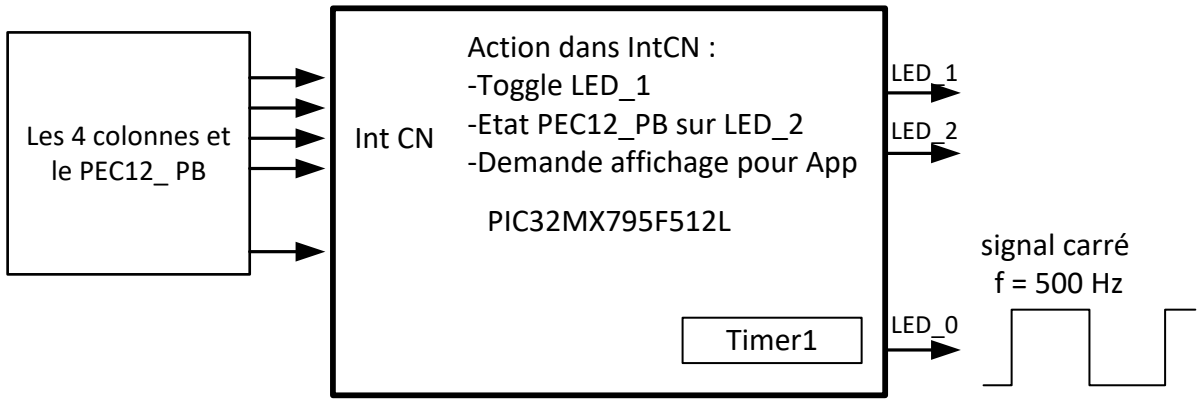
SOLUTION EXERCICE 5_2 PIC32MX

OBJECTIFS

Cet exercice a pour objectif de permettre aux étudiants de mettre en place une interruption de Change Notification pour des entrées et de vérifier concrètement son fonctionnement.

Après l'ébauche sur le papier, l'exercice sera réalisé pratiquement avec un kit PIC32MX795F512L.

Voici le schéma de principe du système demandé :



- Dans la réponse à l'interruption du timer 1, on inverse la LED0 de manière à obtenir un signal carré d'une fréquence de 500 Hz.
- Dans la réponse à l'interruption Change Notification, on effectue les actions suivantes :
    - Inversion (toggle) de la LED1 pour permettre une observation.
    - Lecture état des 5 entrées CN et enregistrement dans appData.
    - Affectation état PEC12_PB sur LED2.
    - Activation de l'application pour gestion de l'affichage.

RAPPEL THEORIQUE

NATURE DES CNX

On dispose de 22 CN, de CN0 à CN21.

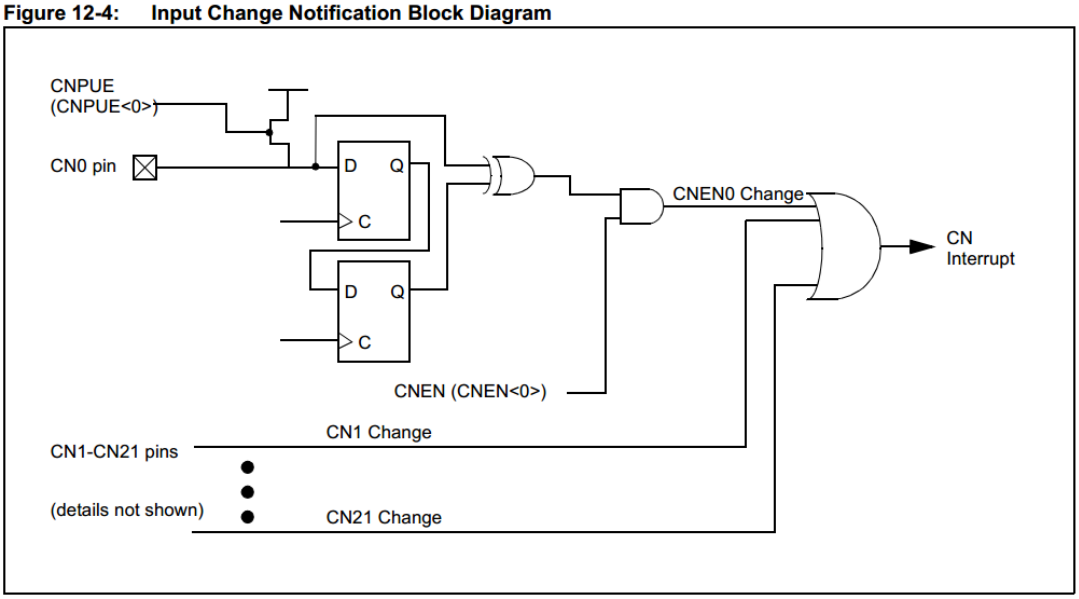
☝ Un CNx est un rôle de plus associé à une broche.

