

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

4. Gestion des entrées-sorties	4-1
4.1. Version MPLABX & Harmony	4-1
4.2. Configuration des fusibles	4-1
4.2.1. Configuration générée dans system_init.c	4-2
4.2.2. Contenu du fichier system_init.c	4-2
4.3. Configuration des E/S	4-3
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
4.4. Concept des E/S digitales	4-14
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
4.5. Les fonctions de plib_ports.h	4-16
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
4.6.	BSP pic32mx_skes	4-27
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
4.7.	Gestion des leds et switch dans le BSP	4-33
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_EnableHbrige	4-36
4.8.	Gestion du convertisseur AD	4-37
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
4.9.	Les fonctions de plib_adc.h	4-41
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
4.10.	Application de test	4-48
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
4.11.	Historique des versions	4-53
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.

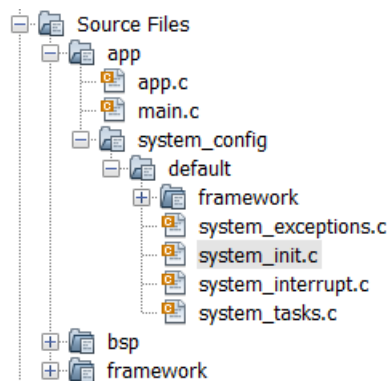
Le détail de la configuration ayant déjà été abordé au chapitre 2, nous aborderons ici uniquement le code résultant.

