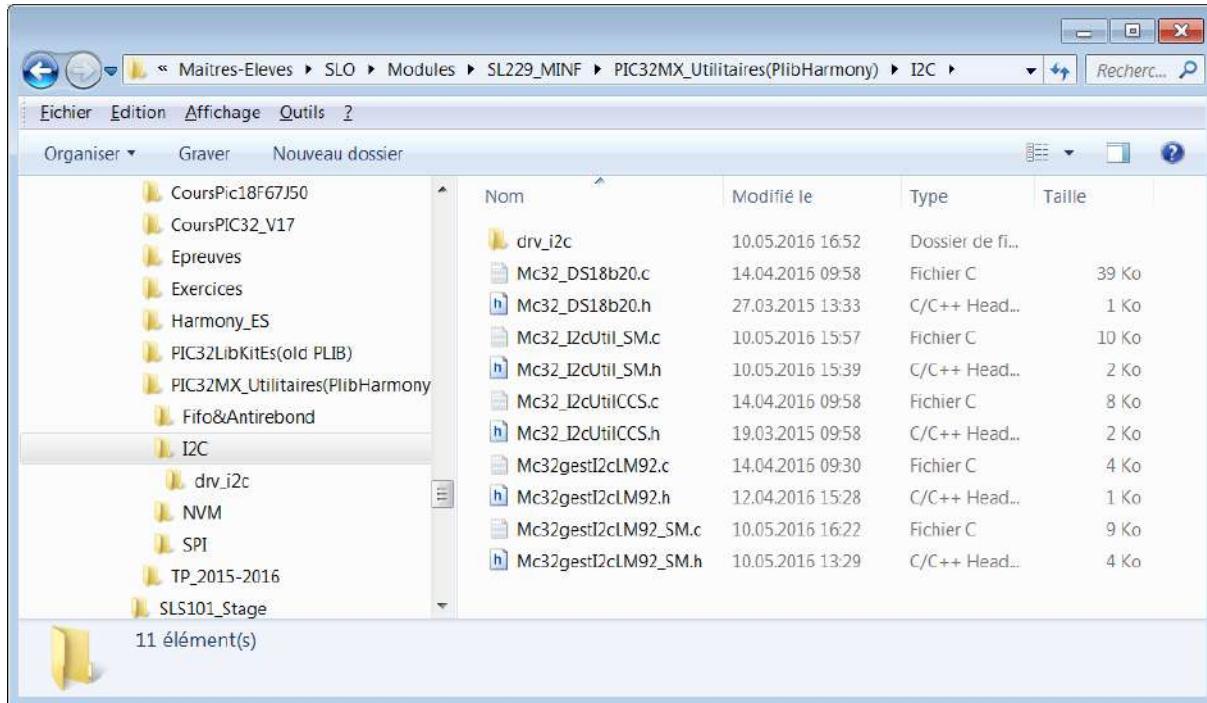
```
case IC_EEPROM:  
    // Appel cyclique traitement SEEPROM  
    I2C_EEPROM_SM_Execute(&DescrEEprom);  
    if (I2C_EEPROM_SM_IsReady(&DescrEEprom)) {  
        ComposantEnCours = IC_LM92;  
    }  
    break;  
} // end switch
```

Utilisation de la situation IsReady indiquant la fin de séquence pour passer au traitement suivant.

## 8.5. LOCALISATION DES FICHIERS I2C

Pour permettre un accès plus facile aux différentes librairies, vous trouverez sous :

...\\Maitres-Eleves\\SLO\\Modules\\SL229\_MINF\\PIC32MX\_Utilitaires(PlibHarmony)\\I2C les fichiers suivants :



On y trouve les deux familles d'utilitaires, l'une avec le traitement standard et l'autre avec machine d'état. Le traitement machine d'état du DS18B20 n'est pas fourni.

Le répertoire `drv_i2c` contient une copie du driver I2C avec les modifications.

## 8.6. CONCLUSION

Ce document devrait permettre, en s'inspirant des principes utilisés pour la gestion par machine d'état du LM92, de s'adapter à la gestion en machine d'état d'autres composants I2C.

## 8.7. HISTORIQUE DES VERSIONS

### 8.7.1. VERSION 1.5 MAI 2015

Reprise et transformation en cours T.P. de la partie SM du chapitre 9 de théorie version de mars 2014. Utilisation du driver I2C de Harmony 1.03.

### 8.7.2. VERSION 1.6 MAI 2016

Adaptation à Harmony 1.06 (très peu de changements par rapport à 1.03). Mesures refaites avec le projet Harmony 1.06.

### 8.7.3. VERSION 1.7 AVRIL 2017

Relecture générale par SCA. Test avec Harmony 1.08 et mise à jour.

### 8.7.1. VERSION 1.71 FÉVRIER 2022

Enlevé références à CCS et PIC18.

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

# **Chapitre 9**

## **Gestion du bus USB**

**❖ T.P. PIC32MX**

**Christian HUBER (CHR)**  
**Serge CASTOLDI (SCA)**  
**Version 1.71 mars 2019**



## CONTENU DU CHAPITRE 9

<b>9. Gestion du bus USB</b>	<b>9-1</b>
<b>9.1. Hardware USB sur le KIT PIC32MX95F512L</b>	<b>9-1</b>
<b>9.2. Utilisation d'un exemple</b>	<b>9-2</b>
9.2.1. Ouverture du projet et adaptation	9-2
9.2.1.1. Ouverture du projet	9-2
9.2.1.2. Renommage du projet	9-2
9.2.1.3. Choix de la configuration	9-3
9.2.2. Réduction à une seule configuration	9-4
9.2.3. Sélection du bsp du kit	9-5
9.2.4. Adaptation du device configuration pour le Kit	9-6
9.2.4.1. Section DEVCFG3	9-6
9.2.4.2. Section DEVCFG2	9-6
9.2.4.3. Section DEVCFG1	9-6
9.2.4.4. Section DEVCFG0	9-6
9.2.5. Génération du code	9-7
9.2.6. Test de compilation	9-7
9.2.7. Nettoyage du projet	9-7
9.2.8. Test de la communication USB	9-8
9.2.8.1. Test avec Putty	9-8
9.2.9. Détails de fonctionnement	9-10
9.2.9.1. Code	9-10
9.2.9.2. Machine d'état	9-13
9.2.9.3. Déroulement d'une lecture	9-14
9.2.9.4. Déroulement d'une écriture	9-14
9.2.9.5. Etat de l'USB	9-15
<b>9.3. Ajout de traitement particulier</b>	<b>9-16</b>
<b>9.4. Exemple d'application</b>	<b>9-17</b>
9.4.1. Initialisation	9-17
9.4.2. Affichage des messages reçus	9-17
9.4.3. Envoi des messages	9-18
9.4.3.1. Contenu initial APP_STATE_SCHEDULE_WRITE	9-18
9.4.3.2. Contenu modifié de APP_STATE_SCHEDULE_WRITE	9-19
9.4.4. Situation avec application Windows	9-20
<b>9.5. Conclusion</b>	<b>9-21</b>
<b>9.6. Historique des versions</b>	<b>9-21</b>
9.6.1. Version 1.5 avril 2015	9-21
9.6.2. Version 1.6 avril 2016	9-21
9.6.3. Version 1.7 mars 2017	9-21
9.6.1. Version 1.71 mars 2019	9-21



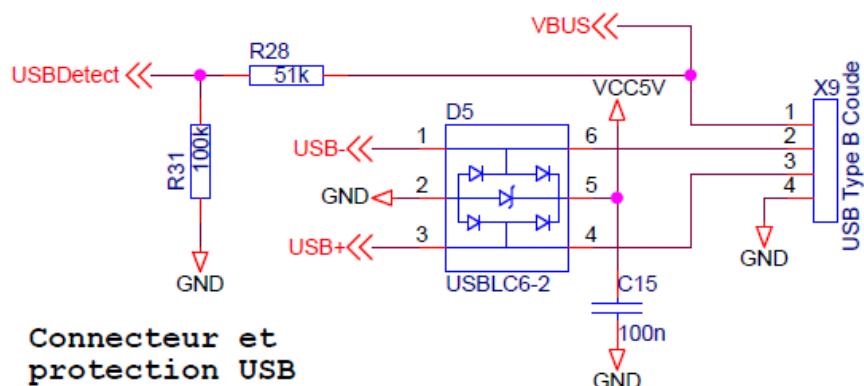
## 9. GESTION DU BUS USB

Ce chapitre a pour objectif de découvrir la gestion du bus USB fourni par Harmony et le MHC.

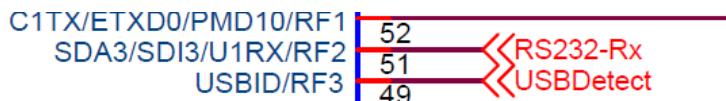
L'objectif est de réaliser un USB device de type cdc (Communication Device Class) qui permettra finalement de gérer l'USB comme un port de communication virtuel.

### 9.1. HARDWARE USB SUR LE KIT PIC32MX795F512L

Voici la circuiterie USB du kit PIC32MX795F512L :



L'entrée USBDetect est une E/S digitale qui correspond à RF3 broche 51.



## 9.2. UTILISATION D'UN EXEMPLE

Bien qu'il soit possible avec le MHC d'utiliser la USB Library et que l'on obtienne les drivers USB, il reste encore passablement de travail pour obtenir une application complète.

Pour découvrir l'USB, nous allons utiliser l'exemple qui se trouve sous :  
 <Répertoire Harmony>\v<n>\apps\usb\device\cdc\_com\_port\_single.

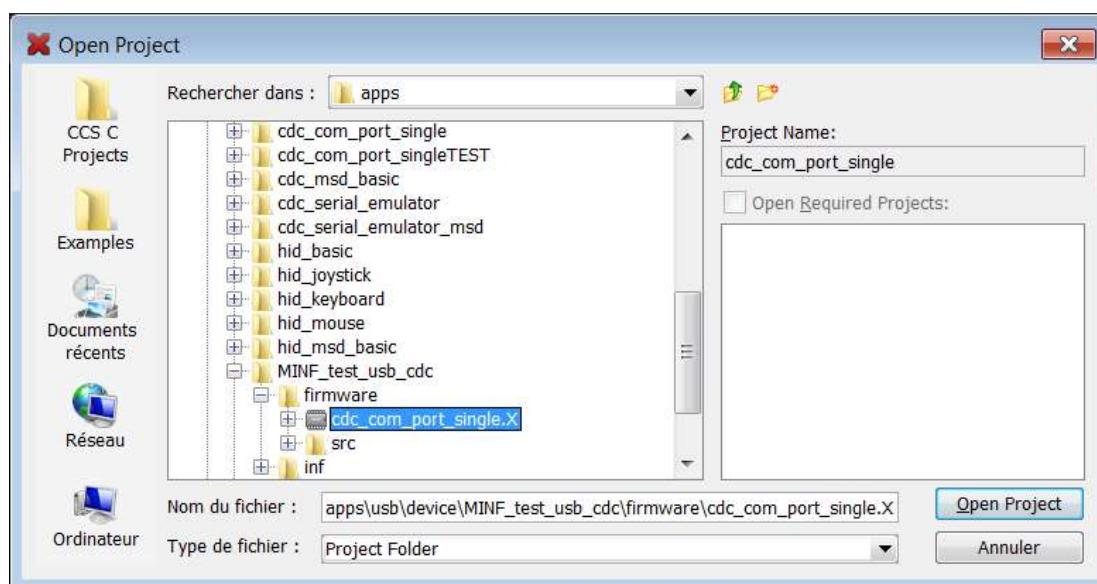
Pour ne pas modifier le contenu de l'exemple, nous copions le répertoire cdc\_com\_port\_single au même niveau et nous renommons la copie en MINF\_test\_usb\_cdc.

Le portage sur le kit SKES du projet suivant a été réalisé avec les logiciels suivants :

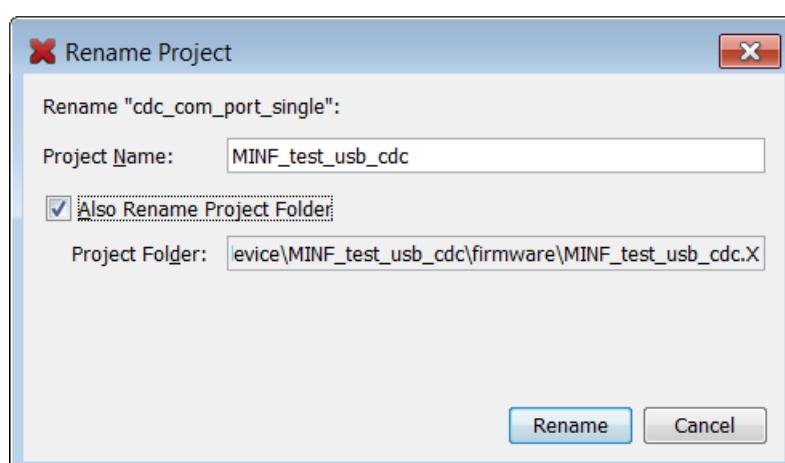
- Harmony v1\_06
- MPLABX IDE v3.10
- XC32 v1.40

### 9.2.1. OUVERTURE DU PROJET ET ADAPTATION

#### 9.2.1.1. OUVERTURE DU PROJET



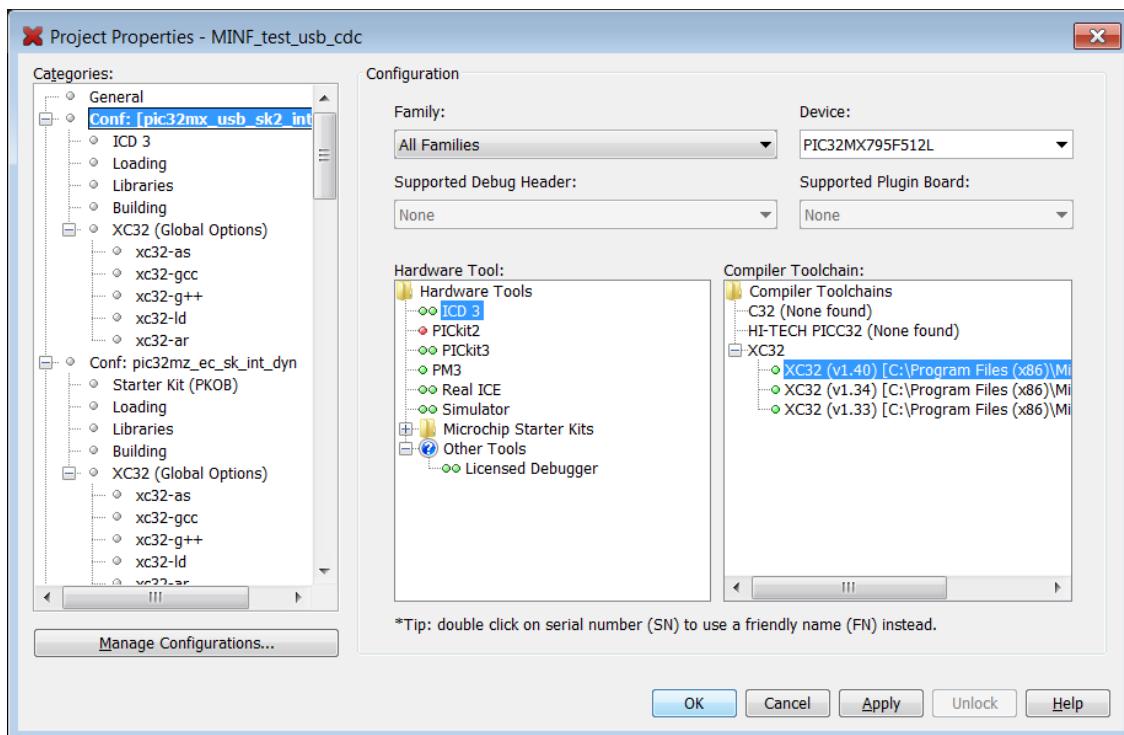
#### 9.2.1.2. RENOMMAGE DU PROJET



### 9.2.1.3. CHOIX DE LA CONFIGURATION

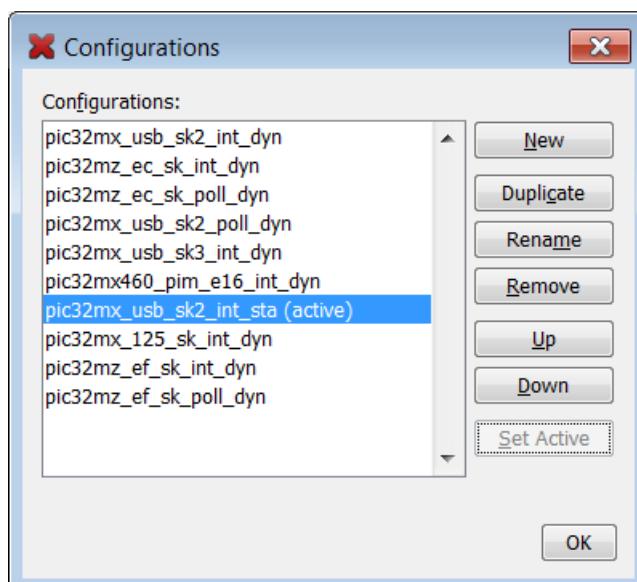
La particularité du projet fourni est qu'il s'agit d'un projet multi configuration et multi bsp.

Dans les propriétés du projet on découvre les multiples configurations.



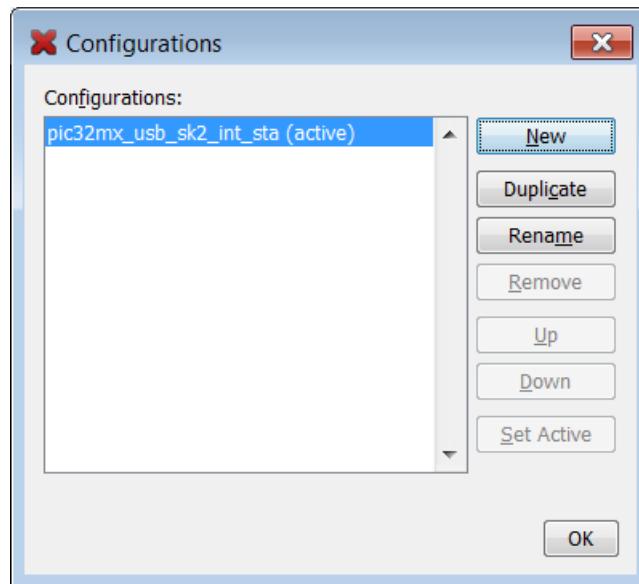
Avec **Manage Configurations...** on peut choisir la configuration active.