Voici la liste des CN pour le PIC32MX795F512L en boitier 100 broches :

CNx	**Nom complet de la pin**	**No pin**
CN0	SOSC0/T1CK/**CN0**/RC14	74
CN1	SOSCI/**CN1**/RC13	73
CN2	PGED1/AN0/**CN2**/RB0	25
CN3	PGEC1/AN1/**CN3**/RB1	24
CN4	AN2/C2IN-/**CN4**/RB2	23
CN5	AN3/C2IN+/**CN5**/RB3	22
CN6	AN4/C1IN-/**CN6**/RB4	21
CN7	AN5/C1IN+/Vbuson/**CN7**/RB5	20
CN8	ECOL/SCK2/U6TX/_U3RTS/PMA5/**CN8**/RG6	10
CN9	ECRS/SDA4/SDI2/**U3RX**/PMA4/**CN9**/RG7	11
CN10	ERXDV/AERXDV/ECRSDV/AECRSDV SCL4/SDO2/**U3TX**/PMA3/**CN10**/RG8	12
CN11	ERXCLK/AERXCLK/EREFCLK/AEREFCLK _SS2/**U6RX**/_U3CTS/PMA2/**CN11**/RG9	14
CN12	AN15/ERXD3/AETXD2/OCFB/PMALL /PMA0/**CN12**/RB15	44
CN13	OC5/PMWR/**CN13**/RD4	81
CN14	PMRD/**CN14**/RD5	82
CN15	ETXEN/PMD14/**CN15**/RD6	83
CN16	ETXCLK/PMD15/**CN16**/RD7	84
CN17	SDA5/SDI4/U2RX/PMA9/**CN17**/RF4	49
CN18	SCL5/SDO4/U2TX/PMA8/**CN18**/RF5	50
CN19	ETXD3/PMD13/**CN19**/RD13	80
CN20	AETXD0/_SS3/U4RX/_U1CTS/**CN20**/RD14	47
CN21	AETXD1/SCK3/U4TX/_U1RTS/**CN21**/RD15	48

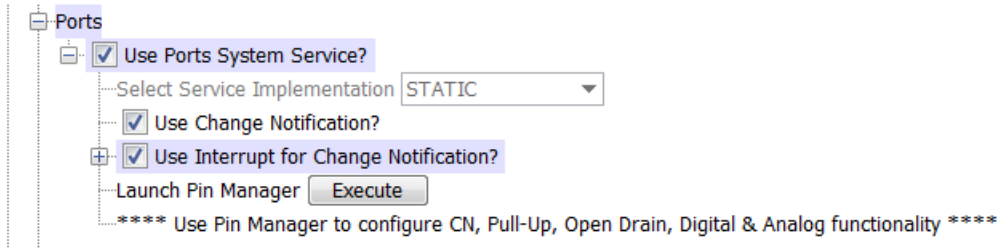
SCHEMA BLOC DU MECANISME D'INPUT CHANGE NOTIFICATION