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 UPLEN =          ON

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

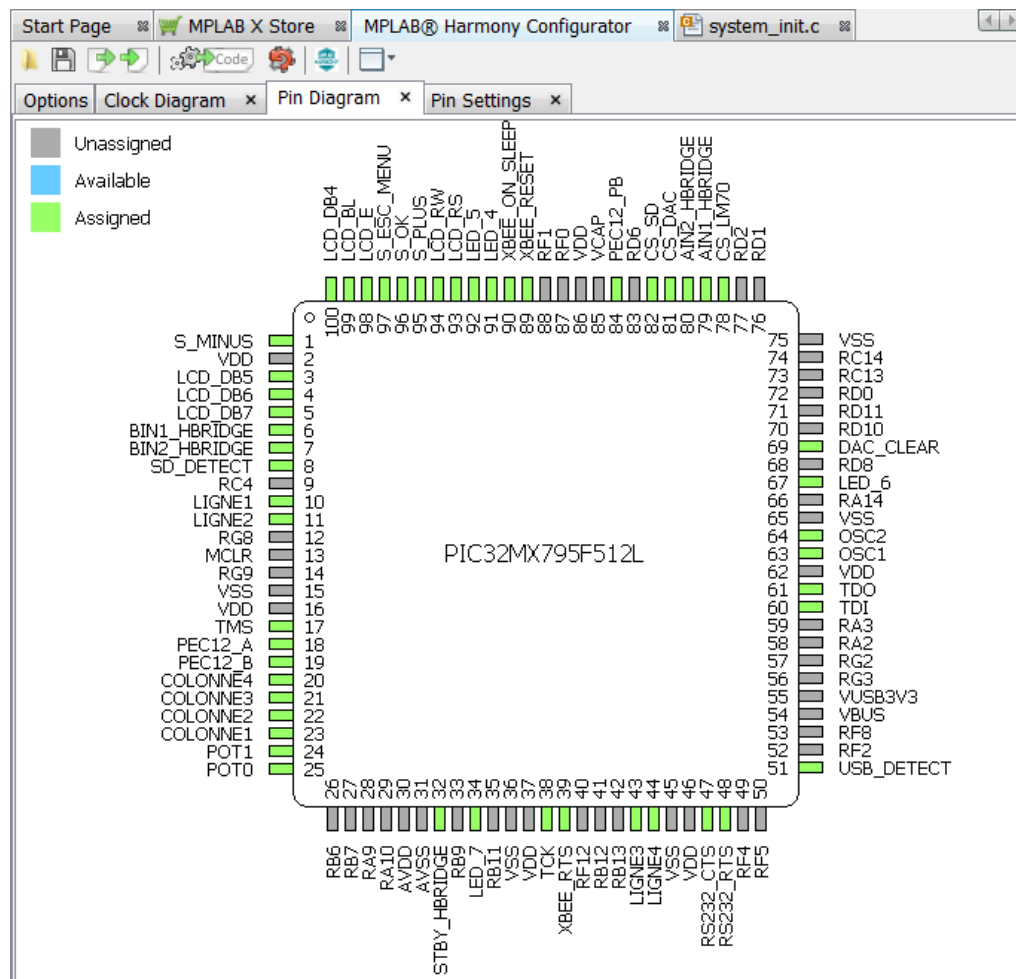
```


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

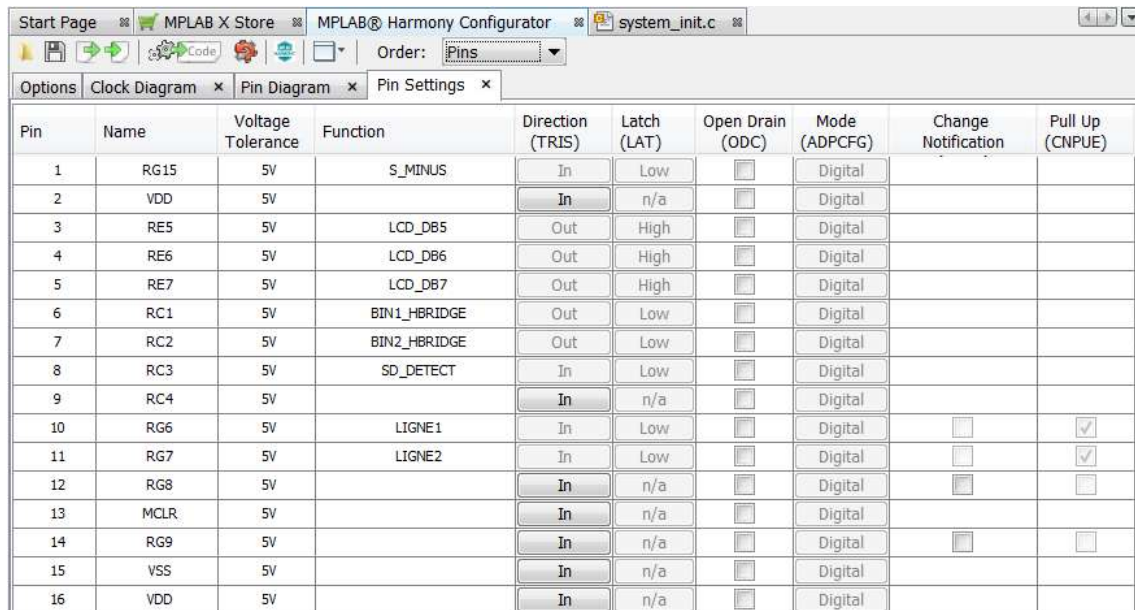
Une partie des définitions des entrées-sorties est fournie par le fichier **BSP.xml** du BSP sélectionné. Les choix imposés par le BSP peuvent être visualisés au niveau du *Pin Diagram* :



On remarque la personnalisation des broches.

4.3.2. PIN SETTING

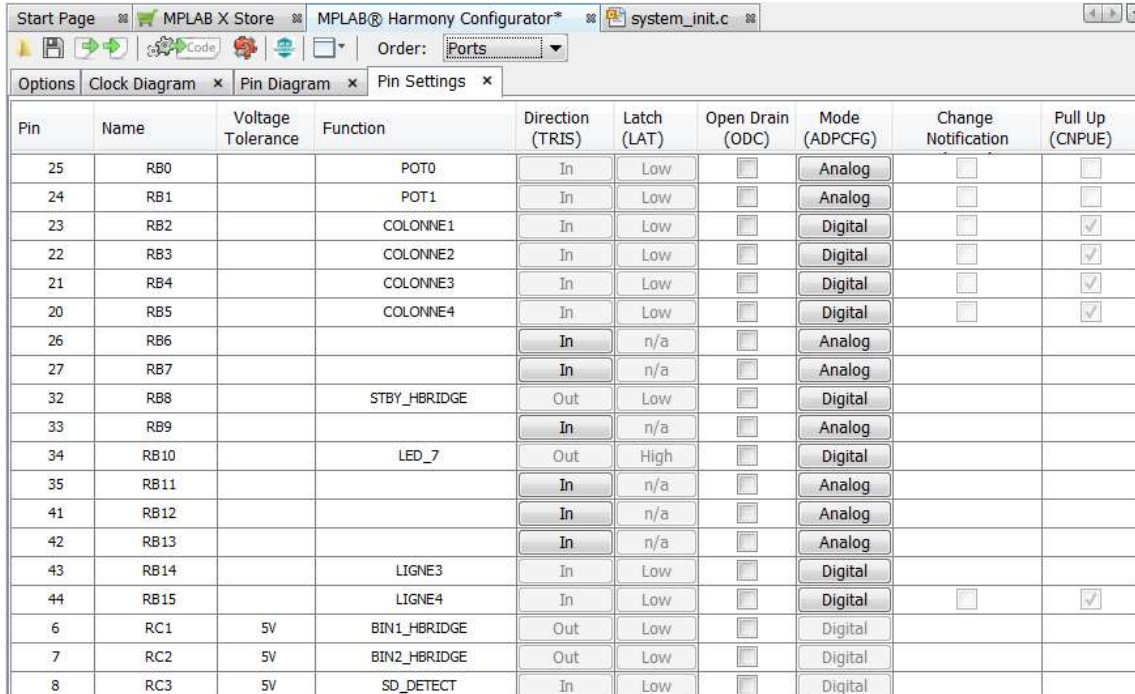
Sous l'onglet Pin Settings, on peut observer le détail de la configuration et modifier les broches qui n'ont pas été configurées.



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 type="checkbox"/>	<input checked="" type="checkbox"/>
11	RG7	5V	LIGNE2	In	Low	<input type="checkbox"/>	Digital	<input type="checkbox"/>	<input checked="" type="checkbox"/>
12	RG8	5V		In	n/a	<input type="checkbox"/>	Digital	<input type="checkbox"/>	<input type="checkbox"/>
13	MCLR	5V		In	n/a	<input type="checkbox"/>	Digital		
14	RG9	5V		In	n/a	<input type="checkbox"/>	Digital	<input type="checkbox"/>	<input 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		POT0	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 type="checkbox"/>	<input checked="" type="checkbox"/>
22	RB3		COLONNE2	In	Low	<input type="checkbox"/>	Digital	<input type="checkbox"/>	<input checked="" type="checkbox"/>
21	RB4		COLONNE3	In	Low	<input type="checkbox"/>	Digital	<input type="checkbox"/>	<input checked="" type="checkbox"/>