Choix de **pic32mx\_usb\_sk2\_int\_sta** en utilisant le bouton "Set Active" :

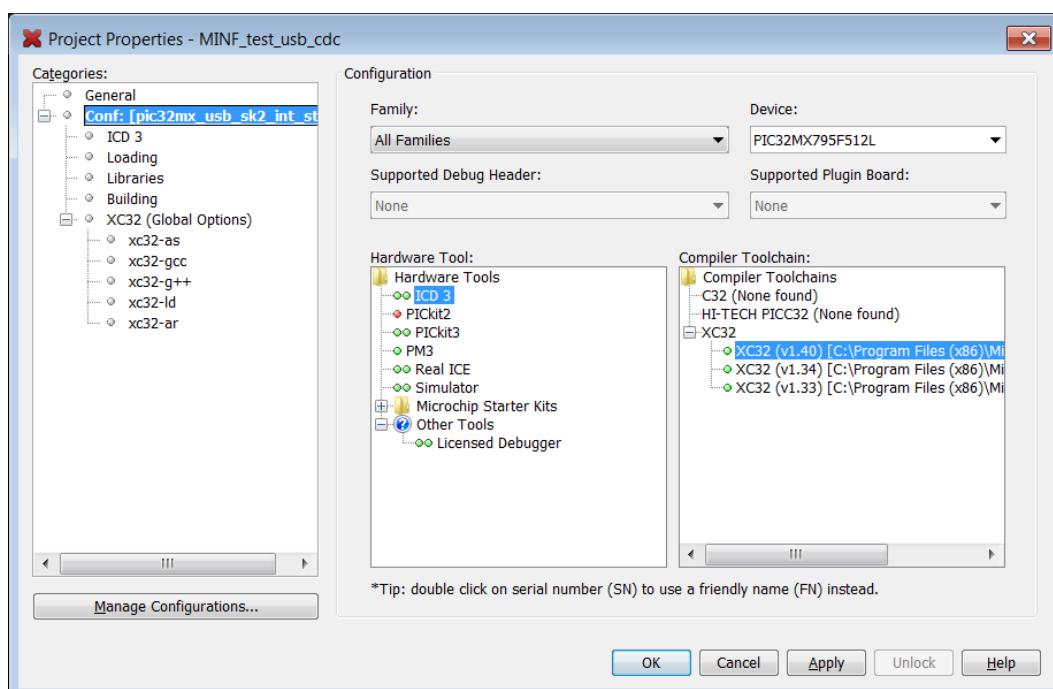


### 9.2.2. REDUCTION A UNE SEULE CONFIGURATION

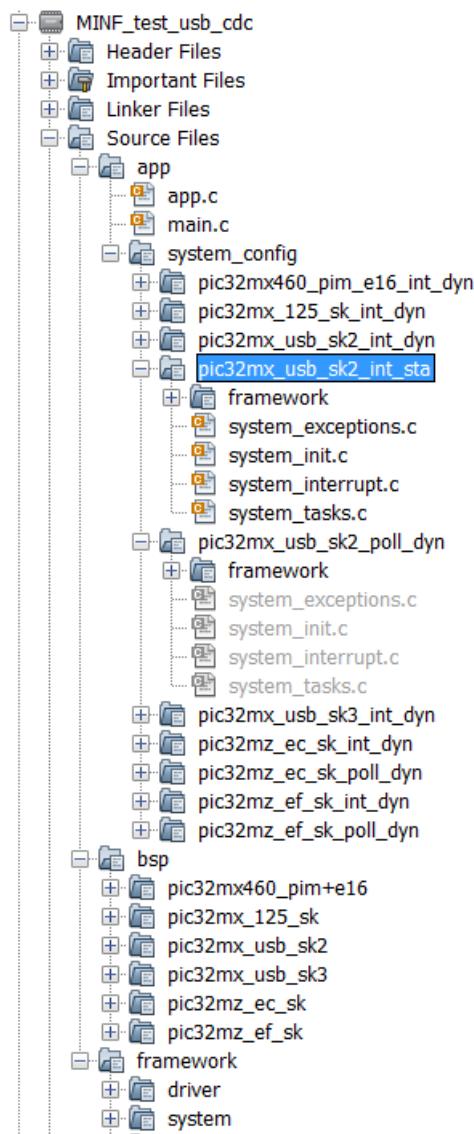
En utilisant le bouton Remove, on enlève toutes les autres configurations.



Ce qui nous ramène à un projet à configuration unique :



Par contre, cela ne supprime pas les répertoires. Pour les autres configurations les fichiers sont en grisé.



### 9.2.3. SELECTION DU BSP DU KIT

Afin de pouvoir sélectionner le bsp du kit, il est nécessaire d'ouvrir le MHC :



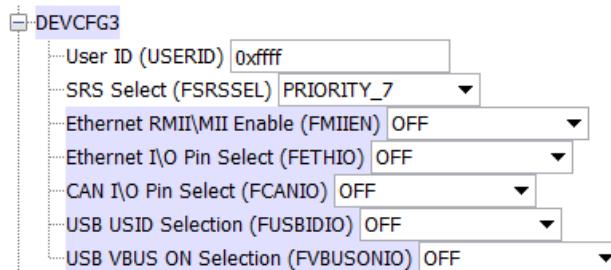
On coche le BSP du Kit ETML-ES.

## 9.2.4. ADAPTATION DU DEVICE CONFIGURATION POUR LE KIT

Les adaptations sont très modestes, le kit usb sk2 de Microchip utilise le même PIC32MX.

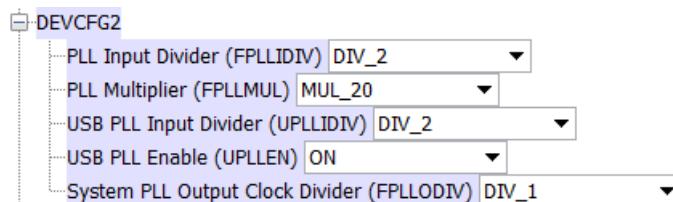
### 9.2.4.1. SECTION DEVCFG3

Pas de modification nécessaire.



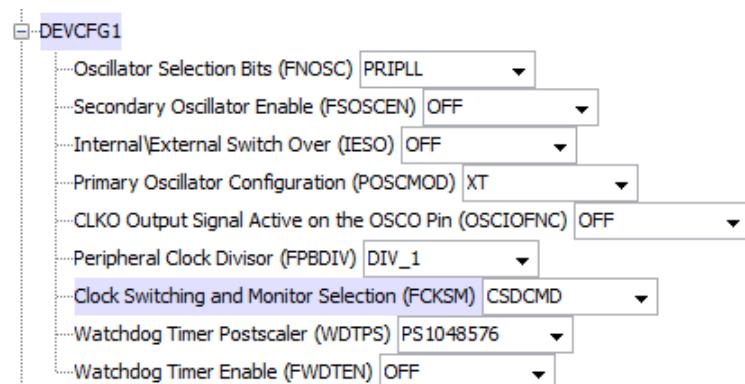
### 9.2.4.2. SECTION DEVCFG2

Pas de modification nécessaire.



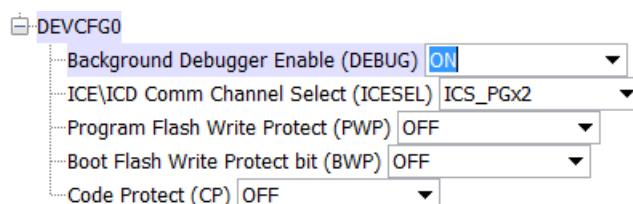
### 9.2.4.3. SECTION DEVCFG1

Pas de modification nécessaire.



### 9.2.4.4. SECTION DEVCFG0

On met le Debugger sur ON.



### 9.2.5. GENERATION DU CODE

Lors de la génération du code, il faut ajouter les éléments qui apparaissent dans la fenêtre de gauche. Par contre, ne pas supprimer les éléments en plus dans la partie de droite.

### 9.2.6. TEST DE COMPILEDATION

Le résultat du clean and build est OK.

```

Output [ ] Configuration Loading Error [ ] MINF_test_usb_cdc (Clean, Build, ...)

" C:\Program Files (x86)\Microchip\pic32\v1_40\bin\xc32-gcc.exe" -g -x c -c -mprocessor=32MX795FS12L -ffunction-sections -O1 -I ..\src -I ..\..\..\..\..\..\framework -I ..\src\system_config\pic32mx795fs12l -o dist\pic32mx_usb_sk2_int_sts\production\MINF_test_usb_cdc.X.production.elf build\pic32mx_usb_sk2_int_sts\production\MINF_test_usb_cdc.X.production.elf
" C:\Program Files (x86)\Microchip\pic32\v1_40\bin\xc32-gcc.exe" -mprocessor=32MX795FS12L -o dist\pic32mx_usb_sk2_int_sts\production\MINF_test_usb_cdc.X.production.elf
make[2]: Leaving directory 'C:/microchip/harmony/v1_06/apps/usb/device/MINF_test_usb_cdc/firmware/MINF_test_usb_cdc.X'
make[1]: Leaving directory 'C:/microchip/harmony/v1_06/apps/usb/device/MINF_test_usb_cdc/firmware/MINF_test_usb_cdc.X'

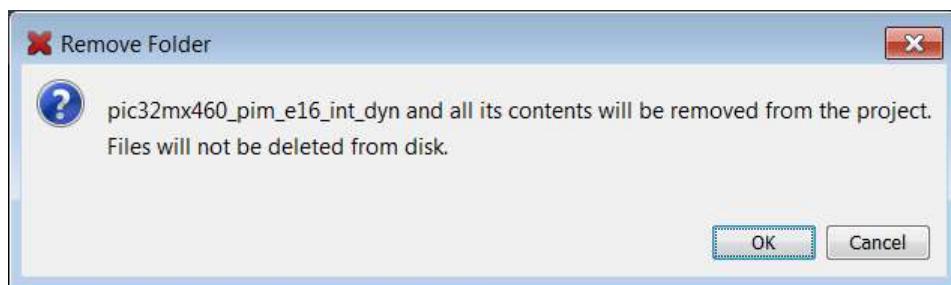
BUILD SUCCESSFUL (total time: 11s)
Loading code from C:/microchip/harmony/v1_06/apps/usb/device/MINF_test_usb_cdc/firmware/MINF_test_usb_cdc.X/dist/pic32mx_usb_sk2_int_sts\production\MINF_test_usb_cdc.X.production.hex...
Loading completed

```

### 9.2.7. NETTOYAGE DU PROJET

Pour éviter une arborescence encombrée, il est possible de faire de l'ordre en utilisant "Remove from project".

Pour un élément on obtient le message suivant :



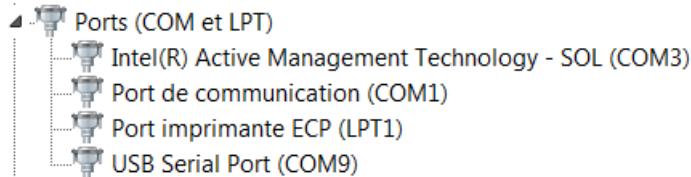
On procèdera de même pour les sections :

Header Files : system\_config et bsp

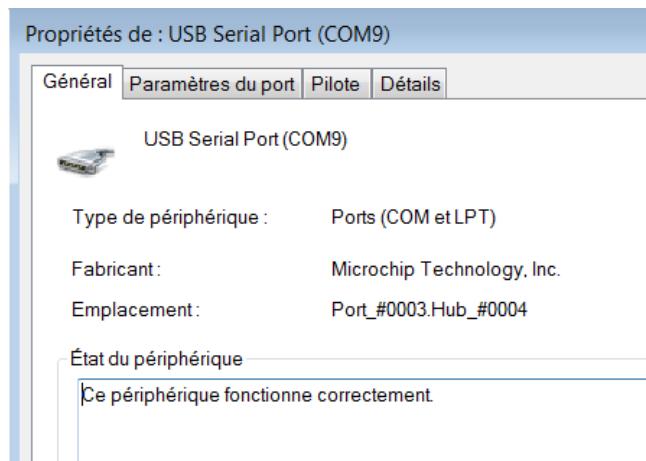
Sources Files : system\_config et bsp

### 9.2.8. TEST DE LA COMMUNICATION USB

En utilisant un port USB 2.0 sur le PC, on obtient l'installation d'un pilote et dans le gestionnaire de périphériques on trouve un USB Serial Port.

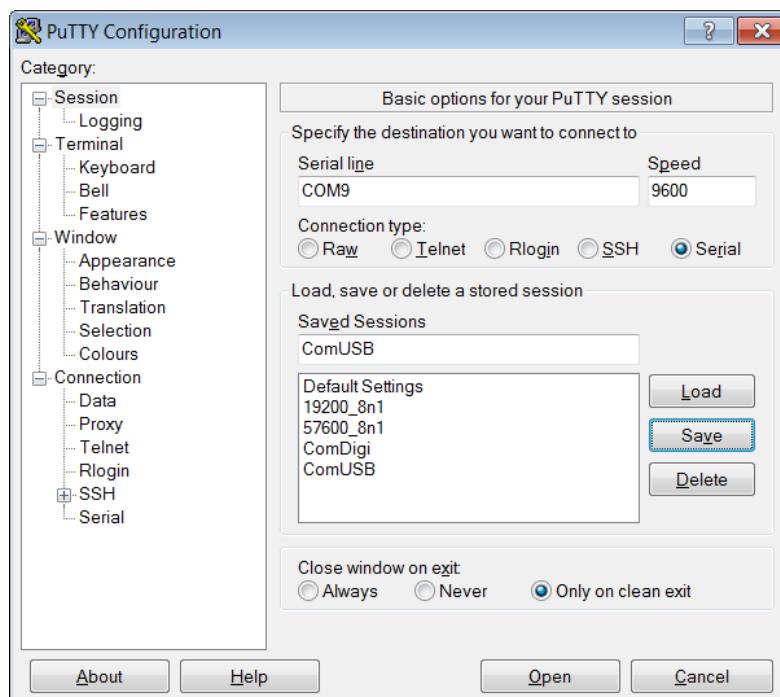


Et c'est bien un élément fourni par Microchip.

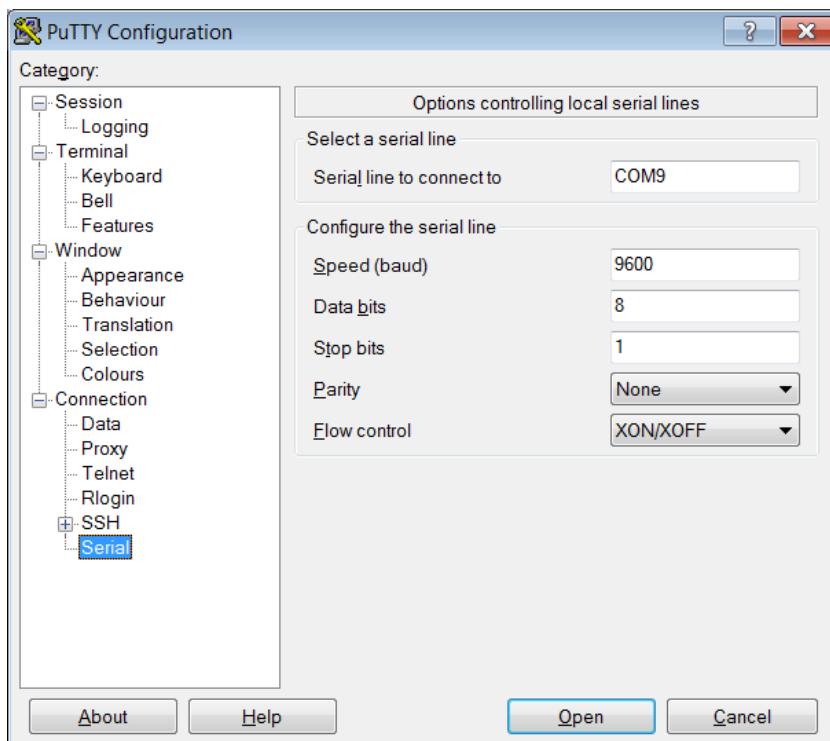


#### 9.2.8.1. TEST AVEC PUTTY

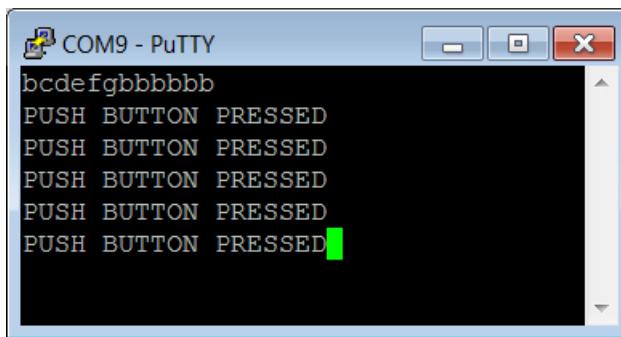
Dans cet exemple, le port série virtuel est le COM9 :



La configuration par défaut fonctionne !



Lorsque l'on presse sur le BSP\_SWITCH\_1 (touche OK du Kit) le message PUSH\_BUTTON\_PRESSED s'affiche. Lors d'une frappe au clavier on reçoit le caractère + 1, donc la touche 'a' affiche 'b'.



Cela montre que le système fonctionne.

### 9.2.9. DETAILS DE FONCTIONNEMENT

Les exemples de codes suivants sont tirés du projet testé avec les logiciels suivants :

- Harmony v1\_08
- MPLABX IDE v3.40
- XC32 v1.42

#### 9.2.9.1. CODE

L'USB a une machine d'état dédiée, comme une application standard. En voici le code :

```

void APP_Tasks (void )
{
    /* Update the application state machine based
     * on the current state */

    switch(appData.state)
    {
        case APP_STATE_INIT:

            /* Open the device layer */
            appData.deviceHandle = USB_DEVICE_Open( USB_DEVICE_INDEX_0,
                DRV_IO_INTENT_READWRITE );

            if(appData.deviceHandle != USB_DEVICE_HANDLE_INVALID)
            {
                /* Register a callback with device layer to get event
                 * notification (for end point 0) */
                USB_DEVICE_EventHandlerSet(appData.deviceHandle,
                    APP_USBDeviceEventHandler, 0);

                appData.state = APP_STATE_WAIT_FOR_CONFIGURATION;
            }
            else
            {
                /* The Device Layer is not ready to be opened. We should try
                 * again later. */
            }
            break;

        case APP_STATE_WAIT_FOR_CONFIGURATION:

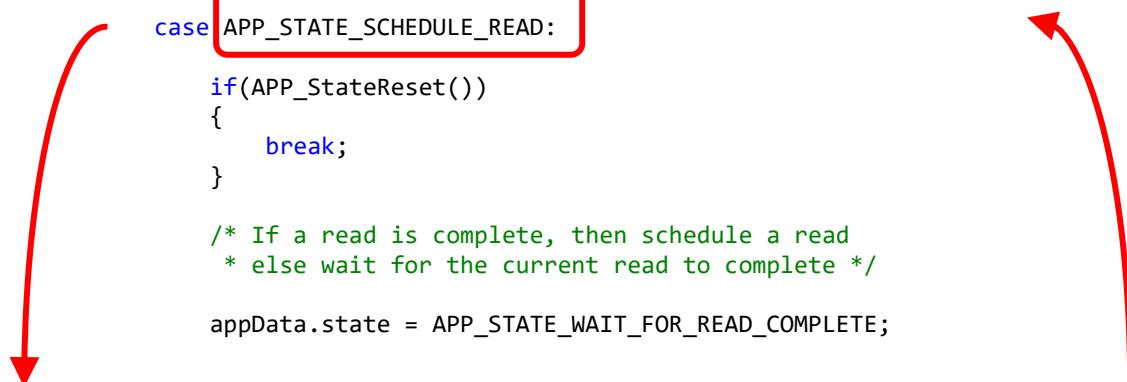
            /* Check if the device was configured */
            if(appData.isConfigured)
            {
                /* If the device is configured then lets start reading */
                appData.state = APP_STATE_SCHEDULE_READ;
            }
            break;

        case APP_STATE_SCHEDULE_READ:
            if(APP_StateReset())
            {
                break;
            }

            /* If a read is complete, then schedule a read
             * else wait for the current read to complete */

            appData.state = APP_STATE_WAIT_FOR_READ_COMPLETE;
    }
}

```



```
if(appData.isReadComplete == true)
{
    appData.isReadComplete = false;
    appData.readTransferHandle =
        USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID;

    USB_DEVICE_CDC_Read (USB_DEVICE_CDC_INDEX_0,
        &appData.readTransferHandle, appData.readBuffer,
        APP_READ_BUFFER_SIZE);

    if(appData.readTransferHandle ==
        USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID)
    {
        appData.state = APP_STATE_ERROR;
        break;
    }
}
break;

case APP_STATE_WAIT_FOR_READ_COMPLETE:
case APP_STATE_CHECK_SWITCH_PRESSED:

    if(APP_StateReset())
    {
        break;
    }

    APP_ProcessSwitchPress();

    /* Check if a character was received or a switch was pressed.
     * The isReadComplete flag gets updated in the CDC event handler. */

    if(appData.isReadComplete || appData.isSwitchPressed)
    {
        appData.state = APP_STATE_SCHEDULE_WRITE;
    }
    break;

case APP_STATE_SCHEDULE_WRITE:

    if(APP_StateReset())
    {
        break;
    }

    /* Setup the write */

    appData.writeTransferHandle = USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID;
    appData.isWriteComplete = false;
    appData.state = APP_STATE_WAIT_FOR_WRITE_COMPLETE;

    if(appData.isSwitchPressed)
    {
        /* If the switch was pressed, then send the switch prompt*/
        appData.isSwitchPressed = false;
        USB_DEVICE_CDC_Write(USB_DEVICE_CDC_INDEX_0,
            &appData.writeTransferHandle, switchPromptUSB, 23,
            USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);
    }
    else
    {
        /* Else echo the received character + 1*/
```

```
        appData.readBuffer[0] = appData.readBuffer[0] + 1;
        USB_DEVICE_CDC_Write(USB_DEVICE_CDC_INDEX_0,
            &appData.writeTransferHandle,
            appData.readBuffer, 1,
            USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);
    }
    break;

case APP_STATE_WAIT_FOR_WRITE_COMPLETE:
    if(APP_StateReset())
    {
        break;
    }

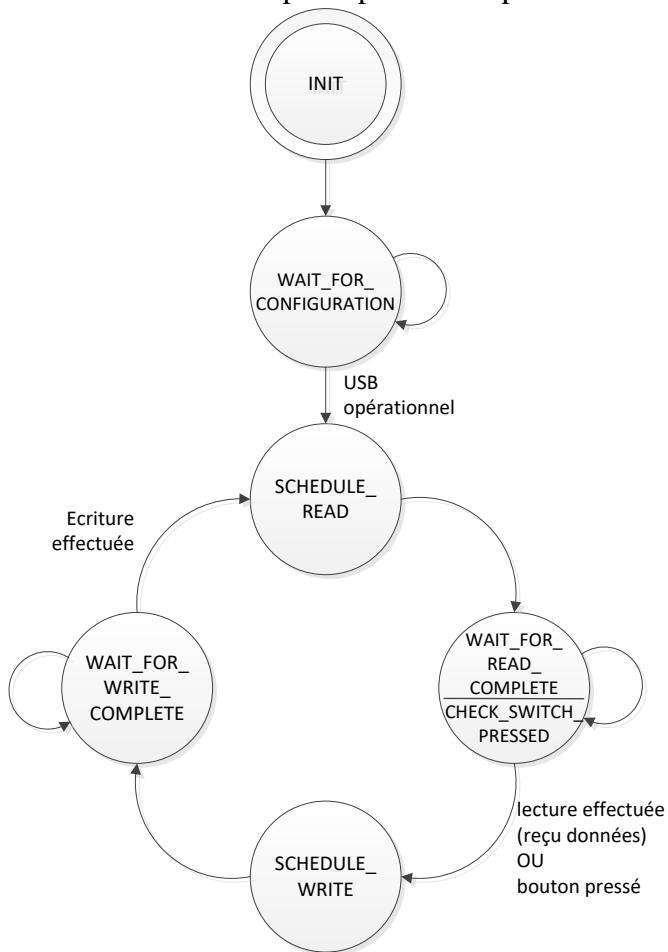
    /* Check if a character was sent. The isWriteComplete
     * flag gets updated in the CDC event handler */

    if(appData.isWriteComplete == true)
    {
        appData.state = APP_STATE_SCHEDULE_READ;
    }
    break;

case APP_STATE_ERROR:
    break;
default:
    break;
}
}
```

### 9.2.9.2. MACHINE D'ETAT

Son fonctionnement simplifié peut être représenté ainsi :



Par soucis de simplification, les transitions de retour depuis chaque état de lecture ou écriture vers l'état WAIT\_FOR\_CONFIGURATION n'ont pas été représentées.  
Ces transitions ont lieu en cas de réinitialisation de l'USB.