Figure 12-4: Input Change Notification Block Diagram



ACTION POSSIBLE AU NIVEAU DU MHC

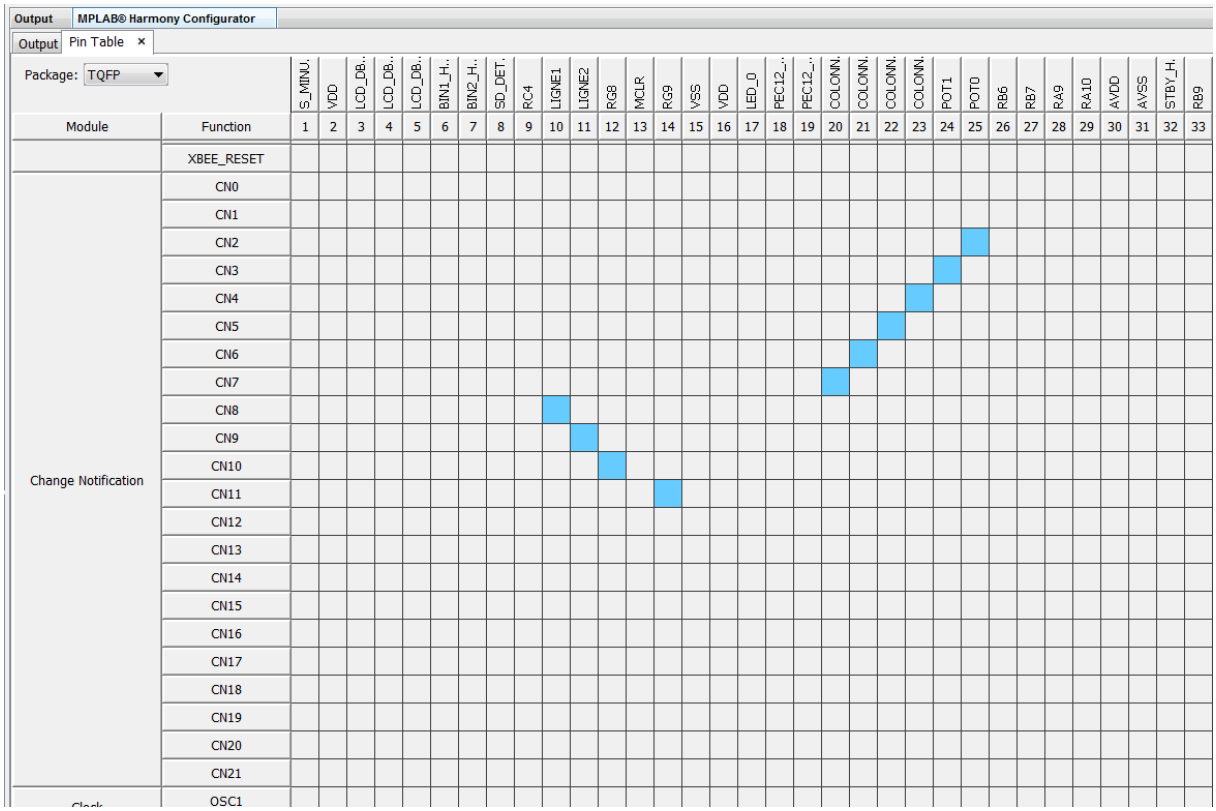
Au niveau du MHC, dans Harmony Framework Configuration > System Services > Ports, il est possible d'établir la configuration suivante :



Avec 

, on aboutit au Pin Manager.

Le système offre la possibilité d'activer graphiquement un CN pour les broches qui en ont la possibilité (indiqué par les carrés bleus).