20	RB5		COLONNE4	In	Low	<input type="checkbox"/>	Digital	<input type="checkbox"/>	<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 type="checkbox"/>	<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.

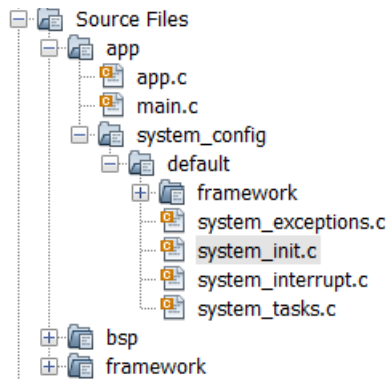
4.3.4. LES FONCTIONS LIÉES AU BSP

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 TMR0 */
    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 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 `PGECn` 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_CnPinsPullUpEnable(PORTS_ID_0,
        SYS_PORT_CNPUEN);
    PLIB_PORTS_CnPinsEnable(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);

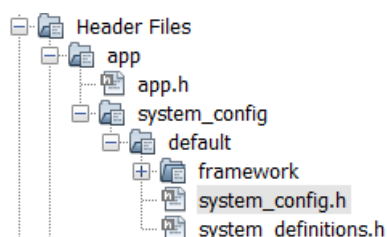
    /* PORT F Initialization */
    PLIB_PORTS_OpenDrainEnable(PORTS_ID_0, PORT_CHANNEL_F,
        SYS_PORT_F_ODC);
    PLIB_PORTS_Write( PORTS_ID_0, PORT_CHANNEL_F,
        SYS_PORT_F_LAT);
    PLIB_PORTS_DirectionOutputSet( PORTS_ID_0,
        PORT_CHANNEL_F,  SYS_PORT_F_TRIS ^ 0xFFFF);

    /* 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_UPLL_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_AD1PCFG          ~0x3ac3
#define SYS_PORT_CNPUE            0x0
#define SYS_PORT_CNEN             0x0

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

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

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

#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.

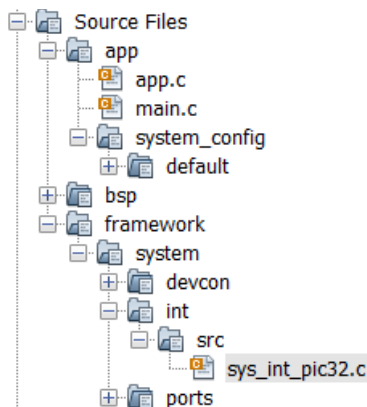
☞ La fonction est quasiment vide. A cause des broches non configurées, il est nécessaire d'effectuer une configuration des broches qui peuvent être analogiques ou digitales.