Les noms des états étant judicieusement choisis, on peut comprendre que l'application passe son temps à faire des lectures, puis dès qu'une donnée est reçue ou que le bouton a été appuyé, elle procède à une écriture. Et la boucle recommence.  
Cela correspond effectivement au comportement observé précédemment.

Partant de cet exemple, un fonctionnement de réception d'une trame, puis de renvoi d'une réponse peut facilement être implémenté. A la manière du bouton pressé, on peut également transmettre spontanément une trame (sans besoin de réception préalable).

### 9.2.9.3. DEROULEMENT D'UNE LECTURE

Pour effectuer une réception de données via USB, la fonction suivante est utilisée :

```
USB_DEVICE_CDC_Read (USB_DEVICE_CDC_INDEX_0,
                      &appData.readTransferHandle,
                      appData.readBuffer,
                      APP_READ_BUFFER_SIZE);
```

Paramètres :

- Les deux premiers paramètres sont à conserver tels quels.
- Le paramètre n°3 est un pointeur sur un buffer où peuvent être stockées les données reçues.
- Le paramètre n°4 représente le nombre d'octets maximum à lire.

Cette fonction fait une requête de lecture au driver USB. Elle n'effectue pas la lecture ! Dans le fonctionnement USB, le master est le PC, le périphérique ne peut pas initier des communications.

On doit ensuite contrôler si la lecture a été effectuée (et donc on pourrait lire le contenu du buffer) ainsi :

```
if (appData.isReadComplete == true)
```

### 9.2.9.4. DEROULEMENT D'UNE ECRITURE

De la même manière, pour effectuer une requête d'écriture de données :

```
USB_DEVICE_CDC_Write(USB_DEVICE_CDC_INDEX_0,
                      &appData.writeTransferHandle,
                      switchPromptUSB,
                      23,
                      USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_
                      COMPLETE);
```

Paramètres :

- Les deux premiers paramètres sont à conserver tels quels.
- Le paramètre n°3 est un pointeur sur un buffer où sont stockées les données à envoyer.
- Le paramètre n°4 représente le nombre d'octets à envoyer.
- Le paramètre n°5 indique qu'on ne désire pas compléter l'envoi par des données supplémentaires, et donc que l'envoi des données peut être effectué.

Comme dans le cas de la lecture, cette fonction fait uniquement une requête d'écriture au driver USB. Elle n'effectue pas l'écriture !

On doit ensuite contrôler si l'écriture a été effectuée :

```
if (appData.isWriteComplete == true)
```

La documentation concernant ces fonctions de lecture et écriture est tirée du l'aide Harmony disponible sous :

<Répertoire Harmony>\v<n>\doc\help\_harmony.chm

A la rubrique

MPLAB Harmony Framework Reference > USB Libraries Help > USB Device Library > USB CDC Device Library > Library Interface > a) Functions

### 9.2.9.5. ETAT DE L'USB

L'application principale devra très certainement pouvoir connaître l'état de l'USB. Cet état peut être :

- 1) Reset (déconnecté),
- 2) Configured (en fonction).
- 3) Suspended (périphérique en mode basse consommation),

L'état "configured" est le seul où des transferts peuvent avoir lieu.

En allant voir le code du gestionnaire d'événements USB (USBDeviceEventHandler() - dans le même fichier), on trouve 3 endroits :

```
*****
 * Application USB Device Layer Event Handler.
 ****
void APP_USBDeviceEventHandler ( USB_DEVICE_EVENT event, void * eventData,
uintptr_t context )
{
    USB_DEVICE_EVENT_DATA_CONFIGURED *configuredEventData;

    switch ( event )
    {
        case USB_DEVICE_EVENT_SOF:

            /* This event is used for switch debounce. This flag is reset
             * by the switch process routine. */
            appData.sofEventHasOccurred = true;
            break;

        case USB_DEVICE_EVENT_RESET:

            /* Update LED to show reset state */
            BSP_LEDOn ( APP_USB_LED_1 );
            BSP_LEDOn ( APP_USB_LED_2 );
            BSP_LEDOff ( APP_USB_LED_3 );

            appData.isConfigured = false;

            break;

        case USB_DEVICE_EVENT_CONFIGURED:

            /* Check the configuration. We only support configuration 1 */
            configuredEventData = (USB_DEVICE_EVENT_DATA_CONFIGURED*)eventData;
            if ( configuredEventData->configurationValue == 1 )
            {
                /* Update LED to show configured state */
                BSP_LEDOff ( APP_USB_LED_1 );
                BSP_LEDOff ( APP_USB_LED_2 );
                BSP_LEDOn ( APP_USB_LED_3 );
            }
    }
}
```

1

2

```

    /* Register the CDC Device application event handler here.
     * Note how the appData object pointer is passed as the
     * user data */

    USB_DEVICE_CDC_EventHandlerSet(USB_DEVICE_CDC_INDEX_0,
APP_USBDeviceCDCEventHandler, (uintptr_t)&appData);

    /* Mark that the device is now configured */
    appData.isConfigured = true;

}

break;

case USB_DEVICE_EVENT_POWER_DETECTED:

    /* VBUS was detected. We can attach the device */
    USB_DEVICE_Attach(appData.deviceHandle);
break;

case USB_DEVICE_EVENT_POWER_REMOVED:

    /* VBUS is not available any more. Detach the device. */
    USB_DEVICE_Detach(appData.deviceHandle);
break;

case USB_DEVICE_EVENT_SUSPENDED:

    /* Switch LED to show suspended state */
    BSP_LEDOff ( APP_USB_LED_1 );
    BSP_LEDOn ( APP_USB_LED_2 );
    BSP_LEDOn ( APP_USB_LED_3 );
break;

case USB_DEVICE_EVENT_RESUMED:
case USB_DEVICE_EVENT_ERROR:
default:
    break;
}
}

```



De ces portions de code (ne pas oublier l'initialisation également), on peut facilement déduire l'état de l'USB et renseigner notre application principale.

### 9.3. AJOUT DE TRAITEMENT PARTICULIER

Dans la perspective d'intégrer une application existante à la communication USB, il est nécessaire d'introduire la possibilité de déclencher cycliquement une action d'application et d'introduire des interruptions supplémentaires.

La machine d'état de l'application issue de l'exemple USB a des états spécifiques au fonctionnement USB. Pour permettre le traitement habituel avec les 3 états (INIT, WAIT, SERVICE\_TASKS), la solution consiste à ajouter une deuxième application distincte avec sa machine d'état. Il faudra prévoir des variables ou fonctions pour se passer les informations entre les 2 machines d'état.

Ceci va permettre, en relation avec system\_interrupt.c, d'activer cycliquement notre application et d'y ajouter les traitements spécifiques.

## 9.4. EXEMPLE D'APPLICATION

Il s'agit maintenant d'étudier l'application et de déterminer comment récupérer les données reçues par USB et en envoyer.

Nous allons réutiliser une application prévue pour le test de l'UART. Cette application envoie des messages de type texte, formatés de la manière suivante :

```
strMess = "!Ch0 " & NudCanal0.Value & " Ch1 " & NudCanal1.Value & "#"
```

### 9.4.1. INITIALISATION

Au début du case APP\_STATE\_INIT: nous ajoutons :

```
// Init pour Kit pic32mx_sk8s
lcd_init();
lcd_bl_on();

// Init AD
BSP_InitADC10();

printf_lcd("MINF_test USB CDC");
lcd_gotoxy(1,2);
printf_lcd("CHR 21.04.2016    ");
```

Ceci avant :

```
/* Open the device layer */
```

### 9.4.2. AFFICHAGE DES MESSAGES REÇUS

Dans un 1<sup>er</sup> temps, nous allons afficher le message reçu sur la 4<sup>ème</sup> ligne de l'afficheur LCD.

C'est dans le case APP\_STATE\_WAIT\_FOR\_READ\_COMPLETE que nous pouvons récupérer le message entrant.

```
case APP_STATE_WAIT_FOR_READ_COMPLETE:
case APP_STATE_CHECK_SWITCH_PRESSED:

    if(APP_StateReset())
    {
        break;
    }

    APP_ProcessSwitchPress();

    /* Check if a character was received or a switch
       was pressed. The isReadComplete flag gets updated in
       the CDC event handler. */

    if(appData.isReadComplete || appData.isSwitchPressed)
    {

        appData.state = APP_STATE_SCHEDULE_WRITE;
        lcd_gotoxy(1,4);
        printf_lcd("%s", appData.readBuffer );
    }
break;
```

:( Il n'y a pas d'élément pour connaître le nombre d'octets reçus. En principe la trame USB est de 64 octets au maximum.

### 9.4.3. ENVOI DES MESSAGES

Création d'un message contenant la valeur des 2 canaux de l'AD.

C'est dans le case APP\_STATE\_SCHEDULE\_WRITE que nous plaçons l'émission du message.

#### 9.4.3.1. CONTENU INITIAL APP\_STATE\_SCHEDULE\_WRITE

Voici le contenu non modifié du case APP\_STATE\_SCHEDULE\_WRITE:

```
case APP_STATE_SCHEDULE_WRITE:
    if(APP_StateReset())
    {
        break;
    }

    /* Setup the write */
    appData.writeTransferHandle =
        USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID;
    appData.isWriteComplete = false;
    appData.state = APP_STATE_WAIT_FOR_WRITE_COMPLETE;

    if(appData.isSwitchPressed)
    {
        /* If the switch was pressed,
           then send the switch prompt*/
        appData.isSwitchPressed = false;
        USB_DEVICE_CDC_Write(USB_DEVICE_CDC_INDEX_0,
                             &appData.writeTransferHandle,
                             switchPromptUSB, 23,
                             USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);
    }
    else
    {
        /* Else echo the received character + 1 */
        appData.readBuffer[0] = appData.readBuffer[0] + 1;
        USB_DEVICE_CDC_Write(USB_DEVICE_CDC_INDEX_0,
                             &appData.writeTransferHandle,
                             appData.readBuffer, 1,
                             USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);
    }
break;
```

#### 9.4.3.2. CONTENU MODIFIÉ DE APP\_STATE\_SCHEDULE\_WRITE

Voici le contenu modifié du case **APP\_STATE\_SCHEDULE\_WRITE**. Suppression de la gestion du switch et traitement pour envoyer un message lorsqu'on en reçoit un.

Ajout des éléments suivants au niveau des données de l'application.

```
APP_DATA appData;
S_ADCResults ValAD;
char SendBuffer[64];

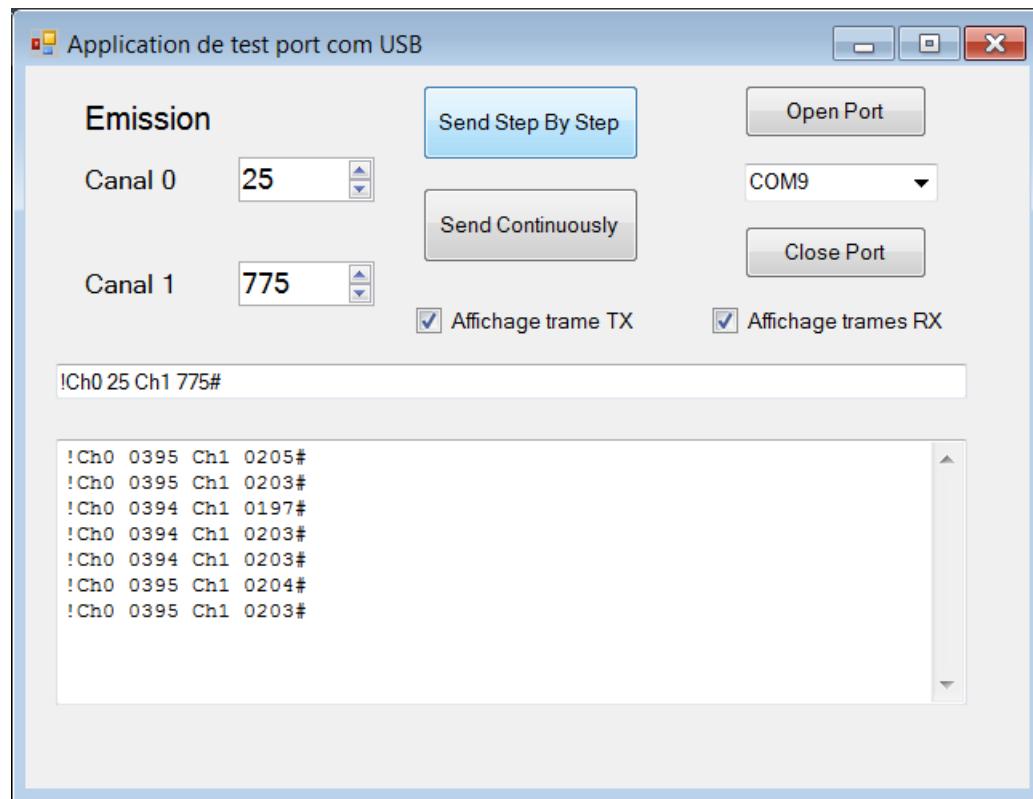
case APP_STATE_SCHEDULE_WRITE:

    if(APP_StateReset())
    {
        break;
    }

    /* Setup the write */
    appData.writeTransferHandle =
        USB_DEVICE_CDC_TRANSFER_HANDLE_INVALID;
    appData.isWriteComplete = false;
    appData.state = APP_STATE_WAIT_FOR_WRITE_COMPLETE;

    // Lecture AD et composition du message
    ValAD = BSP_ReadAllADC();
    sprintf(SendBuffer, "!Ch0 %04d Ch1 %04d#",
            ValAD.Chan0, ValAD.Chan1);
    // Emission du message
    USB_DEVICE_CDC_Write(USB_DEVICE_CDC_INDEX_0,
                         &appData.writeTransferHandle,
                         SendBuffer, strlen(SendBuffer),
                         USB_DEVICE_CDC_TRANSFER_FLAGS_DATA_COMPLETE);
break;
```

#### 9.4.4. SITUATION AVEC APPLICATION WINDOWS



On obtient l'affichage des messages émis par le kit, et sur le kit on obtient l'affichage du message émis par l'application Windows.

Ligne 3 = valeurs envoyées / ligne 4 = valeurs reçues.



## 9.5. CONCLUSION

Ce document devrait permettre, en s'inspirant de la démarche d'adaptation d'un projet USB Microchip, de procéder de même pour d'autres applications et projets particuliers. Dans le cas d'une application finale plus complexe, il peut être très utile de séparer le fonctionnement en plusieurs "app", comme le MHC le permet, chacune ayant sa machine d'état et gérant sa partie.

## 9.6. HISTORIQUE DES VERSIONS

### 9.6.1. VERSION 1.5 AVRIL 2015

Création du document. Version 1.5 pour indiquer l'utilisation de Harmony.

### 9.6.2. VERSION 1.6 AVRIL 2016

Reprise du document avec la situation Harmony 1.06 et MPLABX 3.10. Le fait d'avoir le bsp sélectionnable dans la liste modifie passablement le contenu du document.

### 9.6.3. VERSION 1.7 MARS 2017

Reprise du document par SCA. Relecture. Compléments fonctionnement et reprise projet "cdc\_com\_port\_single".

### 9.6.1. VERSION 1.71 MARS 2019

Compléments mineurs.

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

# **Chapitre 10**

## **Gestion de l’Ethernet et de TCP/IP**

### **❖ T.P. PIC32MX**

**Christian HUBER (CHR)**  
**Serge CASTOLDI (SCA)**  
**Version 1.81 juin 2022**



## CONTENU DU CHAPITRE 10

<b>10.</b>	<b>Gestion de l'Ethernet et de TCP/IP</b>	<b>10-1</b>
<b>10.1.</b>	<b>Hardware Ethernet sur le kit PIC32MX95F512L</b>	<b>10-1</b>
<b>10.2.</b>	<b>Réalisation d'un serveur TCP</b>	<b>10-2</b>
10.2.1.	Utilisation d'un exemple et portage sur le kit	10-2
10.2.2.	Ouverture du projet et simplification	10-2
10.2.3.	Modification de la configuration Harmony	10-4
10.2.3.1.	Test initial	10-6
10.2.3.2.	Conclusion au sujet du portage de l'exemple	10-8
10.2.4.	Intégration du générateur	10-8
10.2.4.1.	Mise en place des fichiers du générateur	10-8
10.2.4.2.	Ajout des timers	10-9
10.2.4.3.	Modification appgen	10-10
10.2.5.	Détails du fonctionnement de l'exemple tcpip_tcp_server	10-11
10.2.5.1.	Détection état du lien TCP	10-12
10.2.5.2.	Réception et envoi de données TCP	10-14
10.2.5.3.	Conclusion au sujet de la modification de l'exemple	10-14
<b>10.3.</b>	<b>Réalisation d'un webserver</b>	<b>10-15</b>
10.3.1.	Utilisation d'un exemple et portage sur le kit	10-15
10.3.1.1.	Ouverture du projet	10-15
10.3.1.2.	Réduction des configurations	10-16
10.3.1.3.	Reduction du projet	10-17
10.3.2.	Modification de la configuration Harmony	10-18
10.3.2.1.	Sélection BSP pic32mx_skes	10-18
10.3.2.2.	Ajout appgen	10-19
10.3.2.3.	Modification device configuration	10-19
10.3.2.4.	Sélection du chip Ethernet physique	10-20
10.3.2.5.	Timers	10-21
10.3.2.6.	Test compilation	10-21
10.3.2.7.	Etat après modifications	10-21
10.3.2.8.	Réduction de la flash nécessaire	10-22
10.3.2.9.	Test initial	10-22
10.3.2.10.	Conclusion au sujet du portage de l'exemple	10-24
10.3.3.	Intégration du générateur	10-25
10.3.3.1.	Mise en place des fichiers du générateur	10-25
10.3.3.2.	Adaptation de system_init.c	10-26
10.3.3.3.	Adaptation de system_interrupt.c	10-27
10.3.3.4.	Adaptation de appgen.c	10-28
10.3.3.5.	Test du générateur	10-29
10.3.4.	Page web personnalisée	10-30
10.3.4.1.	Copie du répertoire web_pages	10-30
10.3.4.2.	Génération des fichiers	10-30
10.3.4.3.	Fichiers modifiés	10-32
10.3.4.4.	Adaptations du fichier custom_http_app.c	10-33
10.3.4.5.	Observation de la page Web	10-39
10.3.4.6.	Traitement en mode Post	10-39
10.3.4.7.	Rafraîchissement automatique de la page web	10-39