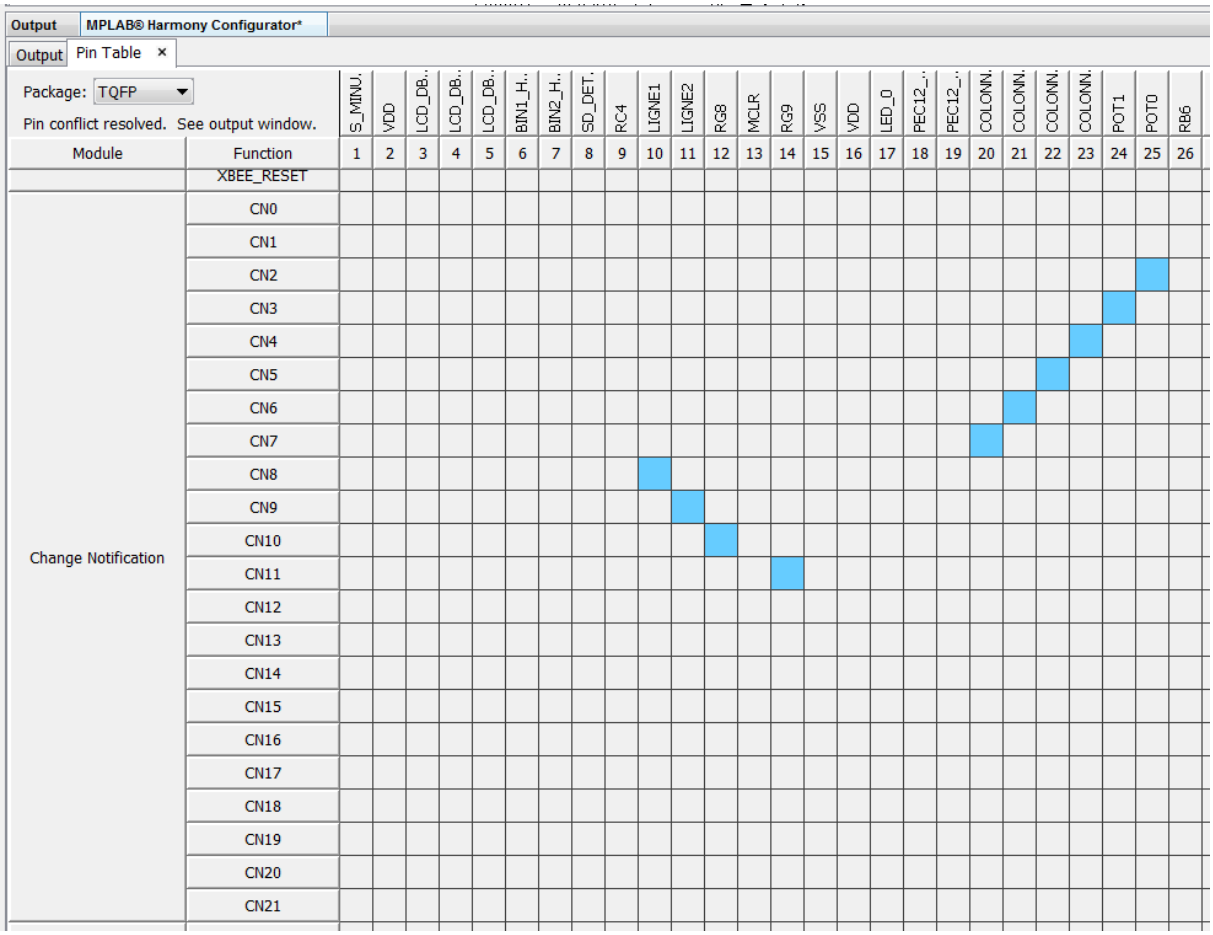


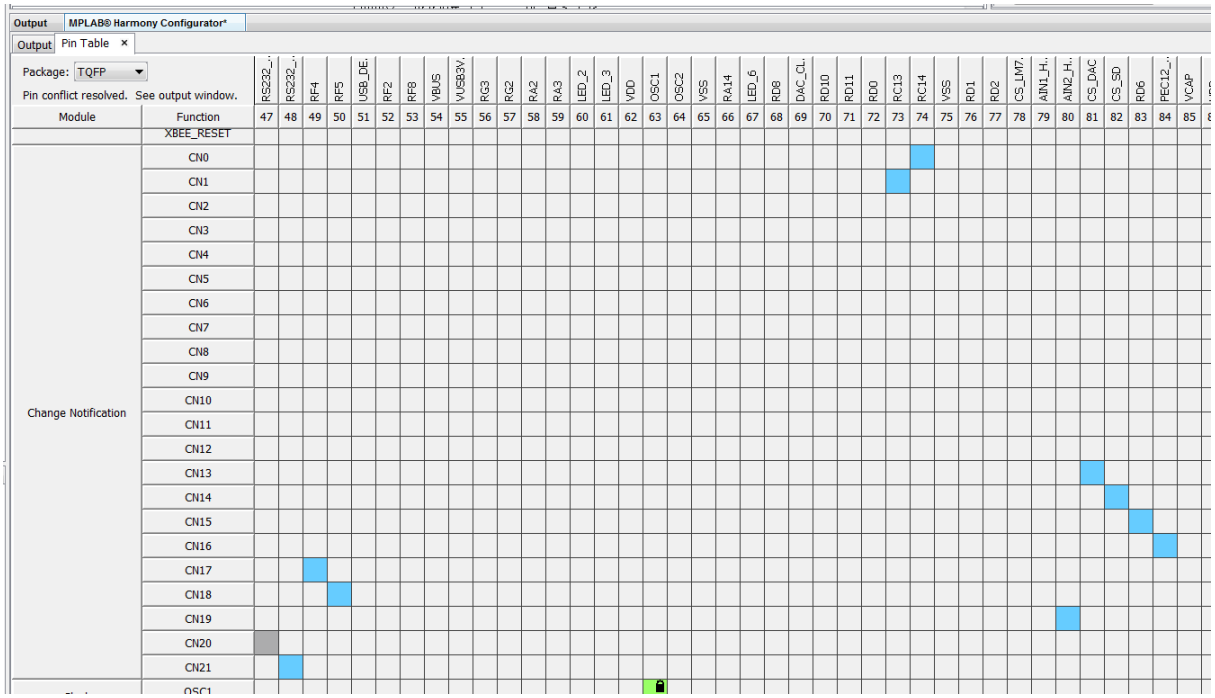
Cette sélection remplace la définition du BSP par le CNx correspondant.

☺ Lors de la génération du code, l'interruption CN sera activée pour les pins sélectionnées, et l'ISR générée.
L'ISR est partagée entre tous les CN.

LES CN VUS DU PIN TABLE

Voici l'ensemble des CN et leur correspondance avec les broches et les définitions du BSP.





a) En vous basant sur les copies d'écran des pin table, indiquez à quels élément du Kit PIC32MX795F512L correspondent les CNx.