```
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 :

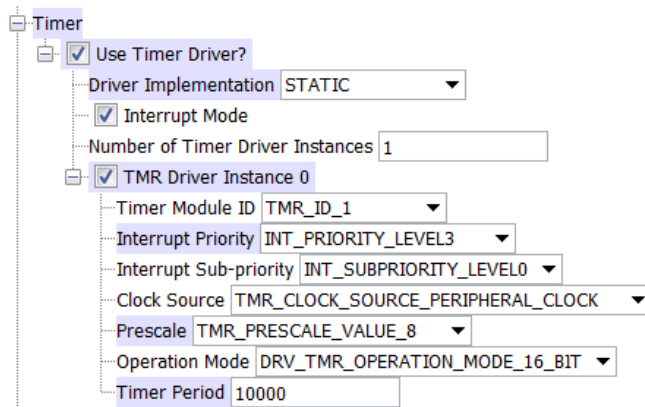


```
void SYS_INT_Initialize ( void )
{
    /* enable the multi vector */
    PLIB_INT_MultiVectorSelect( INT_ID_0 );
}
```

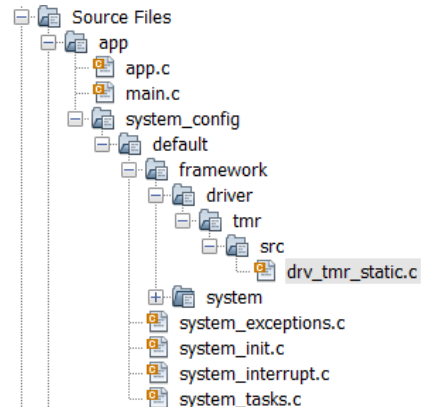
4.3.4.2.5. La fonction DRV_TMR0_Initialize

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_Model6BitEnable(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.

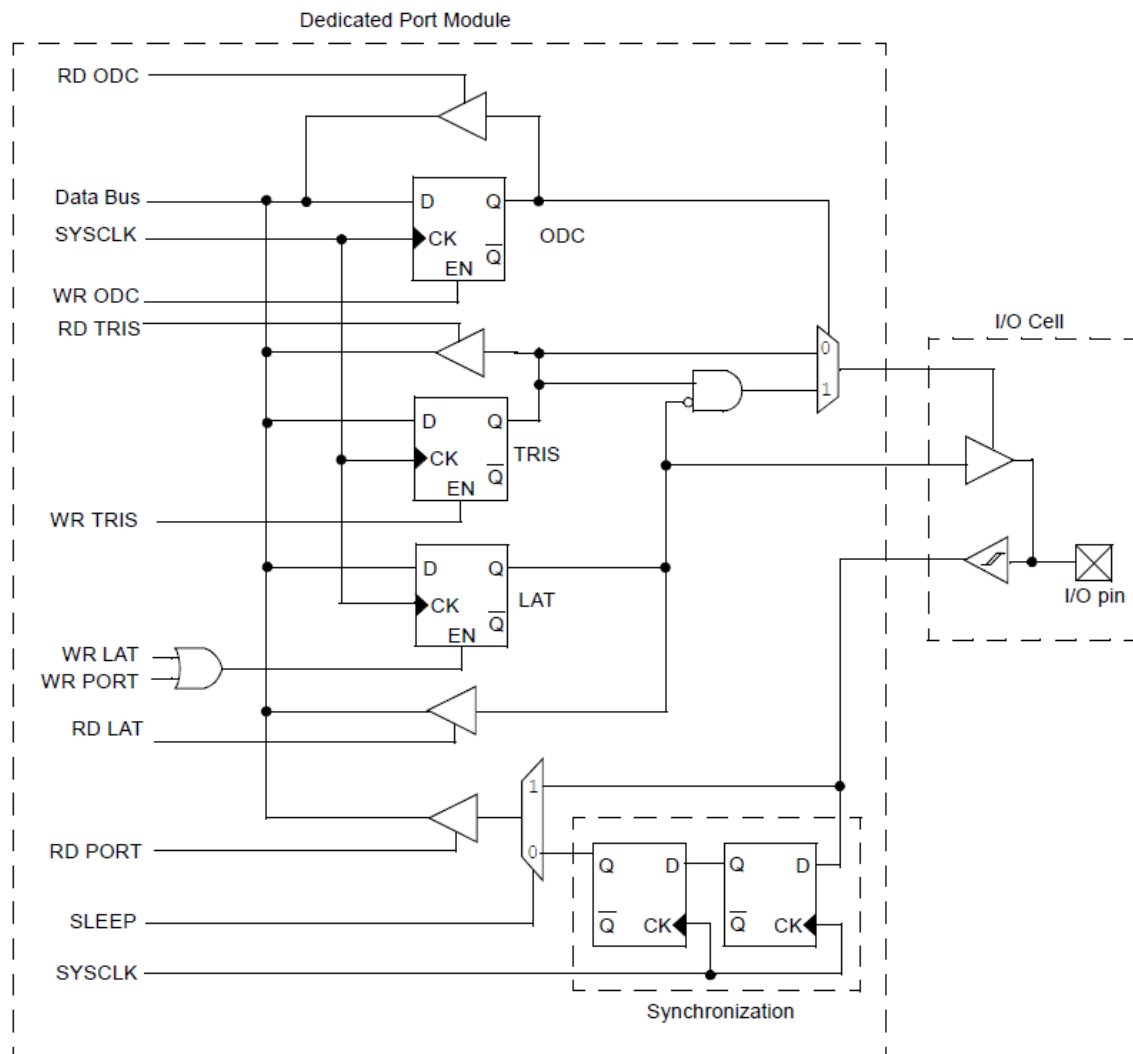
Un exemple d'application est présenté dans la dernière section de ce document.