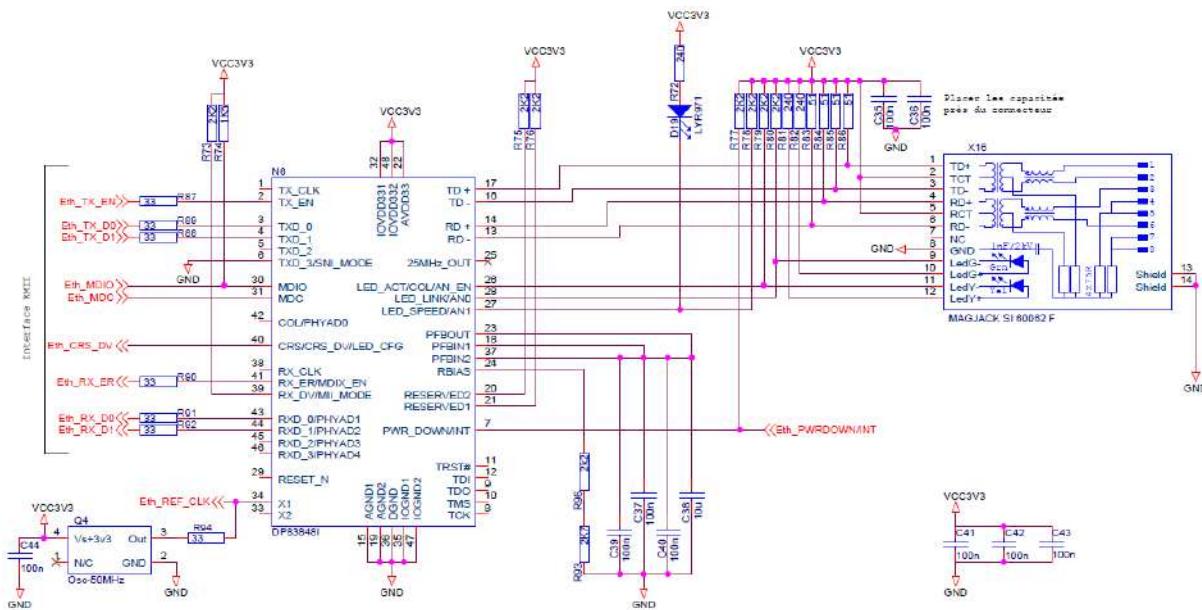
<b>10.4. Conclusion</b>	<b>10-40</b>
<b>10.5. Historique des versions</b>	<b>10-40</b>
10.5.1. Version 1.5 Mai 2015	10-40
10.5.2. Version 1.51 Juin 2015	10-40
10.5.3. Version 1.6 Juin 2016	10-40
10.5.4. Version 1.6 b Juin 2016	10-40
10.5.5. Version 1.7 mai 2017	10-40
10.5.6. Version 1.71 mai 2019	10-40
10.5.7. Version 1.8 avril 2022	10-40
10.5.1. Version 1.81 juin 2022	10-40

## 10. GESTION DE L'ETHERNET ET DE TCP/IP

Ce chapitre a pour objectif de découvrir la gestion de l'Ethernet et de la pile TCP/IP fournies par Harmony.

### 10.1. HARDWARE ETHERNET SUR LE KIT PIC32MX95F512L

Voici la circuiterie Ethernet du kit PIC32MX795F512L qui utilise un circuit DP83848I (10/100 Mb/s Ethernet Physical Layer Transceiver) :



## 10.2. REALISATION D'UN SERVEUR TCP

Cette partie a pour objectif de découvrir comment modifier l'exemple Microchip tcpip\_tcp\_server pour aboutir au pilotage du générateur de signal à partir d'un client TCP distant.

### 10.2.1. UTILISATION D'UN EXEMPLE ET PORTAGE SUR LE KIT

Pour découvrir la réalisation d'un serveur TCP, nous allons utiliser l'exemple qui est sous :  
<Répertoire Harmony>\v<n>\apps\tcpip\tcpip\_tcp\_server

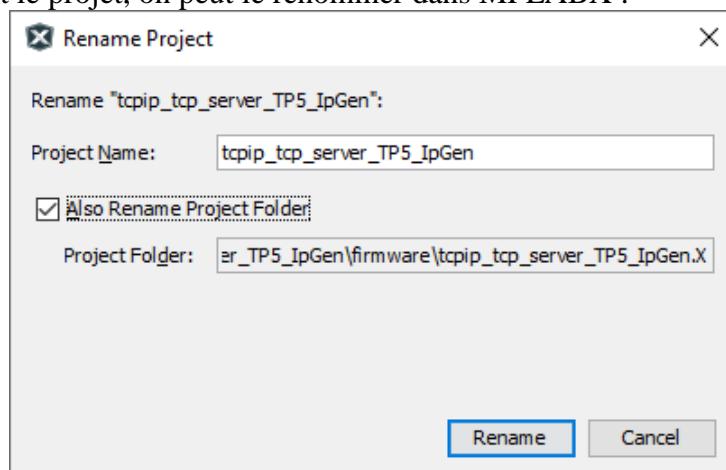
Pour ne pas modifier le contenu de l'exemple, nous copions le répertoire **tcpip\_tcp\_server** au même niveau et nous le renommons **tcpip\_tcp\_server\_TP5\_IpGen**.

Le portage sur le kit SKES du projet suivant et son adaptation ont été réalisés avec les logiciels suivants :

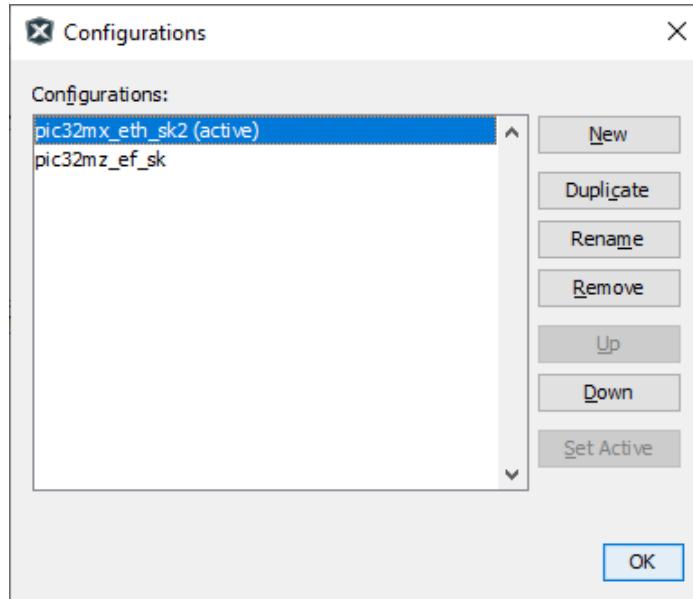
- Harmony v2.06
- MPLABX IDE v5.45
- XC32 v2.50

### 10.2.2. OUVERTURE DU PROJET ET SIMPLIFICATION

Après avoir ouvert le projet, on peut le renommer dans MPLABX :

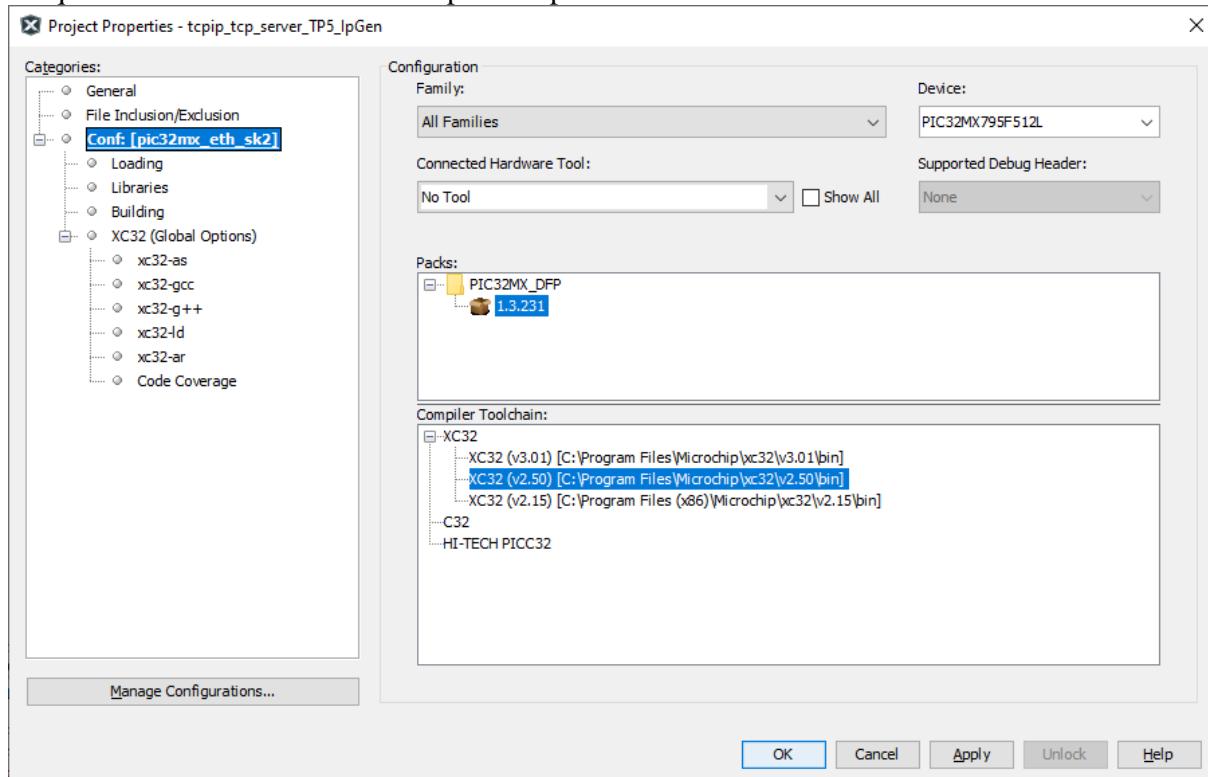


Puis, à partir des propriétés, dans « Manage Configurations », on va supprimer les configurations inutiles :



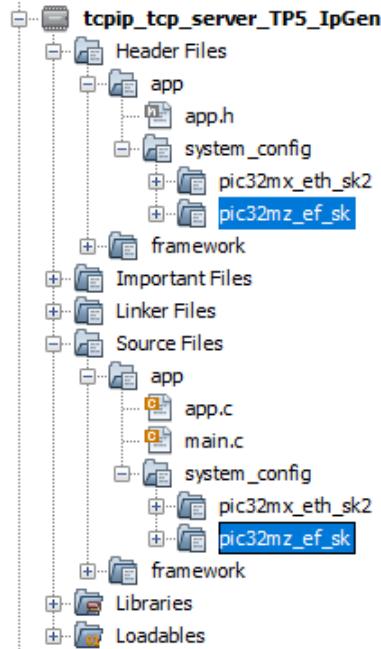
La configuration « pic32mx\_eth\_sk2 » est la plus proche de celle du starter-kit ES. On ne garde que celle-ci.

Ce qui nous ramène à une situation plus simple :



- Au besoin, régler la bonne version du compilateur.
- Ne pas oublier la coche xc32-gcc > preprocessing and messages > additionnal warnings.
- Afin de faciliter le debug, régler le niveau d'optimisation du compilateur à 0.

Dans « Header Files » et « Source Files », on peut également supprimer les fichiers devenus inutiles du fait de la suppression de la configuration :

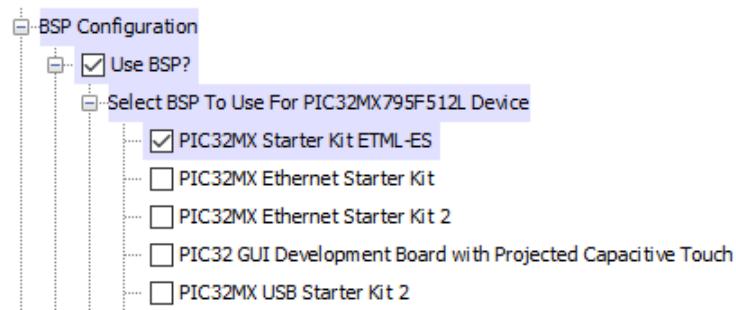


Suite à cela, un « Clean and build » doit se passer sans erreur.

### 10.2.3. MODIFICATION DE LA CONFIGURATION HARMONY

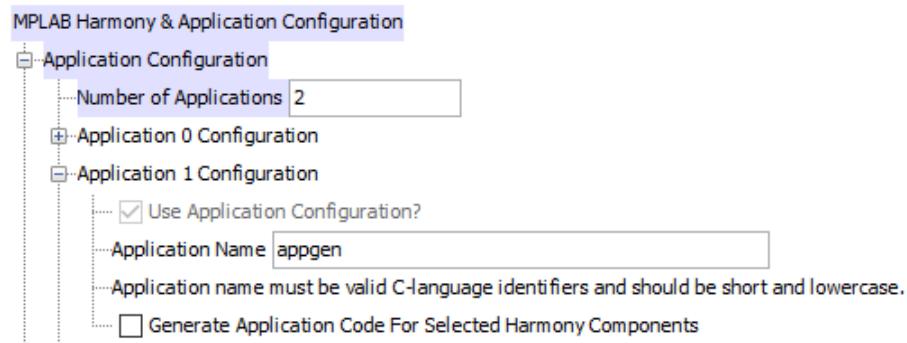
On ouvre le MHC pour adapter la configuration à notre kit (principalement : choix du BSP, sélection et configuration du chip Ethernet physique et ajout d'une application supplémentaire).

Commencer par sélectionner le BSP du starter-kit ES :

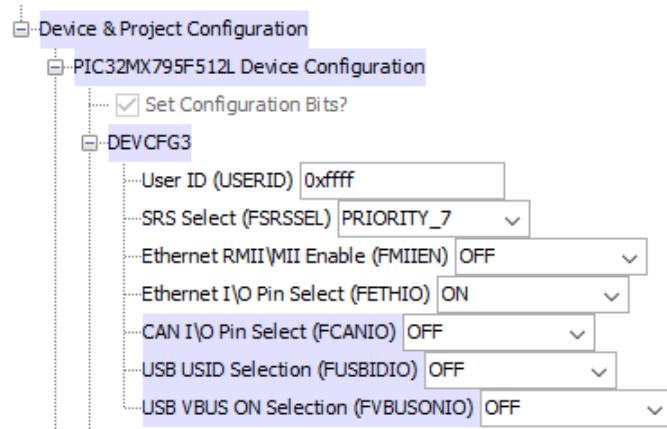


**Ne pas importer la configuration par défaut, sans quoi toute la configuration de l'exemple serait à refaire !**

Ajouter une 2<sup>ème</sup> application que nous appelons « appgen » :

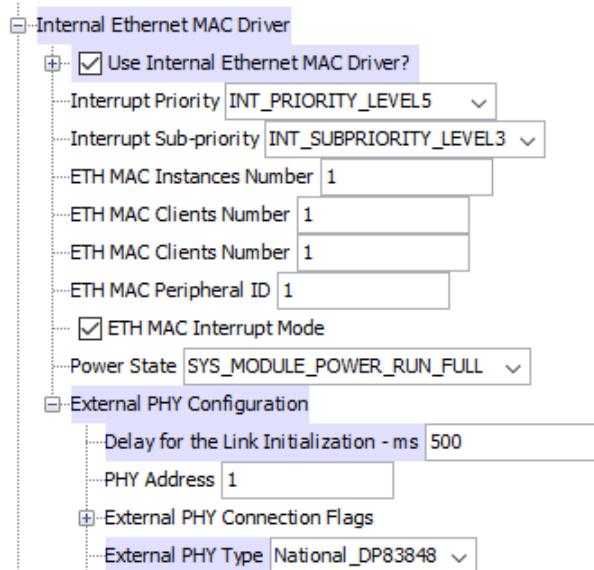


Modifier la configuration dans « Device configuration ». Dans DEVCFG3, régler FETHIO à ON :



On peut également éventuellement désactiver l'oscillateur secondaire dans DEVCFG1 (FOSCEN à OFF).

Comme la configuration utilisée est basée sur un kit utilisant un autre chip PHY, il faut sélectionner celui du starter-kit ES. Sous Drivers > Internal Ethernet MAC Driver > External PHY Configuration > External PHY Type, régler à « National\_DP83848 » :



Générer le code, accepter les modifications.

Suite à cela, un « Clean and build » doit se passer sans erreur.

### **Etat après modifications**

A ce stade, le projet a été porté; il doit être compilable et fonctionnel sur le kit SKES. Moyennant que l'on connaisse l'adresse IP attribuée au kit, on devrait pouvoir tester l'accès au serveur TCP.

#### **10.2.3.1. TEST INITIAL**

##### **10.2.3.1.1. Configuration du BSP**

Ce projet exemple étant destiné à un autre kit, il est possible, sous certaines conditions, qu'il faille reconfigurer les GPIO du PIC32 manuellement.

Cela peut se faire dans le MHC. Dans les fonctions de base nécessaires, on citera :

- Les 4 GPIO nécessaires aux signaux de contrôle du LCD à mettre en sortie,
- Les 8 GPIO nécessaires au LED à mettre en sortie, à l'état d'initialisation désiré.

##### **10.2.3.1.2. Ajout dans appgen.c**

Dans la 2<sup>ème</sup> application, fichier appgen.c, on ajoute le code suivant à l'initialisation :

```
lcd_init();
lcd_b1_on();
printf_lcd("TP5 IpGen 2021");
lcd_gotoxy(1,2);
printf_lcd("SCA"); // nom(s) a adapter
```

Il est nécessaire d'ajouter au préalable : #include "Mc32DriverLcd.h"

##### **10.2.3.1.3. Affichage erreur init stack, modif app.c**

Au niveau de app.c, dans le case APP\_TCPIP\_WAIT\_INIT, on transforme de la manière suivante :

```
case APP_TCPIP_WAIT_INIT:
    tcpipStat = TCPIP_STACK_Status(sysObj.tcpip);
    if(tcpipStat < 0)
    {
        // some error occurred
        SYS_CONSOLE_MESSAGE("APP: TCP/IP stack
                            initialization failed!\r\n");
        //ajout SCA
        lcd_gotoxy(1,4);
        printf_lcd("TCP/IP error !");
        appData.state = APP_TCPIP_ERROR;
```

Il est nécessaire d'ajouter au préalable : #include "Mc32DriverLcd.h"

#### 10.2.3.1.4. Affichage adresse IP

Dans le case APP\_TCPIP\_WAIT\_FOR\_IP, on peut afficher l'adresse IP à chaque changement au niveau de la boucle for :

```

// if the IP address of an interface has changed
// display the new value on the system console
nNets = TCPIP_STACK_NumberOfNetworksGet();

for (i = 0; i < nNets; i++)
{
    netH = TCPIP_STACK_IndexToNet(i);
    if(!TCPIP_STACK_NetIsReady(netH))
    {
        return;      // interface not ready yet!
    }
    ipAddr.Val = TCPIP_STACK_NetAddress(netH);
    if(dwLastIP[i].Val != ipAddr.Val)
    {
        dwLastIP[i].Val = ipAddr.Val;

        SYS_CONSOLE_MESSAGE(TCPIP_STACK_NetNameGet(netH));
        SYS_CONSOLE_MESSAGE(" IP Address: ");
        SYS_CONSOLE_PRINT("%d.%d.%d.%d \r\n", ipAddr.v[0],
                          ipAddr.v[1], ipAddr.v[2], ipAddr.v[3]);
        //ajout SCA : affichage adr. IP
        lcd_gotoxy(1,4);
        printf_lcd("IP:%03d.%03d.%03d.%03d ", ipAddr.v[0],
                  ipAddr.v[1], ipAddr.v[2], ipAddr.v[3]);
    }
    appData.state = APP_TCPIP_OPENING_SERVER;
}
break;

```

#### 10.2.3.1.5. Résultat

Ce premier test permet de vérifier si la base fonctionne et de s'assurer que l'introduction du BSP ne perturbe pas les actions :

- Connecter le kit au réseau d'expérimentation.
- Le message d'annonce, puis l'adresse IP doivent s'afficher.  
Remarque : Les adresses IP 0.0.0.0, ou encore celle statique par défaut 192.168.100.115 peuvent apparaître avant la valeur effective.
- Si l'adresse IP est affichée et semble plausible (voir remarque ci-dessous en cas de problème), ouvrir un terminal TCP en tant que client (par exemple PuTTY en mode « raw ») et se connecter à l'adresse IP du kit sur le port TCP n°9760.