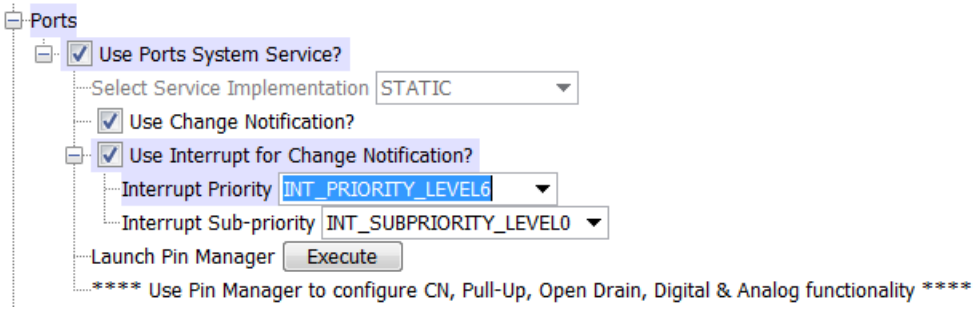
CNx	Nom complet de la pin	No pin	Rôles Kit
CN0	SOSC0/T1CK/**CN0**/RC14	74	
CN1	SOSCI/**CN1**/RC13	73	
CN2	PGED1/AN0/**CN2**/RB0	25	**POT0**
CN3	PGEC1/AN1/**CN3**/RB1	24	**POT1**
CN4	AN2/C2IN-/**CN4**/RB2	23	**COLONNE1**
CN5	AN3/C2IN+/**CN5**/RB3	22	**COLONNE2**
CN6	AN4/C1IN-/**CN6**/RB4	21	**COLONNE3**
CN7	AN5/C1IN+/Vbuson/**CN7**/RB5	20	**COLONNE4**
CN8	ECOL/SCK2/U6TX/_U3RTS/PMA5/**CN8**/RG6	10	**LIGNE1**
CN9	ECRS/SDA4/SDI2/**U3RX**/PMA4/**CN9**/RG7	11	**LIGNE2**
CN10	ERXDV/AERXDV/ECRSDV/AECRSDV SCL4/SDO2/**U3TX**/PMA3/**CN10**/RG8	12	
CN11	ERXCLK/AERXCLK/EREFCLK/AEREFCLK _SS2/**U6RX**/_U3CTS/PMA2/**CN11**/RG9	14	
CN12	AN15/ERXD3/AETXD2/OCFB/PMALL /PMA0/**CN12**/RB15	44	
CN13	OC5/PMWR/**CN13**/RD4	81	**CS_DAC**
CN14	PMRD/**CN14**/RD5	82	**CS_SD**
CN15	ETXEN/PMD14/**CN15**/RD6	83	
CN16	ETXCLK/PMD15/**CN16**/RD7	84	**PEC12_PB**
CN17	SDA5/SDI4/U2RX/PMA9/**CN17**/RF4	49	
CN18	SCL5/SDO4/U2TX/PMA8/**CN18**/RF5	50	
CN19	ETXD3/PMD13/**CN19**/RD13	80	
CN20	AETXD0/_SS3/U4RX/_U1CTS/**CN20**/RD14	47	
CN21	AETXD1/SCK3/U4TX/_U1RTS/**CN21**/RD15	48	**RS232_RTS**

b) Pour obtenir un signal carré de 500 Hz en inversant LED0 à chaque interruption du timer 1, quelle doit être sa période ? Déterminez également le prescaler.

Période d'un signal carré 500 Hz : 1/500 = 0.002 [s] soit 2 [ms]
Période du timer 1 : inversion à la demi période → 1 ms. 1000 us * 80 = 80'000. Besoin d'un prescaler de 80'000 / 65536 = 1.22 donc 2 mais le plus proche disponible est 8.
Période = 1000 * 80 / 8 = 10'000 → **9'999**
Prescaler du timer 1 : **8**

REALISATION PRATIQUE

- Création d'un projet avec Harmony pour le kit PIC32MX
- Utilisation du timer 1 avec interruption. prendre niveau 4.
- Utilisation des CN pour les pins voulus, avec interruption. Prendre niveau 6.



MODIFICATION DU FICHIER SYSTEM_INTERRUPT.C

- Compléter l'ISR du timer 1 avec l'inversion de la LED0. Utilisez la fonction BSP_LEDToggle. Incrémentez le champ Elapsedms de appData.

```
// Timer 1 @ 1kHz
void __ISR(_TIMER_1_VECTOR, ipl4AUTO)
                            IntHandlerDrvTmrInstance0(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_1);

    BSP_LEDToggle(BSP_LED_0); //pour observation

    //incrémentation ms
    if (appData.Elapsedms < 0xFFFFFFFF)
        appData.Elapsedms++;
}
```