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 ⁽¹⁾	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⁽¹⁾

Ambient temperature under bias	-40°C to +85°C
Storage temperature	-65°C to +150°C
Voltage on V _{DD} with respect to V _{SS}	-0.3V to +4.0V
Voltage on any pin that is not 5V tolerant, with respect to V _{SS} (Note 3)	-0.3V to (V _{DD} + 0.3V)
Voltage on any 5V tolerant pin with respect to V _{SS} when V _{DD} ≥ 2.3V (Note 3)	-0.3V to +5.5V
Voltage on any 5V tolerant pin with respect to V _{SS} when V _{DD} < 2.3V (Note 3)	-0.3V to +3.6V
Voltage on V _{BUS} with respect to V _{SS}	-0.3V to +5.5V
Voltage on V _{CORE} with respect to V _{SS}	-0.3V to 2.0V
Maximum current out of V _{SS} pin(s)	300 mA
Maximum current into V _{DD} pin(s) (Note 2)	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 (Note 2)	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
⚡	<code>PLIB_PORTS_Clear</code>	Clears the selected digital port/latch bits.
⚡	<code>PLIB_PORTS_DirectionGet</code>	Reads the direction of the selected digital port.
⚡	<code>PLIB_PORTS_DirectionInputSet</code>	Makes the selected pins direction input
⚡	<code>PLIB_PORTS_DirectionOutputSet</code>	Makes the selected pins direction output.
⚡	<code>PLIB_PORTS_PinOpenDrainDisable</code>	Disables the open drain functionality for the selected pin.
⚡	<code>PLIB_PORTS_PinOpenDrainEnable</code>	Enables the open drain functionality for the selected pin.
⚡	<code>PLIB_PORTS_Read</code>	Reads the selected digital port.
⚡	<code>PLIB_PORTS_Set</code>	Sets the selected bits of the Port
⚡	<code>PLIB_PORTS_Toggle</code>	Toggles the selected digital port/latch.
⚡	<code>PLIB_PORTS_Write</code>	Writes the selected digital port/latch.
⚡	<code>PLIB_PORTS_OpenDrainDisable</code>	Disables the open drain functionality for the selected port.
⚡	<code>PLIB_PORTS_OpenDrainEnable</code>	Enables the open drain functionality for the selected port pins.
⚡	<code>PLIB_PORTS_ReadLatched</code>	Reads/Gets data from the selected Latch.
⚡	<code>PLIB_PORTS_ChannelModeSelect</code>	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^{er} groupe de fonctions.

```
/* AN and CN Pins Initialization */
PLIB_PORTS_AnPinsModeSelect(PORTS_ID_0,
                             SYS_PORT_AD1PCFG, PORTS_PIN_MODE_DIGITAL);
PLIB_PORTS_CnPinsPullUpEnable(PORTS_ID_0,
                               SYS_PORT_CNPUEN);
PLIB_PORTS_CnPinsEnable(PORTS_ID_0, SYS_PORT_CNEN);
PLIB_PORTS_ChangeNoticeEnable(PORTS_ID_0);
```

4.5.5.1.1. La fonction PLIB_PORTS_AnPinsModeSelect

La fonction PLIB_PORTS_AnPinsModeSelect permet d'établir si une pin est utilisée en digital ou en analogique (pour les pins ayant une fonction analogique).

```
PLIB_PORTS_AnPinsModeSelect(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_CnPinsPullUpEnable

La fonction PLIB_PORTS_CnPinsPullUpEnable 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_CnPinsPullUpEnable(PORTS_ID_0, CHANGE_NOTICE_PIN_5 |
                                CHANGE_NOTICE_PIN_8 |
                                CHANGE_NOTICE_PIN_13);
```

4.5.5.1.3. La fonction PLIB_PORTS_CnPinsEnable

La fonction PLIB_PORTS_CnPinsEnable 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_CnPinsEnable(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_30,
    PORTS_ANALOG_PIN_31,
    PORTS_ANALOG_PINS_ALL
} 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 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;