Le programme du PIC32 ouvre un serveur TCP qui écoute sur le port 9760 et reboucle les caractères reçus en majuscules. De plus, le serveur fermera la connexion TCP s'il reçoit le caractère correspondant à l'appui sur 'ESC'.

### Remarque

Si le kit ne semble pas (toujours) obtenir d'adresse IP via DHCP, il est possible que cela provienne d'un timeout dû à un serveur DHCP trop lent. Ce cas de figure a pu être observé sur le réseau d'expérimentation de l'ES.

- Un symptôme peut être que l'adresse IP statique par défaut reste affichée (192.168.100.115), comme si on ne recevait pas de réponse DHCP.
- La solution est d'augmenter le timeout dans le MHC :  
Harmony Framework Configuration → TCP/IP Stack → DHCP Client → DHCP Request Time-out (seconds) → passer de 2s à 5s.  
Avec 5s, une configuration IP devrait être correctement reçue via DHCP et affichée.

### 10.2.3.2. CONCLUSION AU SUJET DU PORTAGE DE L'EXEMPLE

Cette partie a montré comment porter l'exemple Microchip sur le kit SKES. La partie suivante concerne l'intégration d'une application à l'exemple et les interactions avec le serveur TCP.

### 10.2.4. INTEGRATION DU GENERATEUR

Il s'agit maintenant d'intégrer les fichiers du générateur et de faire évoluer l'application. Il est encore nécessaire d'ajouter manuellement les timers 1 et 3.

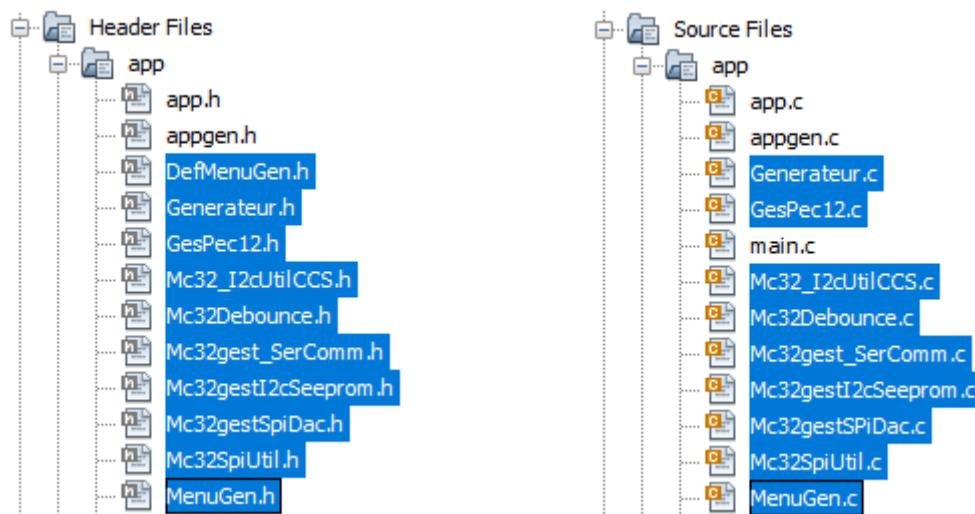
Dans cet exemple, nous reprenons les fichiers issus de l'adaptation à l'USB.

#### 10.2.4.1. MISE EN PLACE DES FICHIERS DU GENERATEUR

Il faut copier presque tous les fichiers source du TP générateur USB dans le répertoire du projet actuel:

<Répertoire Harmony>\v<n>\apps\tcpip\tcpip\_tcp\_server\_TP5\_IpGen  
\firmware\src

#### 10.2.4.1.1. Résultat de l'ajout dans les header et sources



#### 10.2.4.2. AJOUT DES TIMERS

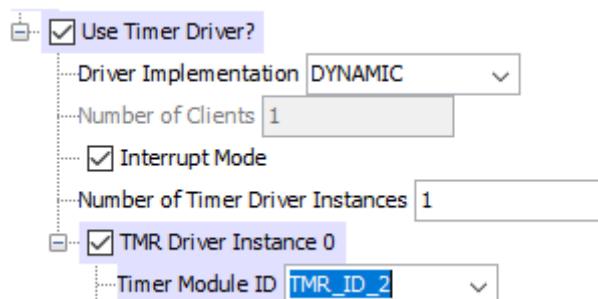
L'exemple de Microchip utilisé `tcpip_tcp_server` inclut un driver de timer dynamique. Ce driver utilise le timer 1.

Pour notre générateur de fonctions, nous utilisons les timers 1 et 3 en mode statique. Il nous faut des timings inférieurs à la milliseconde et aussi précis que possible. Un driver dynamique ne convient pas, et le MHC ne permet pas de mixer les types statique et dynamique.

La configuration de l'exemple ne convient donc pas au générateur de fonctions ; nous allons la modifier.

##### 10.2.4.2.1. Modification du timer utilisé par l'exemple

Dans le MHC, changer le timer dynamique utilisé de `TMR_ID_1` en `TMR_ID_2` afin de libérer le timer 1 (utilisé par notre générateur), puis régénérer le code.



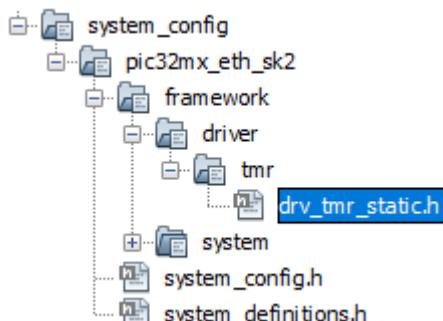
##### 10.2.4.2.2. Ajout des timers 1 et 3

Nous allons ici reprendre les fichiers des drivers statiques des timers 1 et 3 créés dans le projet du générateur USB. Il suffit de copier le répertoire `tmr` de l'ancien projet et de le copier sous :

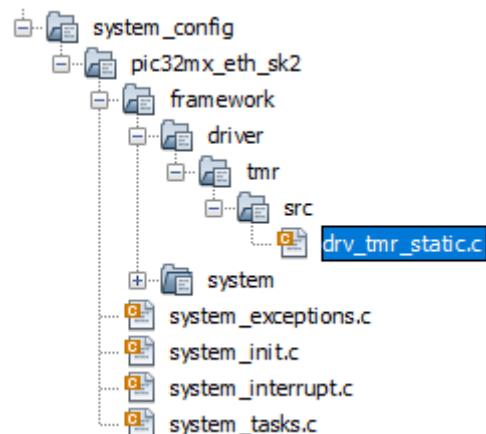
`<Répertoire Harmony>\v<n>\apps\tcpip\ tcpip_tcp_server_TP5_IpGen\firmware\src\system_config\pic32mx_eth_sk2\framework\driver`.

Ajouter les fichiers .h et .c dans le projet en recréant les logical folder :

Dans les header :



Dans les sources :



Il faut encore ajouter manuellement les appels aux fonctions d'initialisation des timers, par exemple dans la fonction SYS\_Initialize ou encore dans APPGEN\_Initialize. La deuxième alternative présente l'avantage de ne pas être écrasée par une régénération du code.

```
// ajout init drivers timers statiques
DRV_TMR0_Initialize();
DRV_TMR1_Initialize();
```

Un #include est nécessaire pour les appels des fonctions du driver de timer :

```
#include "driver/tmr/drive_tmr_static.h" //driver timer static
```

Finalement, copier le code des ISR des timers dans system\_interrupt.c :

```
// timer 1 configure pour interrupt toutes les 1 ms
// anti-rebond et appgen
void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
    _IntHandlerDrvTmrInstance0 (void)

// timer 3 pour la generation de signal
Void __ISR(_TIMER_3_VECTOR, ipl7AUTO)
    _IntHandlerDrvTmrInstance1(void)
```

 Il est possible qu'il faille renommer une ISR pour éviter un doublon ! Le nouveau nom choisi est sans influence sur le reste du programme.

#### 10.2.4.3. MODIFICATION APPGEN

A ce stade, le code de app\_gen peut être en grande partie repris du générateur USB, en adaptant les parties interagissant avec l'USB afin de communiquer en TCP.

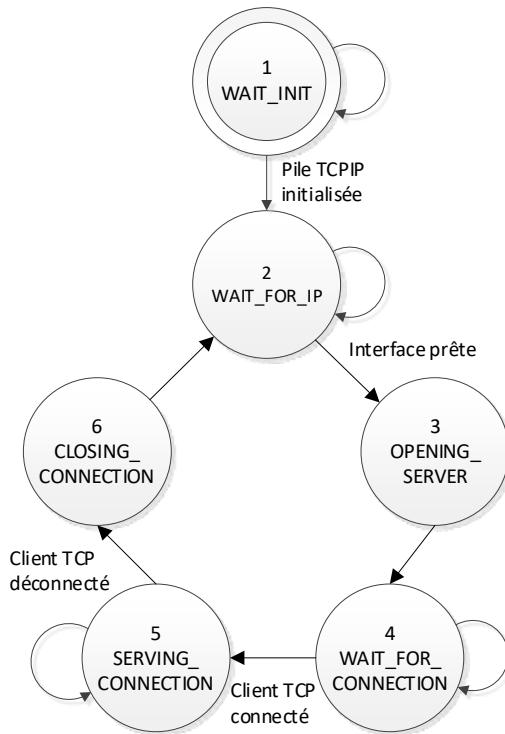
Les principales parties à adapter sont :

- Affichage de la nouvelle adresse IP à chaque changement.  
Le §10.2.3.1.4 donne les principes de détection du changement et d'obtention de la nouvelle adresse IP. Dans le cas d'une application à plusieurs machines d'état, il est toutefois plus propre de centraliser l'affichage (idéalement dans menugen.c).
- Détection de l'état du lien TCP au lieu de la connexion USB.
- Réception et envoi de données via TCP au lieu d'USB.

Ces 2 derniers points sont détaillés ci-dessous.

### 10.2.5. DÉTAILS DU FONCTIONNEMENT DE L’EXEMPLE TCPIP\_TCP\_SERVER

Le projet exemple tcPIP\_tcp\_server peut être décrit selon la machine d'état suivante :



Les états ont des noms explicites. Voici une description des états permettant les interactions ou la communication :

2. Dans cet état, une nouvelle adresse IP obtenue pourra être détectée (voir §10.2.3.1.4).
3. Un serveur TCP (socket) est ouvert. Le programme est ensuite disponible pour qu'un client se connecte.
4. Attente d'une connexion TCP de la part d'un client.
5. Un client est connecté. Les transferts de données peuvent être réalisés dans cet état.
6. Fermeture du serveur TCP.

### 10.2.5.1. DÉTECTION ÉTAT DU LIEN TCP

Les fonctions suivantes peuvent être utiles pour connaître l'état d'une liaison réseau :

- **TCPIP\_STACK\_NetIsLinked()**  
Permet de connaître l'état physique (connecté/déconnecté) de la liaison Ethernet. Ne donne que l'état de la couche TCPIP d'accès au réseau et ne renseigne pas sur la connexion d'un client TCP au serveur.
- **TCPIP\_TCP\_WasReset()**  
Indique si le lien TCP (socket) a été déconnecté depuis le dernier appel.
- **TCPIP\_TCP\_IsConnected()**  
Renvoie l'état du lien TCP.

Les 2 dernières fonctions sont d'une même utilité. La dernière (TCPIP\_TCP\_IsConnected) sera préférée car elle est déjà utilisée dans l'exemple en question.

Toutefois des modifications supplémentaires sont nécessaires afin de pouvoir détecter une fermeture non propre de la connexion (de la part du client) ou encore une interruption du réseau. Il faut forcer le socket TCP serveur à créer un échange de données régulier avec le client, même en cas d'absence de données à transférer (« keepAlive »). Ce mécanisme permet de vérifier que le client est toujours joignable, et donc que le lien TCP est actif.

Cela peut être fait simplement en réglant les propriétés du socket TCP à la suite de son ouverture dans l'état « OPENING\_SERVER » (ajouts en gras) :

```

case APP_TCPIP_OPENING_SERVER:
{
    SYS_CONSOLE_PRINT("Waiting for Client Connection on port:
                      %d\r\n", SERVER_PORT);
    appData.socket = TCPIP_TCP_ServerOpen(IP_ADDRESS_TYPE_IPV4,
                                          SERVER_PORT, 0);
    if (appData.socket == INVALID_SOCKET)
    {
        SYS_CONSOLE_MESSAGE("Couldn't open server socket\r\n");
        break;
    }

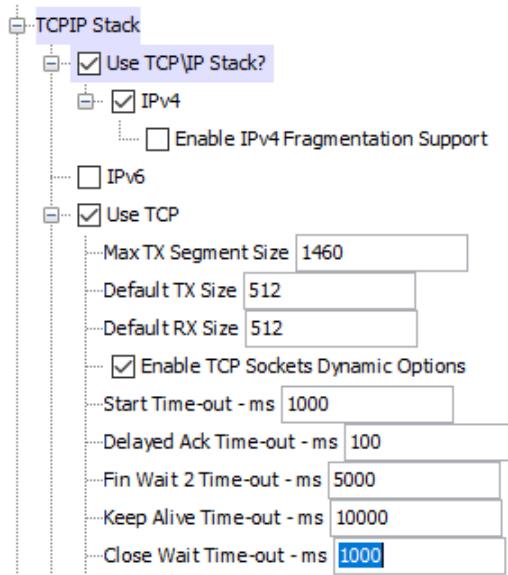
    //SCA set keepalive
    // Nécessaire si on veut que TCPIP_TCP_IsConnected() détecte
    // la déconnexion du câble
    appData.keepAlive.keepAliveEnable = true;
    appData.keepAlive.keepAliveTmo = 1000;
        // [ms] / 0 => valeur par défaut
    appData.keepAlive.keepAliveUnackLim = 2;
        // [nb de tentatives] / 0 => valeur par défaut
    TCPIP_TCP_OptionsSet(appData.socket, TCP_OPTION_KEEP_ALIVE,
                          &(appData.keepAlive));

    appData.state = APP_TCPIP_WAIT_FOR_CONNECTION;
}
break;

```

La déclaration du champ de structure appData.keepAlive est nécessaire au préalable.

Finalement, il faut régler la propriété TCP « Close Wait Time-Out » dans le MHC à une valeur différente de zéro. Par exemple 1000 ms :



Avec le mécanisme et les réglages décrits ci-dessus, la machine d'état de l'exemple sortira automatiquement de l'état « SERVING\_CONNECTION » en cas de déconnexion TCP, quelle qu'en soit la cause (déconnexion physique ou logicielle, propre ou non).

Le lecteur est invité à se reporter à l'aide de Harmony pour davantage d'informations.

### 10.2.5.2. RECEPTION ET ENVOI DE DONNEES TCP

Dans l'état « SERVING\_CONNECTION » les fonctions suivantes peuvent être utilisées pour communiquer via TCP :

#### **TCPIP\_TCP\_GetIsReady(appData.socket)**

- Renvoie le nombre d'octets qui peuvent être lus (qui ont été reçus).
- Paramètre : socket. A laisser tel quel.

#### **TCPIP\_TCP\_ArrayGet(appData.socket, buffer, len)**

- Lit des octets reçus via TCP (les enlève du buffer de réception TCP et les copie dans buffer).
- Paramètre 1 : socket. A laisser tel quel.
- Paramètre 2 : buffer où vont être copiées les données.
- Paramètre 3 : nombre d'octets à lire.
- Retourne le nombre d'octets lus.

#### **TCPIP\_TCP\_PutIsReady(appData.socket)**

- Renvoie le nombre d'octets qui peuvent être envoyés (la place libre dans le buffer).
- Paramètre : socket. A laisser tel quel.

#### **TCPIP\_TCP\_ArrayPut(appData.socket, buffer, len)**

- Envoie des données via un socket TCP. Les données sont copiées dans le buffer d'envoi TCP ; l'envoi à proprement parler sera géré par la pile TCPIP.
- Paramètre 1 : socket. A laisser tel quel.
- Paramètre 2 : buffer où sont stockées les données à envoyer.
- Paramètre 3 : nombre d'octets à envoyer.
- Retourne le nombre d'octets effectivement copiés dans le buffer d'envoi.

### 10.2.5.3. CONCLUSION AU SUJET DE LA MODIFICATION DE L'EXEMPLE

Cette partie a montré comment modifier l'exemple Microchip en lui l'intégrant une autre application sous la forme d'une deuxième machine d'état. La manière de communiquer via TCP a ensuite été abordée.

## 10.3. REALISATION D'UN WEB SERVER

Cette partie a pour objectif de découvrir comment modifier l'exemple Microchip web\_server\_nvm\_mpfs pour aboutir au pilotage du générateur de signal à partir de sa page web. Cet exemple de base présente un serveur web dont les pages sont stockées dans la flash interne du PIC32.

### 10.3.1. UTILISATION D'UN EXEMPLE ET PORTAGE SUR LE KIT

Pour découvrir la réalisation d'un web server, nous allons utiliser l'exemple qui est sous :  
<Répertoire Harmony>\v<n>\apps\tcpip\web\_server\_nvm\_mpfs

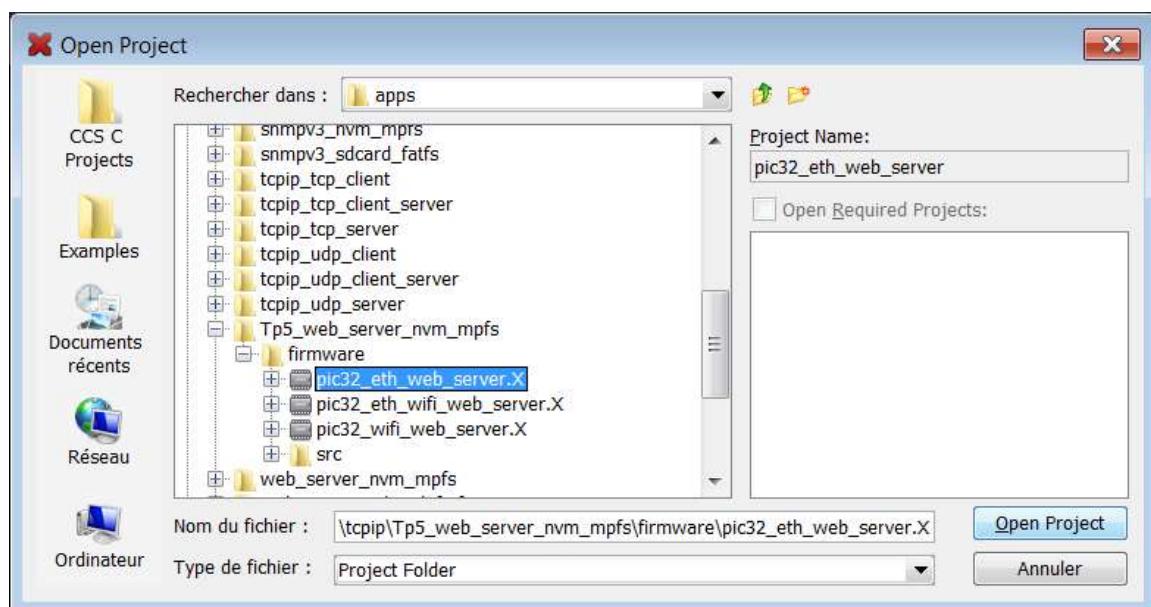
Pour ne pas modifier le contenu de l'exemple, nous copions le répertoire **web\_server\_nvm\_mpfs** au même niveau et nous le renommons **Tp5\_web\_server**.

Le portage sur le kit SKES du projet suivant et son adaptation ont été réalisés avec les logiciels suivants :

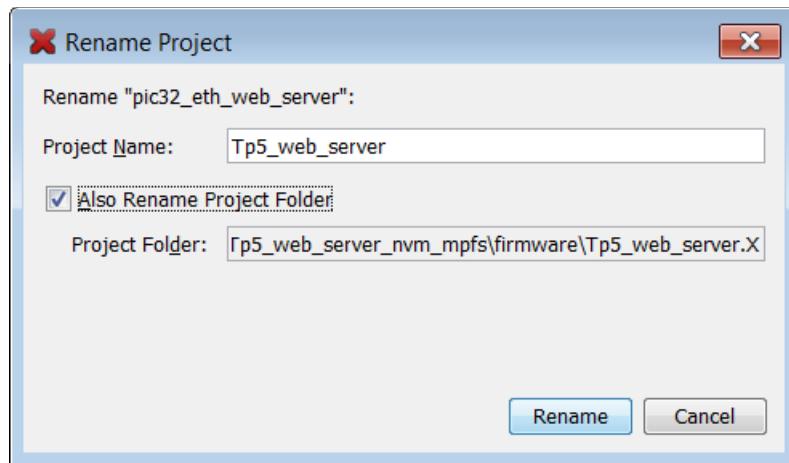
- Harmony v2.05.01
- MPLABX IDE v4.15
- XC32 v2.05

#### 10.3.1.1. OUVERTURE DU PROJET

Le projet contient une triple configuration. Nous choisissons pic32\_eth\_web\_server.X comme ci-dessous :

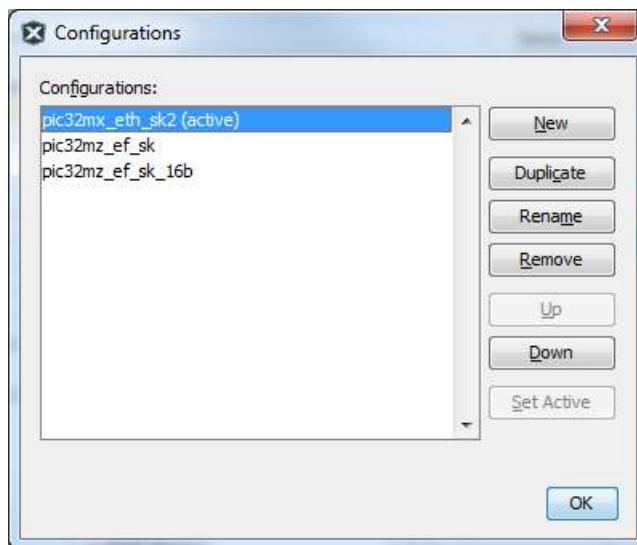


On renomme le projet en Tp5\_web\_server :



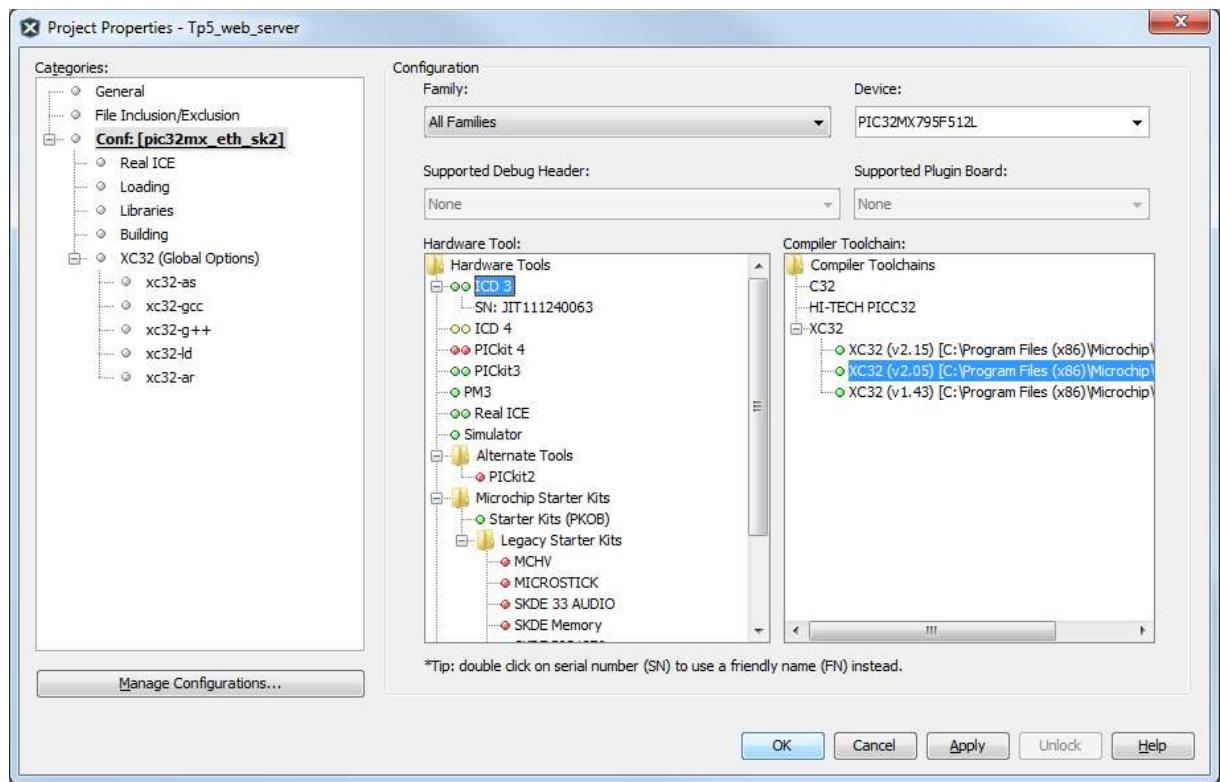
#### 10.3.1.2. REDUCTION DES CONFIGURATIONS

A partir des propriétés du projet, puis [Manage Configurations...](#), on obtient :



Nous ne conservons que la première (pic32mx\_eth\_sk2), c'est la plus proche de la configuration de notre kit.

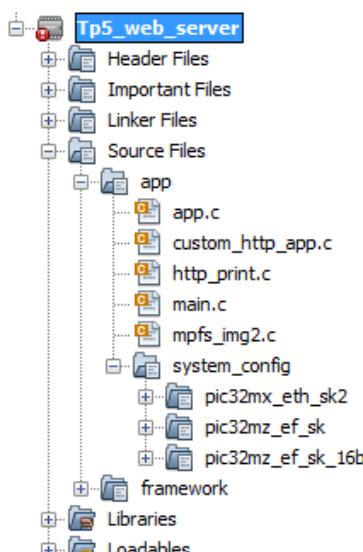
Ce qui nous ramène à une situation plus simple :



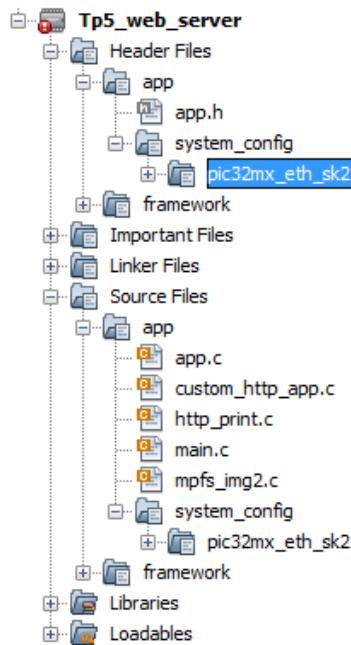
- Au besoin, régler la bonne version du compilateur.
- Ne pas oublier la coche xc32-gcc > preprocessing and messages > additionnal warnings.
- Afin de faciliter le debug, régler le niveau d'optimisation du compilateur à 0.

#### 10.3.1.3. REDUCTION DU PROJET

Dans l'arborescence du projet, sous "header files" et "source files", on retrouve les différentes configurations et les bsp.



En utilisant **Remove From Project**, on supprime les différents répertoires pour ne garder que "pic32mx\_eth\_sk2" à chaque fois. On aboutit à la situation suivante :



#### 10.3.1.3.1. Vérification si compilation OK

Avant d'effectuer d'autres modifications, il faut vérifier si le résultatat du Clean and Build donne un bon résultat.



```

Configuration Loading Error * Tp5_web_server (Clean, Build, ...) *
" C:\Program Files (x86)\Microchip\xc32\v1.40\bin"\\"xc32-bin2hex dist\pic32mx_eth_sk\production/Tp5_web_server.X.production.elf
make[2]: Leaving directory 'C:/microchip/harmony/v1_06/apps/tcpip/Tp5_web_server_nvm_mpfs/firmware/Tp5_web_server.X'
make[1]: Leaving directory 'C:/microchip/harmony/v1_06/apps/tcpip/Tp5_web_server_nvm_mpfs/firmware/Tp5_web_server.X'

BUILD SUCCESSFUL (total time: 25s)
Loading symbols from C:/microchip/harmony/v1_06/apps/tcpip/Tp5_web_server_nvm_mpfs/firmware/Tp5_web_server.X/dist/pic32mx_eth_sk/

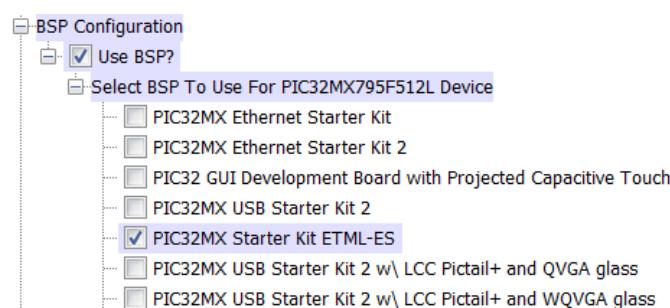
```

#### 10.3.2. MODIFICATION DE LA CONFIGURATION HARMONY

On ouvre le MHC pour adapter la configuration à notre kit (principalement : choix du BSP, sélection du chip Ethernet physique et ajout d'une application supplémentaire).

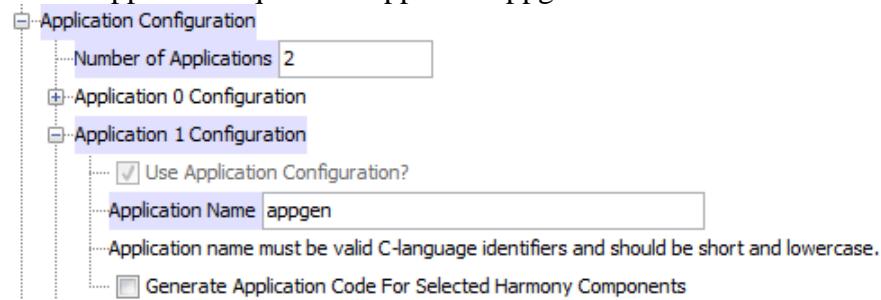
##### 10.3.2.1. SELECTION BSP PIC32MX\_SKES

Sélection de  PIC32MX Starter Kit ETML-ES



### 10.3.2.2. AJOUT APPGEN

Ajout d'une 2<sup>ème</sup> application que nous appelons appgen.

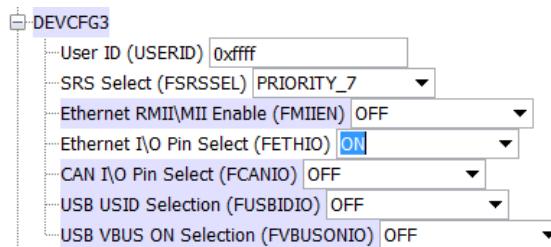


### 10.3.2.3. MODIFICATION DEVICE CONFIGURATION

La configuration ne demande que quelques modifications :

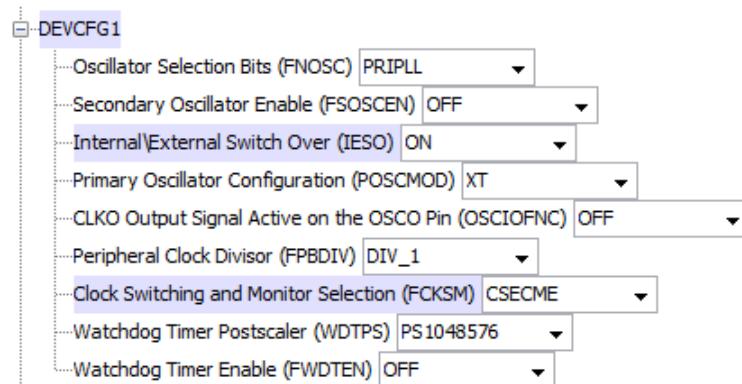
#### 10.3.2.3.1. Modification DEVCFG3

Il faut modifier FETHIO de OFF à ON pour utiliser le port Ethernet par défaut (cette configuration sélectionne le jeu de pins RMII connectées au chip physique externe).



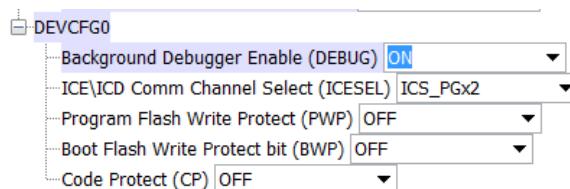
#### 10.3.2.3.2. Modification DEVCFG1

Désactiver l'oscillateur secondaire (FSOSCEN à OFF au lieu de ON).



#### 10.3.2.3.3. Modification DEVCFG0

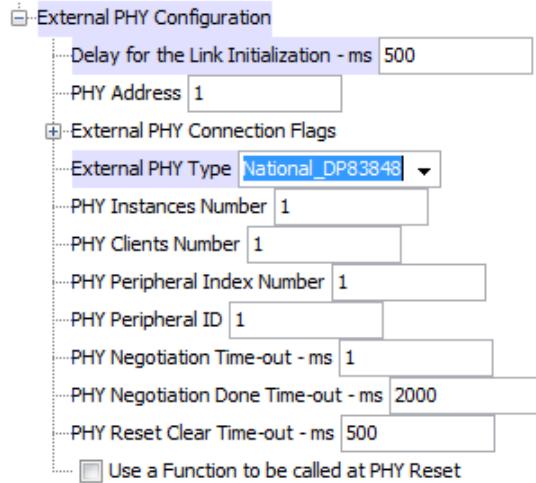
Pour DEBUG ON au lieu de OFF.



#### 10.3.2.4. SELECTION DU CHIP ETHERNET PHYSIQUE

Avec la configuration choisie (la plus proche de notre kit), le projet exemple se base sur l'utilisation d'un chip physique Ethernet qui est différent du nôtre.

Dans Harmony Framework Configuration > Drivers > Internal Ethernet MAC Driver > External PHY Configuration, sélectionner le bon chip (National\_DP83848) :



### 10.3.2.5. TIMERS

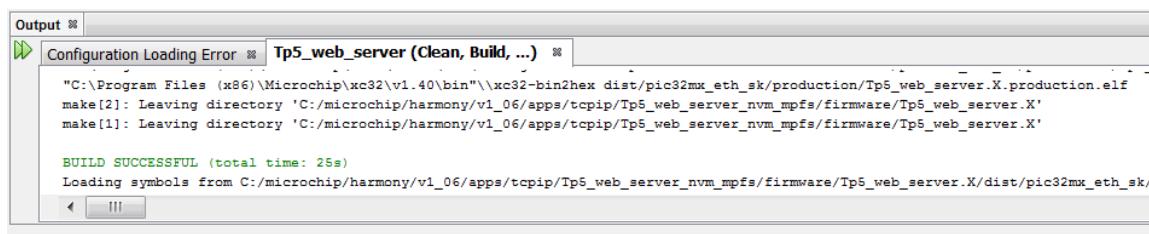
L'exemple de Microchip inclut un driver de timer dynamique. Ce driver utilise déjà le timer 2.

Pour notre générateur de fonctions, nous utilisons les timers 1 et 3 en mode statique. Il nous faut des timings inférieurs à la milliseconde et aussi précis que possible. Un driver dynamique ne convient pas, et il n'est pas possible de mixer les types statique et dynamique.

La configuration de l'exemple ne convient donc pas au générateur de fonctions. Nous conservons la configuration du timer de l'exemple. Les 2 drivers de timers seront ajoutés manuellement au projet par la suite.

### 10.3.2.6. TEST COMPILEMENT

Génération du code, puis test de compilation.



```
Output Configuration Loading Error Tp5_web_server (Clean, Build, ...)

"C:\Program Files (x86)\Microchip\XC32\v1.40\bin"\\"xc32-bin2hex dist\pic32mx_eth_sk\production\Tp5_web_server.X.production.elf
make[2]: Leaving directory 'C:/microchip/harmony/v1_06/apps/tcpip/Tp5_web_server_nvm_mpfs/firmware/Tp5_web_server.X'
make[1]: Leaving directory 'C:/microchip/harmony/v1_06/apps/tcpip/Tp5_web_server_nvm_mpfs/firmware/Tp5_web_server.X'

BUILD SUCCESSFUL (total time: 25s)
Loading symbols from C:/microchip/harmony/v1_06/apps/tcpip/Tp5_web_server_nvm_mpfs/firmware/Tp5_web_server.X/dist/pic32mx_eth_sk/
```

### 10.3.2.7. ETAT APRES MODIFICATIONS

A ce stade, le projet a été porté; il doit être compilable et fonctionnel sur le kit SKES. Moyennant que l'on connaisse l'adresse IP attribuée au kit, on devrait pouvoir tester l'accès à sa page web. L'obtention de l'adresse IP est détaillée ci-après.

Remarquons qu'une grande partie de la flash interne est utilisée par les fichiers du serveur web et d'autres fonctionnalités qui ne sont pas forcément utiles.

### 10.3.2.8. REDUCTION DE LA FLASH NECESSAIRE

Cette étape n'est pas obligatoire. Elle permet de réduire la flash nécessaire au projet, par exemple pour utilisation sur un autre modèle de PIC32. On peut enlever différentes fonctionnalités que l'on n'utilise pas. Dans MHC, enlever les coches :

- Sous Harmony Framework Configuration → System Services :
  - Command
  - console (l'exemple inclut une ligne de commande accessible via USB)
  - debug
- Sous Harmony Framework Configuration → TCPIP Stack :
  - FTP (les fichiers du serveur web sont également accessibles via FTP, mais de manière limitée dans la flash interne.)
  - Telnet Server
  - TCPIP commands (le fait d'enlever cette coche enlève également la coche USB Stack)
- Sous Harmony Framework Configuration:
  - USB Stack

Ces modifications demandent d'enlever du code les parties qui utilisent les fonctionnalités enlevées. Dans app.c, enlever :

- Les appels à SYS\_CONSOLE\_PRINT() et SYS\_CONSOLE\_MESSAGE()
- Les appels à SYS\_CMD\_READY\_TO\_READ()
- Les variables netName et netBiosName (déclaration et affectations)

### 10.3.2.9. TEST INITIAL

Plusieurs modifications peuvent être utiles afin d'utiliser par exemple les LED ou encore l'affichage LCD afin d'y afficher l'adresse IP.

- Configuration du BSP  
→ Se reporter au §10.2.3.1.1
- Ajout dans appgen.c  
→ Se reporter au §10.2.3.1.2 10.2.3.1.1
- Affichage erreur init stack, modif app.c  
→ Se reporter au §10.2.3.1.3

### 10.3.2.9.1. Affichage adresse IP

On peut également afficher l'adresse IP. Il est possible de se reporter au §10.2.3.1.4 qui détaille un fonctionnement basique.

Une version plus élaborée, qui tient compte de la déconnexion de l'interface réseau (câble Ethernet), est détaillée ci-dessous.