EtatLed5 = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                     LED5_PORT, LED5_BIT);
EtatLed6 = 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",
            EtatLed5, EtatLed6, 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. ☹ Établit 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);
```

Togg (inverse) les bits 15, 7, 6, 5, 4, 1 et 0 du port A, ce qui inverse les leds 6 à 0.

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 allume 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 éteint 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 éteint 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;
EtatLed5 = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                     LED5_PORT, LED5_BIT);
EtatLed5 = !EtatLed5;
PLIB_PORTS_PinWrite(PORTS_ID_0, LED5_PORT,
                    LED5_BIT, EtatLed5);
```

¶ On remarque l'utilisation de la fonction **PinGetLatched** pour connaître 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 :

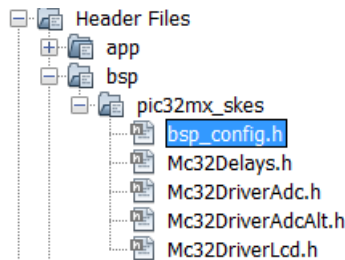
```
#define LED4_W      LATAbits.LATA6    // Led4 écriture
#define LED4_R      PORTAbits.RA6     // Led4 lecture
```


4.6. BSP PIC32MX_SKES

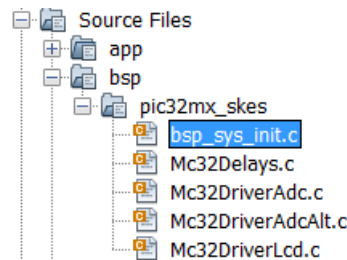
Le BSP (Board Support Package) spécifique au starter-kit ES a été réalisé en modifiant les fichiers d'un BSP microchip et sur la base du starter-kit version B.

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.

PLIB_PORTS_PinClear	PLIB_PORTS_PinToggle
PLIB_PORTS_PinSet	PLIB_PORTS_PinGet
PLIB_PORTS_PinWrite	PLIB_PORTS_PinGetLatched

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
```

```
// Définitions 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 switches 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_EnableHbrige(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