Dans le cas APP\_TCPIP\_TRANSACT, on peut afficher l'adresse IP à chaque changement au niveau de la boucle for :

```
// if the IP address of an interface has changed
// display the new value on the system console
nNets = TCPIP_STACK_NumberOfNetworksGet();

for(i = 0; i < nNets; i++)
{
    netH = TCPIP_STACK_IndexToNet(i);
    ipAddr.Val = TCPIP_STACK_NetAddress(netH);

    //ajout SCA pour obtenir nouvel affichage à chaque
    //changement
    netIsLinked[1] = netIsLinked[0];      //décalage états
    //lecture nouvel état
    netIsLinked[0] = TCPIP_STACK_NetIsLinked(netH);

    if(dwLastIP[i].Val != ipAddr.Val || (netIsLinked[1] != netIsLinked[0])) //IP ou état du lien changé ?
    {
        dwLastIP[i].Val = ipAddr.Val;
        if (!netIsLinked[0]) //interface déconnectée ?
        {
            ipAddr.Val = 0;
        }
        //SYS_CONSOLE_PRINT("%s IP Address: %d.%d.%d.%d
                           \r\n",
                           TCPIP_STACK_NetNameGet(netH),
                           ipAddr.v[0], ipAddr.v[1], ipAddr.v[2],
                           ipAddr.v[3]);
        lcd_gotoxy(1,4);
        printf_lcd("IP:%03d.%03d.%03d.%03d ", ipAddr.v[0],
                   ipAddr.v[1], ipAddr.v[2], ipAddr.v[3]);
    }
}
```

Cela nécessite la déclaration suivante préalable :

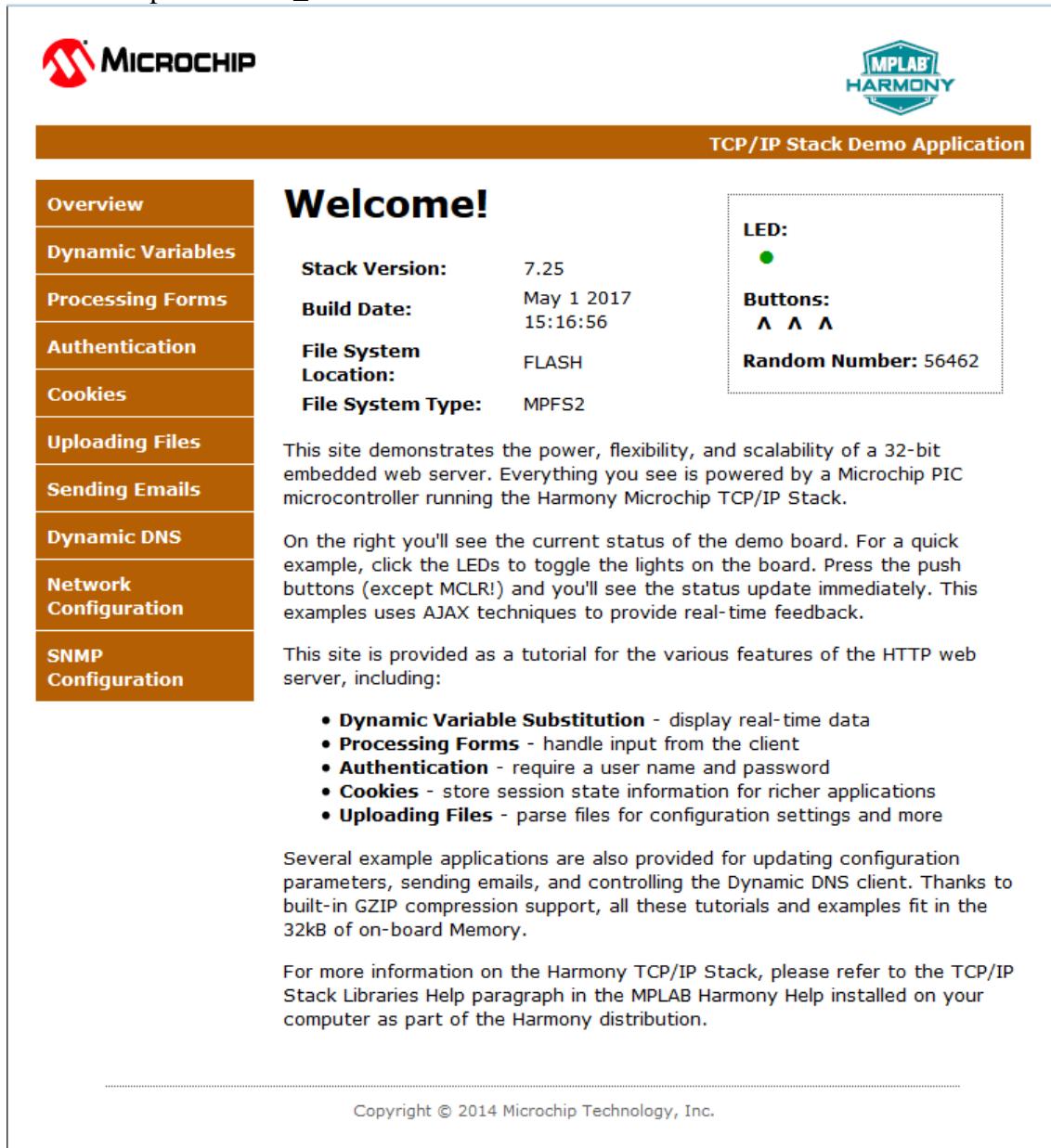
```
//SCA pr détection changement adr. IP
static bool      netIsLinked[2] = {false, false};
```

La variable netIsLinked[0] peut être utilisée pour connaître l'état du réseau.

### 10.3.2.9.2. Résultat

Ce premier test permet de vérifier si la base fonctionne et de s'assurer que l'introduction du BSP ne perturbe pas les actions :

- Connecter le kit au réseau d'expérimentation.
  - Le message d'annonce, puis l'adresse IP doivent s'afficher.
- Remarque : Les adresses IP 0.0.0.0, ou encore celle statique par défaut 192.168.100.115 peuvent apparaître avant la valeur effective.
- Si l'adresse IP est affichée et semble plausible, ouvrir un navigateur à l'adresse « [http://<adresse\\_IP>](http://<adresse_IP>) ». On doit obtenir :



**MICROCHIP**

**MPLAB HARMONY**

**TCP/IP Stack Demo Application**

**Welcome!**

**Overview**

**Dynamic Variables**

**Processing Forms**

**Authentication**

**Cookies**

**Uploading Files**

**Sending Emails**

**Dynamic DNS**

**Network Configuration**

**SNMP Configuration**

**Stack Version:** 7.25  
**Build Date:** May 1 2017  
**File System Location:** FLASH  
**File System Type:** MPFS2

**LED:** ●  
**Buttons:** ▲ ▲ ▲  
**Random Number:** 56462

This site demonstrates the power, flexibility, and scalability of a 32-bit embedded web server. Everything you see is powered by a Microchip PIC microcontroller running the Harmony Microchip TCP/IP Stack.

On the right you'll see the current status of the demo board. For a quick example, click the LEDs to toggle the lights on the board. Press the push buttons (except MCLR!) and you'll see the status update immediately. This examples uses AJAX techniques to provide real-time feedback.

This site is provided as a tutorial for the various features of the HTTP web server, including:

- **Dynamic Variable Substitution** - display real-time data
- **Processing Forms** - handle input from the client
- **Authentication** - require a user name and password
- **Cookies** - store session state information for richer applications
- **Uploading Files** - parse files for configuration settings and more

Several example applications are also provided for updating configuration parameters, sending emails, and controlling the Dynamic DNS client. Thanks to built-in GZIP compression support, all these tutorials and examples fit in the 32kB of on-board Memory.

For more information on the Harmony TCP/IP Stack, please refer to the TCP/IP Stack Libraries Help paragraph in the MPLAB Harmony Help installed on your computer as part of the Harmony distribution.

Copyright © 2014 Microchip Technology, Inc.

Les touches du kit doivent modifier l'affichage sous "Buttons".

### 10.3.2.10. CONCLUSION AU SUJET DU PORTAGE DE L'EXEMPLE

Cette partie a montré comment porter l'exemple Microchip sur le kit SKES. La partie suivante concerne l'intégration d'une application à l'exemple et les interactions avec le webserver.

### 10.3.3. INTEGRATION DU GENERATEUR

Il s'agit maintenant d'intégrer les fichiers du générateur et de faire évoluer l'application. Il est encore nécessaire d'ajouter manuellement les timers 1 et 3.

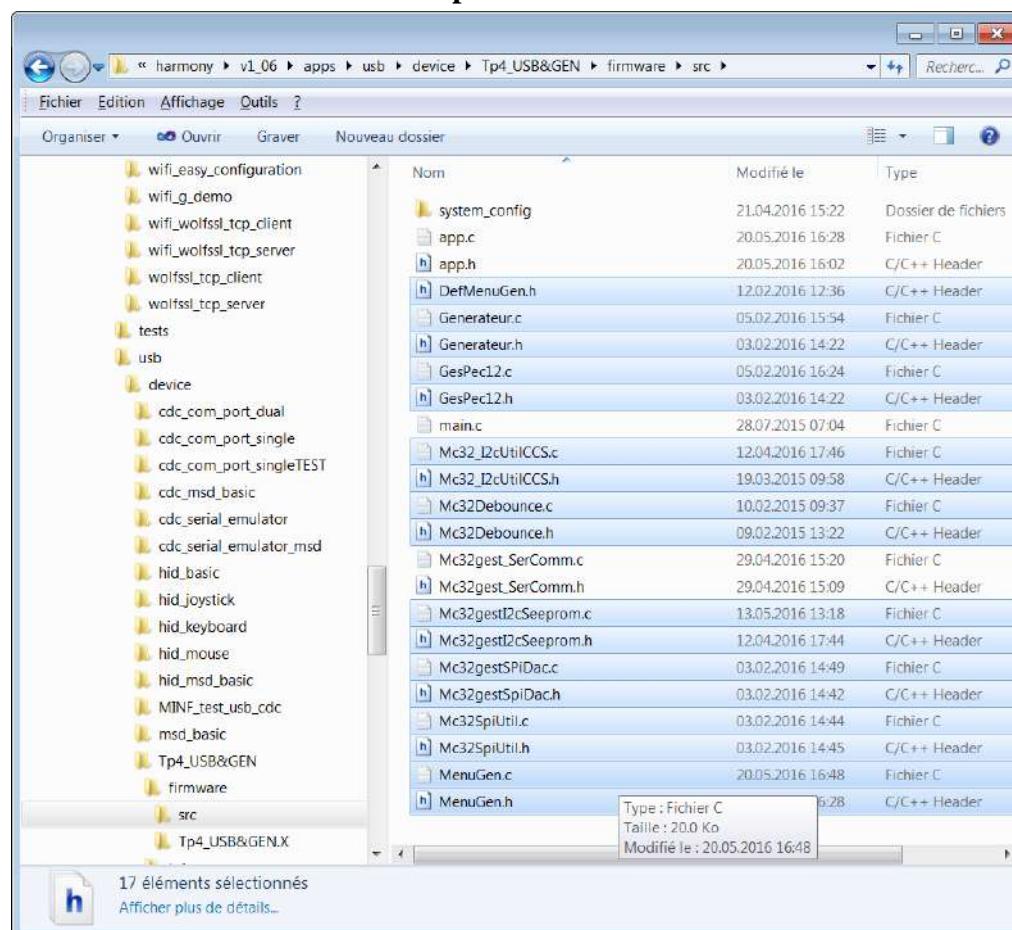
Dans cet exemple, nous reprenons les fichiers issus de l'adaptation à l'USB.

#### 10.3.3.1. MISE EN PLACE DES FICHIERS DU GENERATEUR

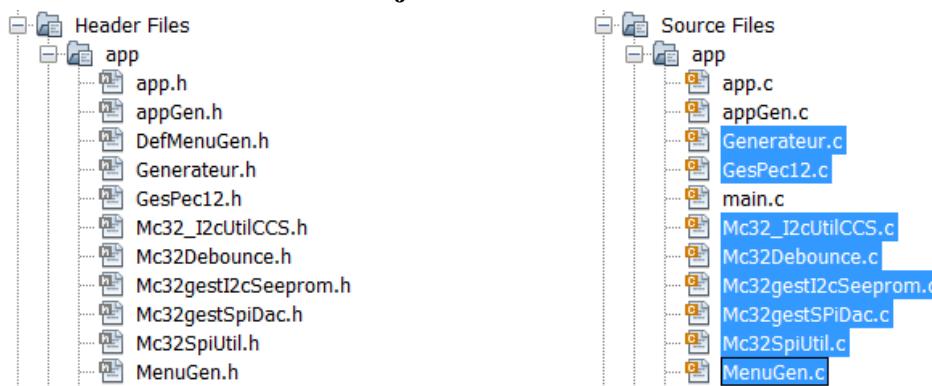
Il faut copier presque tous les fichiers source du TP générateur USB dans le répertoire du projet actuel :

<Répertoire Harmony>\v<n>\apps\tcpip\Tp5\_web\_server\_nvm\_mpfs\firmware\src

##### 10.3.3.1.1. Liste des fichiers à copier



##### 10.3.3.1.2. Résultat de l'ajout dans les header et sources



### 10.3.3.2. ADAPTATION DE SYSTEM\_INIT.C

L'exemple fourni n'a pas de driver statique pour les timers.

#### 10.3.3.2.1. Récupération des fonctions driver timer static

Nous allons ici copier les fichiers des drivers statiques des timers créés dans un autre projet.

Il suffit de copier le répertoire **tmr** depuis :

<Répertoire Harmony>\v<n>\apps\usb\device\Tp4\_UsbGen\firmware\src\

system\_config\pic32mx\_usb\_sk2\_int\_sta\framework\driver

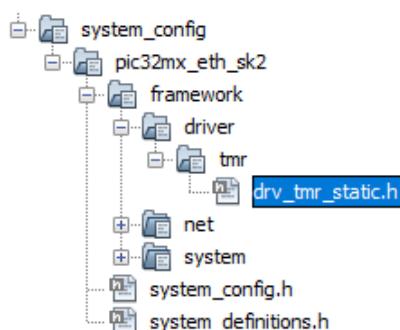
sous :

<Répertoire Harmony>\v<n>\apps\tcpip\Tp5\_web\_server\_nvm\_mpfs\firmware\src\

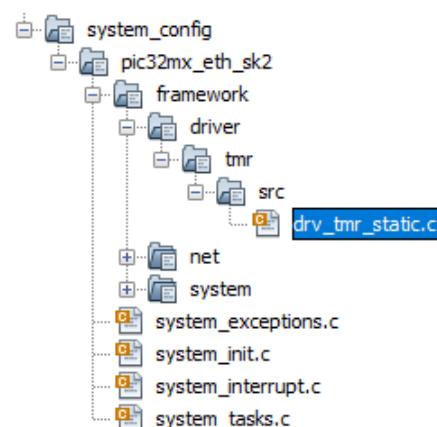
system\_config\pic32mx\_eth\_sk2\framework\driver qui est vide.

Ajouter les fichiers .h et .c .dans le projet en recréant les logical folder :

Dans les header :



Dans les sources :



#### 10.3.3.2.2. Ajout des appels init des drivers statiques

Il faut ajouter les 2 appels des init. des driver timer static dans la fonction **SYS\_Initialize**.

```

/* Board Support Package Initialization */
BSP_Initialize();

/* Initialize Drivers */
// CHR ajout init drivers timers statiques
DRV_TMR0_Initialize();
DRV_TMR1_Initialize();

sysObj.drvTmr0 = DRV_TMR_Initialize(DRV_TMR_INDEX_0,
                                     (SYS_MODULE_INIT *)&drvTmr0InitData);

```

### 10.3.3.2.3. Ajout include driver static dans system\_definitions.h

Il est nécessaire d'ajouter le #include suivant dans le fichier **system\_definitions.h** pour pouvoir compiler system\_init.c.

```
#include "app.h"
#include "appgen.h"
#include "driver/tmr/drv_tmr_static.h" //driver timer static
```

### 10.3.3.3. ADAPTATION DE SYSTEM\_INTERRUPT.C

Il s'agit d'ajouter dans le fichier system\_interrupt.c les réponses aux interruptions des timers 1 & 3 que l'on récupère du projet du Tp4\_UsbGen.

Il faut ouvrir les 2 fichiers dans MPLAB X et effectuer la copie.

⚠ Il faut modifier le nom d'une ISR pour éviter un doublon !

```
// Int Timer1 cycle 1ms pour antirebond
// Activation application tous les 10 cycles
void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
    _IntHandlerDrvTmrInstance0(void)
```

A renommer en :

```
void __ISR(_TIMER_1_VECTOR, ipl3AUTO)
    _IntHandlerTimer1(void)
```

### 10.3.3.3.1. Ajout dans appgen.h

Pour arriver à compiler le fichier system\_interrupt.c, il est nécessaire d'ajouter des #include dans appgen.h, ainsi que des définitions.

#### 10.3.3.3.1.1. Ajouts #include

```
// Spécifique menu et générateur
#include "DefMenuGen.h"
#include "Mc32Debounce.h"
#include "GesPec12.h"
#include "Générateur.h"
```

#### 10.3.3.4. ADAPTATION DE APPGEN.C

##### 10.3.3.4.1. Partie APPGEN\_STATE\_INIT

Cette partie contient l'initialisation des différents éléments du générateur. Elle est reprise telle quelle du générateur projet du générateur USB :

```
case APPGEN_STATE_INIT:  
{  
    //init. lcd  
    lcd_init();  
    lcd_bl_on();  
  
    SPI_InitLTC2604(); // Init SPI DAC  
  
    I2C_InitMCP79411(); //init. comme I2C avec EEPROM  
                        // externe I2C  
  
    Pec12Init(); // Initialisation du PEC12  
  
    // Initialisation du menu et rapatrie valeurs signal de  
    // l'EEPROM externe  
    MENU_Initialize(&LocalParamGen);  
  
    //par défaut (tant que rien reçu), signal USB = signal  
    // local  
    RemoteParamGen = LocalParamGen;  
  
    // Initialisation du generateur  
    GENSIG_Initialize(&LocalParamGen);  
  
    // Active les timers  
    DRV_TMR0_Start();  
    DRV_TMR1_Start();  
  
    appgenData.state = APPGEN_STATE_WAIT;  
  
    break;  
}
```

#### 10.3.3.4.2. Partie APPGEN\_STATE\_SERVICE\_TASKS

Dans cette partie, on enlève tout ce qui touche à la gestion de l'USB. On se retrouve alors avec une gestion simple du menu :

```
case APPGEN_STATE_SERVICE_TASKS:  
    //affichage menu  
    if (MENU_Execute(&LocalParamGen, !APP_NetIsLinked())  
    {  
        //mise à jour signal si le menu le demande  
        //met à jour fréquence  
        GENSIG_UpdatePeriode(&LocalParamGen);  
        //met à jour échantillons  
        GENSIG_UpdateSignal(&LocalParamGen);  
    }  
    appgenData.state = APPGEN_STATE_WAIT;  
    break;
```

Par rapport au fonctionnement en USB, il y a eu une simplification et on n'utilise plus qu'un seul jeu de paramètres.

La fonction APP\_NetIsLinked() a dû être créée en remplacement de APP\_USBIsConnected() qui était valable pour le générateur de fonctions USB. Cette nouvelle fonction indique l'état de la connexion ethernet pour affichage du menu local ou remote. Voir § 10.2.3.1.4 "Affichage adresse IP" pour lecture de l'état du réseau.

#### 10.3.3.5. TEST DU GENERATEUR

A ce stade, la compilation doit être OK.

Sans le câble réseau, on obtient le réglage du générateur et son action.

Lorsque le câble réseau est branché, on obtient l'affichage avec le # et les actions sur le Pec12 sont sans effet.

La page web n'a pas changé et n'a pas d'interaction avec le générateur de fonctions.

### 10.3.4. PAGE WEB PERSONNALISEE

Ce qui suit reprend les explications issues du webinaire de Microchip en 3 parties :

1. "TCP/IP Networking: Web-Based Status Monitoring"  
document Microchip en543032
2. "TCP/IP Networking, Part 2: Web-Based Control"  
document Microchip en543035
3. "TCP/IP Networking, Part 3: Advanced Web-Based Control"  
document Microchip en543038

Cette documentation est disponible sous :

...\\Maitres-Eleves\\SLO\\Modules\\SL229\_MINF\\TP\\CoursLabo

#### 10.3.4.1. COPIE DU REPERTOIRE WEB\_PAGES

Il faut tout d'abord supprimer le répertoire web\_pages sous  
<Répertoire Harmony>\\v<n>\\apps\\tcpip\\Tp5\_web\_server\_nvm\_mpfs\\firmware\\src

Et le remplacer par celui qui est fourni sous :

...\\Maitres-Eleves\\SLO\\Modules\\SL229\_MINF\\TP\\TP5\_WebGen.

#### 10.3.4.2. GENERATION DES FICHIERS

Il faut exécuter l'utilitaire "Microchip MPFS Generator" dont l'exécutable est le fichier **mpfs2.jar** qui se trouve sous :

<Répertoire Harmony>\\v<n>\\utilities\\mpfs\_generator

Cet utilitaire sert à analyser le code HTML de notre page web et à y repérer les variables à afficher. Ces variables doivent être présentées entre tildes (~').