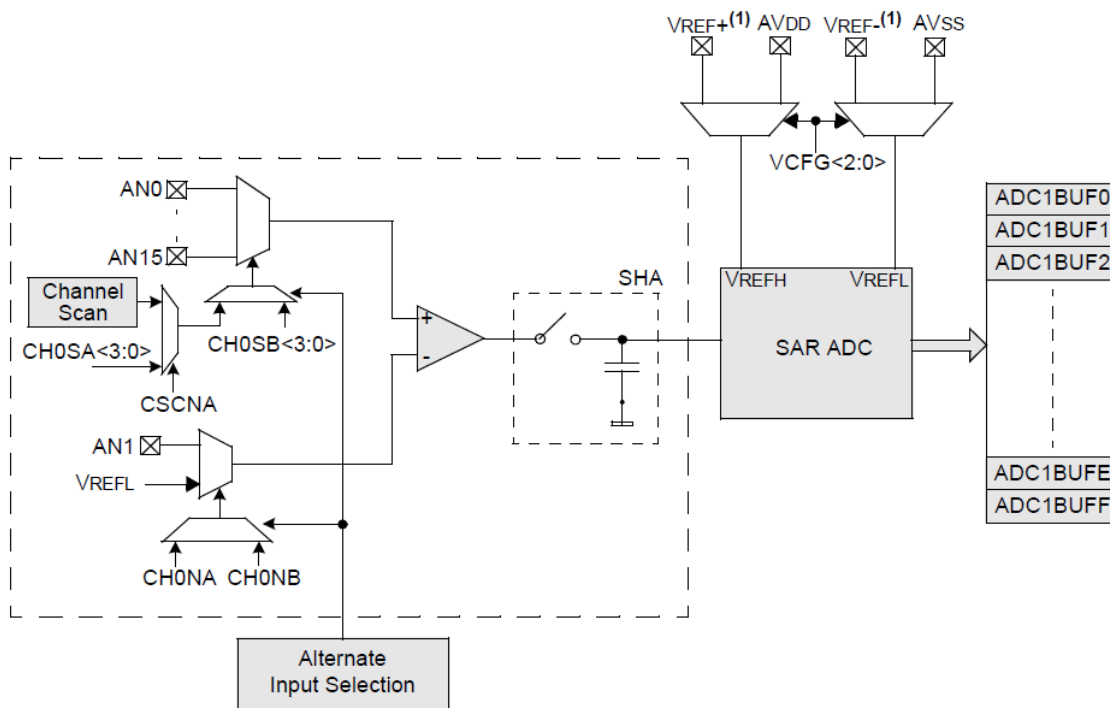
    STBY_HBRIDGE_W = 1;   // STBY High
}
```

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 :

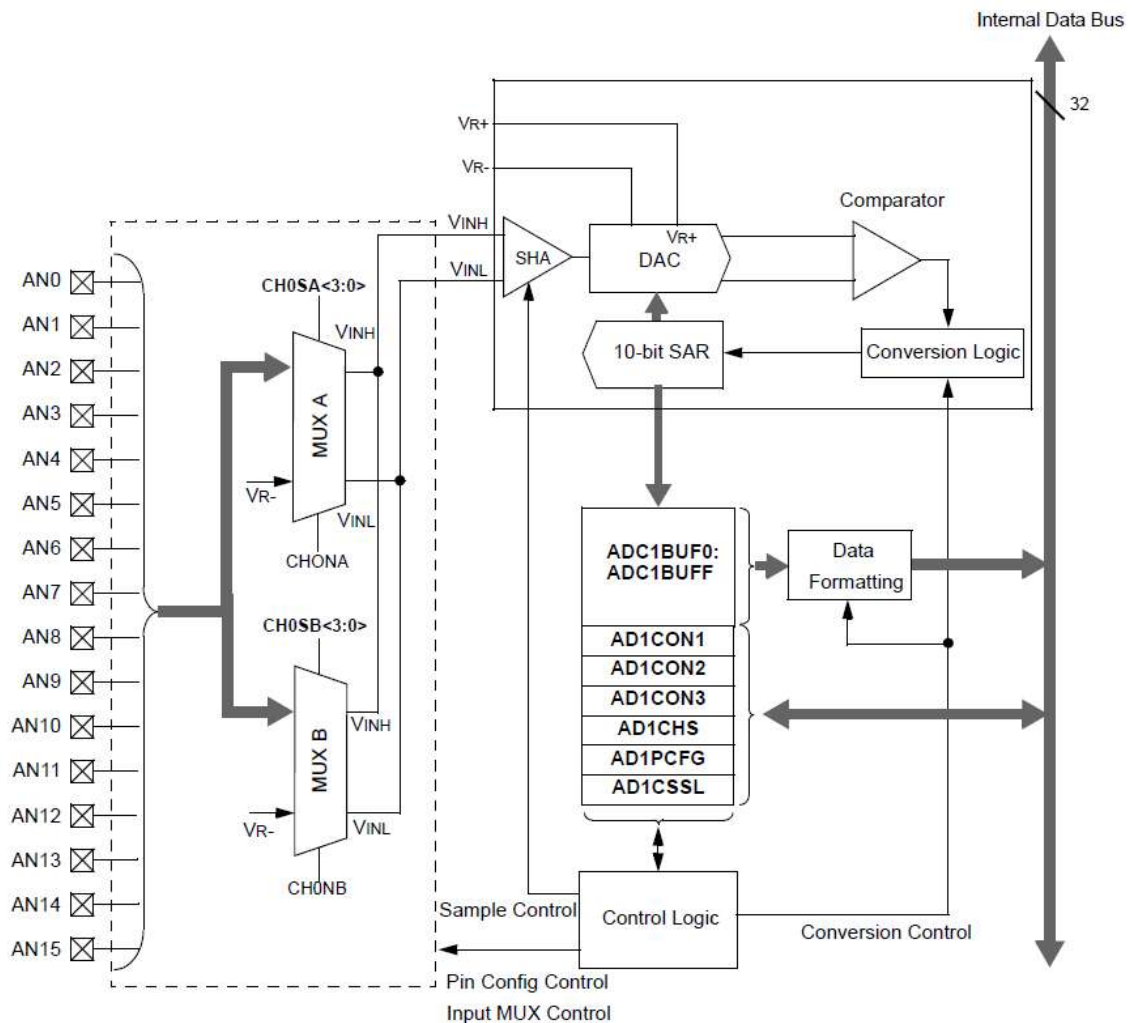


Ses caractéristiques sont :

- 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 alternate, 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

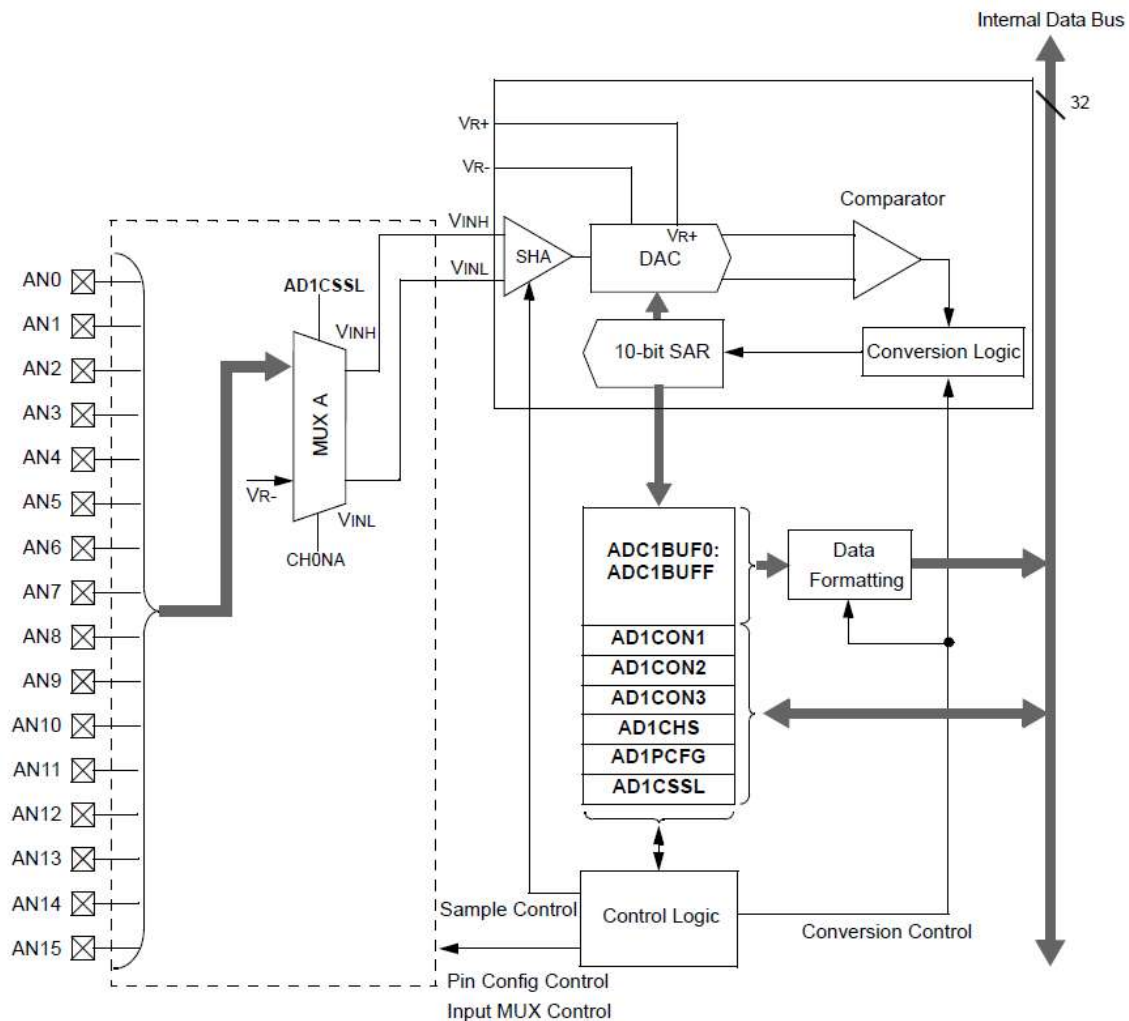
En mode alterné, les entrées provenant du MUX A et du MUX B sont converties en alternance.



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.

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

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

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 Mc32DRIVERADCA1T.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.c` 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'état 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_bl_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
    EtatLed5 = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                         LED5_PORT, LED5_BIT);
    EtatLed5 = !EtatLed5;
    PLIB_PORTS_PinWrite(PORTS_ID_0, LED5_PORT,
                        LED5_BIT, EtatLed5);

    // 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
    EtatLed5 = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                         LED5_PORT, LED5_BIT);
    EtatLed6 = PLIB_PORTS_PinGetLatched(PORTS_ID_0,
                                         LED6_PORT, LED6_BIT);
    ToucheOK = PLIB_PORTS_PinGet(PORTS_ID_0,
                                  S_OK_PORT, S_OK_BIT);

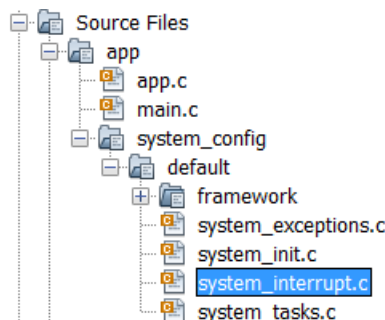
    lcd_gotoxy(1,3);
    printf_lcd("led5 %1d led6 %1d OK = %1d",
               EtatLed5, EtatLed6, ToucheOK);
    VallatchA = PLIB_PORTS_ReadLatched(PORTS_ID_0,
                                       PORT_CHANNEL_A);

    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 avons 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 aisé de déclencher une action toute 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.