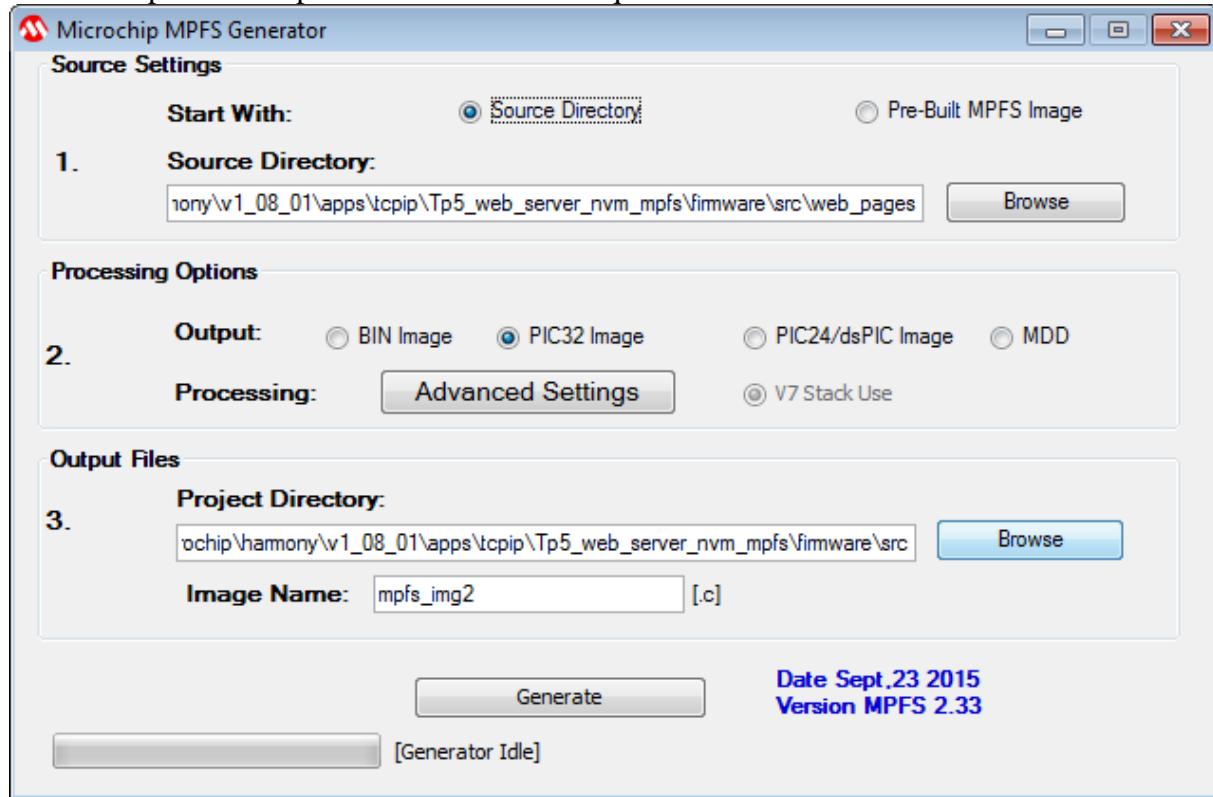
Exemple d'extrait de code HTML :

Valeur de la variable : ~variable~

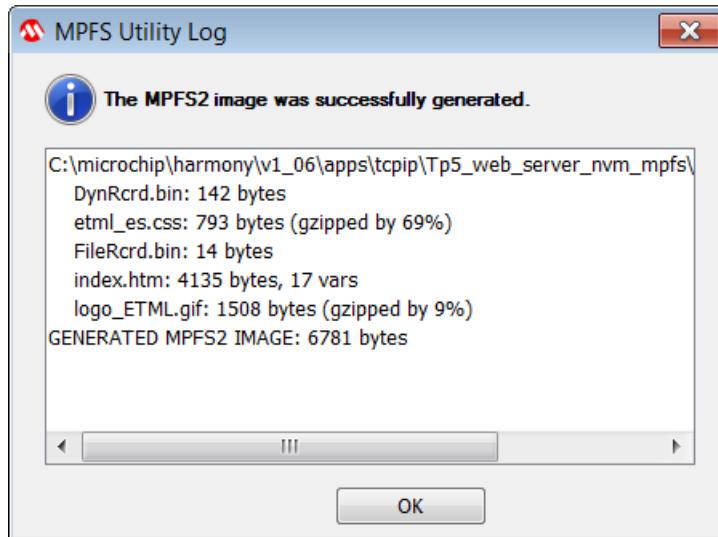
Régler les éléments suivants :

1. Chemin d'accès vers les fichiers web du projet
2. Sélectionner PIC32 image

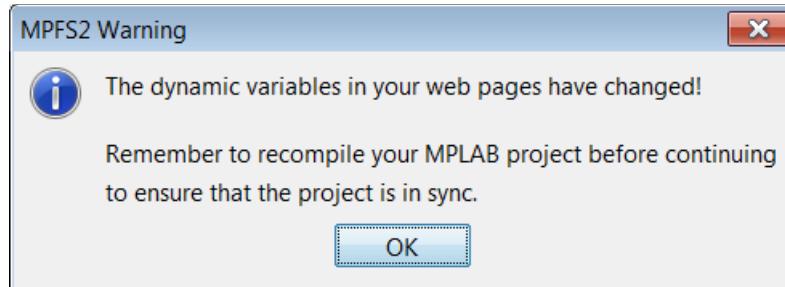
Les autres paramètres peuvent être laissés tels quels :



Et finalement Generate ! On obtient :



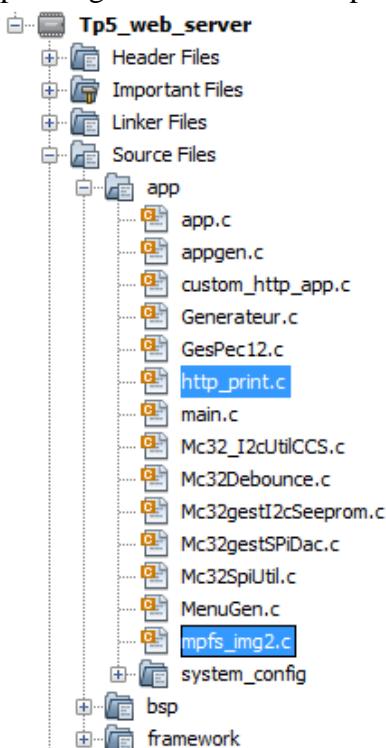
Puis :



Le message signale que des variables dynamiques ont changé. Il y a donc des adaptations de code à effectuer (ci-dessous).

#### 10.3.4.3. FICHIERS MODIFIES

Les fichiers http\_print.c et mpfs\_img.c ont été modifiés par l'utilitaire :



⚠ Ils ne doivent pas être modifiés manuellement !

##### 10.3.4.3.1. Contenu mpfs\_img.c

Ce fichier contient, sous la forme d'un tableau constant, le système de fichiers pour la flash interne du microcontrôleur incluant les fichiers web du serveur.

##### 10.3.4.3.2. Contenu http\_print.c

Pour chaque variable à afficher présente dans le code HTML de la page web, un appel à une fonction d'affichage correspondante a été généré :

```
#include "tcpip/tcpip.h"

void TCPIP_HTTPP_Print(HTTP_CONN_HANDLE connHandle,uint32_t callbackID);
void TCPIP_HTTPP_Print_shapeSelected(HTTP_CONN_HANDLE connHandle,uint16_t);
void TCPIP_HTTPP_Print_freq(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTPP_Print_ampl(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTPP_Print_offset(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTPP_Print_dutyCycle(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTPP_Print_version(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTPP_Print_builddate(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTPP_Print_drive(HTTP_CONN_HANDLE connHandle);
void TCPIP_HTTPP_Print_fstype(HTTP_CONN_HANDLE connHandle);
```

Remarque : cela correspond aux prototypes des fonctions qu'il faut implémenter dans le fichier custom\_http\_app.c.

#### 10.3.4.4. ADAPTATIONS DU FICHIER CUSTOM\_HTTP\_APP.C

Après un clean and build, on obtient les erreurs suivantes :

```
"C:\Program Files (x86)\Microchip\xc32\v1.40\bin\xc32-gcc.exe" -mprocessor=32MX795FS12L -o dist/pic32mx_eth_sk/production/Tp5_web_server.X.production
build/pic32mx_eth_sk/production/_ext/1360937237/http_print.o: In function `TCPIP_HTTPP_Print':
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server/nvm_mpfs/firmware/src/http/print.c:60: undefined reference to `TCPIP_HTTPP_Print_shapeSelected'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server/nvm_mpfs/firmware/src/http/print.c:63: undefined reference to `TCPIP_HTTPP_Print_shapeSelected'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server/nvm_mpfs/firmware/src/http/print.c:66: undefined reference to `TCPIP_HTTPP_Print_shapeSelected'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server/nvm_mpfs/firmware/src/http/print.c:69: undefined reference to `TCPIP_HTTPP_Print_shapeSelected'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server/nvm_mpfs/firmware/src/http/print.c:72: undefined reference to `TCPIP_HTTPP_Print_shapeSelected'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server/nvm_mpfs/firmware/src/http/print.c:75: undefined reference to `TCPIP_HTTPP_Print_freq'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server/nvm_mpfs/firmware/src/http/print.c:78: undefined reference to `TCPIP_HTTPP_Print_ampl'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server/nvm_mpfs/firmware/src/http/print.c:81: undefined reference to `TCPIP_HTTPP_Print_offset'
c:/microchip/harmony/v1_06/apps/tcpip/tp5_web_server/nvm_mpfs/firmware/src/http/print.c:84: undefined reference to `TCPIP_HTTPP_Print_dutyCycle'
collect2.exe: error: ld returned 255 exit status
make[2]: *** [dist/pic32mx_eth_sk/production/Tp5_web_server.X.production.hex] Error 255
```

Il est nécessaire d'implémenter les fonctions dans le fichier custom\_http\_app.c

##### 10.3.4.4.1. Ajout des fonctions dans custom\_http\_app.c

Il y a 4 fonctions à ajouter :

- TCPIP\_HTTPP\_Print\_shapeSelected()
- TCPIP\_HTTPP\_Print\_freq()
- TCPIP\_HTTPP\_Print\_ampl()
- TCPIP\_HTTPP\_Print\_offset()

Cela correspond aux nouvelles 4 variables de la page web pour lesquelles aucune fonction d'affichage n'existe encore. Il suffit de copier les prototypes et d'ajouter {}.

 Il faut encore corriger la fonction TCPIP\_HTTPP\_Print\_shapeSelected.

```
//-----
// ajout SCA : fonctions pour affichage variables dynamiques TP5 WebGen

// nécessaire de nommer le 2ème paramètre
void TCPIP_HTTPP_Print_shapeSelected(HTTP_CONN_HANDLE connHandle,uint16_t shapeNb)
{
}

void TCPIP_HTTPP_Print_freq(HTTP_CONN_HANDLE connHandle)
{
}

void TCPIP_HTTPP_Print_ampl(HTTP_CONN_HANDLE connHandle)
{
}

void TCPIP_HTTPP_Print_offset(HTTP_CONN_HANDLE connHandle)
{
}

#endif // #if defined(TCPIP_STACK_USE_HTTP_SERVER)
```

¶ Ce sont les fonctions pour afficher les variables du PIC sur la page Web. Nous les implémenterons par la suite.

#### 10.3.4.4.2. Observation de la page Web

A ce stade, une compilation ne doit pas générer d'erreur. On peut tester le programme et l'accès à la page web.

Malgré les fonctions vides, on obtient la page spécifique du générateur. On peut modifier les paramètres, mais cela n'a pas d'effet ; il manque les mécanismes de transmission de valeurs.

#### 10.3.4.4.3. Affichage des variables du PIC sur la page web

Selon le § 10.3.4.2 "Génération des fichiers" :

- Les variables que l'on désire afficher sur la page web ont été placées entre tildes dans le code html.
- Sur la base des variables trouvées, l'utilitaire Microchip MPFS Generator a généré :
  - Un fichier mpfs\_img2.c contenant les fichiers web sous la forme d'une grande constante C.
  - Un prototype de fonction pour l'affichage de chaque variable dans http\_print.c.

Il est finalement nécessaire d'implémenter le retour des valeurs des variables. Les fonctions sont à implémenter dans custom\_http\_app.c.

##### 10.3.4.4.3.1. Fonction d'affichage d'une variable

En s'inspirant des fonctions TCPIP\_HTTP\_Print\_...() déjà présentes, on peut facilement créer sur le même modèles celles qui permettront d'afficher nos variables. Par exemple, pour l'affichage de l'offset, cela donne :

```
void TCPIP_HTTP_Print_Offset(HTTP_CONN_HANDLE connHandle)
{
    char str[6];
    TCP_SOCKET sktHTTP =
        TCPIP_HTTP_CurrentConnectionSocketGet(connHandle);

    //conversion entier->ascii
    itoa(str, LocalParamGen.Offset, 10);

    //envoi chaîne
    TCPIP_TCP_StringPut(sktHTTP, (uint8_t*)str);
}
```

Remarque :

L'utilitaire Microchip MPFS Generator génère les prototypes de fonctions nécessaires dans http\_ptrint.c. Il ne modifie pas custom\_http\_app.c. Ce dernier fichier peut donc contenir des fonctionnalités qui étaient présentes dans l'exemple de page web de Microchip, mais qui ne sont plus nécessaires dans notre cas (par exemple : fonctions TCPIP\_HTTP\_Print\_...() d'affichages de variables qui n'existent pas sur notre page).

#### 10.3.4.4.4. Réception des variables de la page web par le PIC

##### 10.3.4.4.4.1. Fonction GetExecute dans custom\_http\_app.c

La page web fournie est conçue pour transmettre ses valeurs au PIC à travers un mécanisme http de type "GET". Le principe est d'accéder à une page et de transmettre des paramètres dans l'adresse de la page accédée (URL). C'est une méthode simple. Son désavantage est la limitation à environ 100 octets transmis, mais cela est largement suffisant pour nous.

La fonction **GetExecute** contient le traitement correspondant à la page demo de Microchip :

```
HTTP_IO_RESULT TCPIP_HTTP_GetExecute(
                                    HTTP_CONN_HANDLE connHandle)
{
    const uint8_t *ptr;
    uint8_t filename[20];
    uint8_t* httpDataBuff;

    // Load the file name
    // Make sure uint8_t filename[] above is large enough
    // for your longest name
    SYS_FS_FileNameGet(TCPIP_HTTP_CurrentConnectionFileGet
                        (connHandle), filename, 20);

    httpDataBuff
        = TCPIP_HTTP_CurrentConnectionDataBufferGet
            (connHandle);

    // If its the forms.htm page
    if(!memcmp(filename, "forms.htm", 9))
    {
        // Seek out each of the four LED strings, and if it
        // exists set the LED states
        ptr = TCPIP_HTTP_ArgGet(httpDataBuff,
                               (const uint8_t *)"led2");
        if(ptr)
            BSP_LEDStateSet(APP_TCPIP_LED_3, (*ptr == '1'));
        //LED2_IO = (*ptr == '1');

        ptr = TCPIP_HTTP_ArgGet(httpDataBuff,
                               (const uint8_t *)"led1");
        if(ptr)
            BSP_LEDStateSet(APP_TCPIP_LED_2, (*ptr == '1'));
        //LED1_IO = (*ptr == '1');
    }

    else if(!memcmp(filename, "cookies.htm", 11))
    {
        // This is very simple. The names and values we
        // want are already in
        // the data array. We just set the hasArgs value
        // to indicate how many
```

```

// name/value pairs we want stored as cookies.
// To add the second cookie, just increment this
// value.
// remember to also add a dynamic variable callback
// to control the printout.
TCPIP_HTTP_CurrentConnectionHasArgsSet(connHandle,
                                         0x01);
}
// If it's the LED updater file
else if(!memcmp(filename, "leds.cgi", 8))
{
    // Determine which LED to toggle
    ptr = TCPIP_HTTP_ArgGet(httpDataBuff,
                           (const uint8_t *)"led");
    // Toggle the specified LED
    switch(*ptr) {
        case '0':
            BSP_LEDToggle(APP_TCPIP_LED_1);
            //LED0_IO ^= 1;
            break;
        case '1':
            BSP_LEDToggle(APP_TCPIP_LED_2);
            //LED1_IO ^= 1;
            break;
        case '2':
            BSP_LEDToggle(APP_TCPIP_LED_3);
            //LED2_IO ^= 1;
            break;
    }
}
return HTTP_IO_DONE;
}

```

#### 10.3.4.4.4.2. Adaptation de GetExecute dans custom\_http\_app.c

Nous devons adapter le code à la chaîne reçue (l'URL) :



Pour cela, nous reprenons le principe utilisé dans le code de la demo. Il s'agit de :

- Vérifier que l'accès est fait sur la bonne page web (index.htm).
- Parser l'URL à l'aide de la fonction TCPIP\_HTTP\_ArgGet() pour trouver les valeurs des paramètres.

Le code de la fonction nous donne :

```

HTTP_IO_RESULT TCPIP_HTTP_GetExecute (
                                         HTTP_CONN_HANDLE connHandle)
{
    const uint8_t *ptr;
    uint8_t filename[20];

```

```

        uint8_t *httpDataBuff;
        int32_t tmp;

        // Load the file name.
        // Make sure uint8_t filename[] above is large enough
        // for your longest name.

        SYS_FS_FileNameGet(TCPIP_HTTP_CurrentConnectionFileGet(
            connHandle), filename, 20);

        httpDataBuff
            = TCPIP_HTTP_CurrentConnectionDataBufferGet
                (connHandle);
        //accès à la page index.htm ?
        if(!memcmp(filename, "index.htm", 9))
        {
            ptr = TCPIP_HTTP_ArgGet
                (httpDataBuff, (uint8_t*)"newShape");

            //récupère forme
            tmp = atoi((char*)ptr);
            LocalParamGen.Forme = tmp -1;

            //récupère fréq.
            ptr = TCPIP_HTTP_ArgGet
                (httpDataBuff, (uint8_t*)"newFreq");
            tmp = atoi((char*)ptr);
            LocalParamGen.Frequence = tmp;

            //récupère ampl.
            ptr = TCPIP_HTTP_ArgGet
                (httpDataBuff, (uint8_t*)"newAmpl");
            tmp = atoi((char*)ptr);
            LocalParamGen.Amplitude = tmp;

            //récupère offset
            ptr = TCPIP_HTTP_ArgGet
                (httpDataBuff, (uint8_t*)"newOffset");
            tmp = atoi((char*)ptr);
            LocalParamGen.Offset = tmp;

            //signale nouvelles valeurs reçues
            APPGEN_NewValuesReceived();
        }
        return HTTP_IO_DONE;
    }
}

```

Suivant le cas de figure de page web, il pourrait être nécessaire de tester la validité des valeurs reçues ou de les traiter (par exemple : arrondi, limitation, etc).

Ne pas oublier gérer la mise à jour de l'affichage et du signal en cas de réception de nouvelles valeurs.

#### 10.3.4.5. OBSERVATION DE LA PAGE WEB

On obtient l'affichage des valeurs avec quittance et action sur le générateur.



#### 10.3.4.6. TRAITEMENT EN MODE POST

La méthode http GET fait passer directement les données à transmettre dans l'URL, et elle est limitée à environ 100 octets.

Il existe une méthode POST qui palie à ces désavantages. Les données sont alors transmises dans le corps de la requête HTTP et la taille à transmettre est illimitée. Elle peut s'avérer nécessaire, par exemple pour transmettre un fichier d'un navigateur à un serveur, mais est plus compliquée à traiter que le GET..

Cette méthode ne sera pas décrite ici, le lecteur pouvant se référer à la partie 3 des webinaires de Microchip (voir sources au § 10.3.4), ainsi qu'au programme exemple web\_server\_nvm\_mpfs, utilisé dans ce document, qui inclut un exemple de POST.

#### 10.3.4.7. RAFRAICHISSEMENT AUTOMATIQUE DE LA PAGE WEB

Il peut également être nécessaire de rafraîchir la page web automatiquement, sans que l'utilisateur ne doive la recharger. L'exemple de Microchip inclut à cet effet un mécanisme AJAX, dont le principe est le suivant :

1. Le navigateur charge la page web à afficher
2. Cette page inclut du code javascript qui va périodiquement refaire une requête d'un fichier xml au serveur. Ce fichier contient uniquement les valeurs des variables qui doivent être remises à jour sur la page.
3. Le code javascript modifie l'affichage des variables (sans avoir dû retransmettre la page en entier).

Le lecteur est invité à se reporter à l'exemple web\_server\_nvm\_mpfs de Microchip pour en savoir plus.

## 10.4. CONCLUSION

Ce document devrait permettre, en s'inspirant des exemples effectués, de créer d'autres applications supportant TCP/IP.

## 10.5. HISTORIQUE DES VERSIONS

### 10.5.1. VERSION 1.5 MAI 2015

Création du document étape par étape. Version 1.5 pour indiquer l'utilisation de Harmony.

### 10.5.2. VERSION 1.51 JUIN 2015

Ajout de l'intégration du générateur.

### 10.5.3. VERSION 1.6 JUIN 2016

Adaptation au projet sous Harmony 1.06 et Mplabx 3.10.

### 10.5.4. VERSION 1.6 B JUIN 2016

Adaptation de l'intégration du générateur.

### 10.5.5. VERSION 1.7 MAI 2017

Reprise du document par SCA. Relecture. Compléments page web et mise au propre.

### 10.5.6. VERSION 1.71 MAI 2019

Mise à jour pour Harmony 2 (config. du projet exemple pic32mx\_eth\_sk2 remplace pic32mx\_eth\_sk qui n'existe plus, chip physique change).

### 10.5.7. VERSION 1.8 AVRIL 2022

Ajout portage exemple serveur TCP.

### 10.5.1. VERSION 1.81 JUIN 2022

Ajout réglage « close wait timeout » dans le paramétrage du serveur TCP afin de détecter correctement une déconnexion.