- Compléter l'ISR de CHANGE_NOTICE avec les actions suivantes :
  - L'inversion de la LED1 avec la fonction BSP_LEDToggle.
  - Lire les 4 colonnes et affecter les 4 booléens correspondants dans appData en utilisant la fonction PLIB_PORTS_PinGet.
  - Lire l'état du PEC12_PB et l'affecter à la LED2 par action directe.
  - Ajouter l'appel à la fonction APP_UpdateState avec (APP_STATE_SERVICE_TASKS) en paramètre.
  - Effectuer le SourceFlagClear après la lecture des CN ! (nécessaire au mécanisme).

```
void __ISR(_CHANGE_NOTICE_VECTOR, ipl6AUTO)
                          _IntHandlerChangeNotification(void)
{
    BSP_LEDToggle(BSP_LED_1); //pour observation interrupt
    LED2_W = PEC12_PB; // pour observation état PEC12_PB
    appData.EtatCol1 = PLIB_PORTS_PinGet( PORTS_ID_0,
                              COLONNE1_PORT, COLONNE1_BIT);
    appData.EtatCol2 = PLIB_PORTS_PinGet( PORTS_ID_0,
                              COLONNE2_PORT, COLONNE2_BIT);
    appData.EtatCol3 = PLIB_PORTS_PinGet( PORTS_ID_0,
                              COLONNE3_PORT, COLONNE3_BIT);
    appData.EtatCol4 = PLIB_PORTS_PinGet( PORTS_ID_0,
                              COLONNE4_PORT, COLONNE4_BIT);
    appData.EtatPec12Pb = PLIB_PORTS_PinGet( PORTS_ID_0,
                              PEC12_PB_PORT, PEC12_PB_BIT);

    PLIB_INT_SourceFlagClear(INT_ID_0,
                               INT_SOURCE_CHANGE_NOTICE);
    APP_UpdateState(APP_STATE_SERVICE_TASKS);
}
```

Remarque : Il faut lire l'état des entrées CN avant de quittancer l'interruption !

MODIFICATION DES FICHIER APP.H & APP.C

- Ajoutez l'état WAIT et la fonction APP_UpdateState. Mettre en place le switch avec les 3 états.
- Ajouter les éléments suivants dans le typedef APP_DATA :

```
bool EtatCol1;
bool EtatCol2;
bool EtatCol3;
bool EtatCol4;
bool EtatPec12Pb;
uint32_t Elapsedms;
```

CONTENU DE APP.H

```
typedef enum
{
    /* Application's state machine's initial state. */
    APP_STATE_INIT=0,
    APP_STATE_WAIT,
    APP_STATE_SERVICE_TASKS
} APP_STATES;
typedef struct
{
    /* The application's current state */
    APP_STATES state;
    /* TODO: Define any additional data used by the
application. */
    uint32_t Elapsedms;

    bool EtatCol1;
    bool EtatCol2;
    bool EtatCol3;
    bool EtatCol4;
    bool EtatPec12Pb;

} APP_DATA;
```

☝ Ne pas oublier :

```
extern APP_DATA appData;  // pour usage public
```

Et le prototype de la fonction APP_UpdateState().

CONTENU DE APP.C

Actions **case APP_STATE_INIT** :

- Init. lcd et backlight ON
- Mettre à 0 Elapsedms
- Pour que la lecture des colonnes fonctionne, il faut un niveau '0' sur les lignes. Etablir les 4 lignes en sortie, à 0 (utilisation des fonctions PLIB_PORTS).
- Etablir appData.state à WAIT

```
case APP_STATE_INIT:
{
    appData.Elapsedms = 0;

    //Mise en sortie à '0' des 4 lignes (B14-B15-G6-G7)
    PLIB_PORTS_DirectionOutputSet(PORTS_ID_0,
            PORT_CHANNEL_B, 0xC000);
    PLIB_PORTS_DirectionOutputSet(PORTS_ID_0,
            PORT_CHANNEL_G, 0x00C0);
    PLIB_PORTS_Clear(PORTS_ID_0, PORT_CHANNEL_B, 0xC000);
    PLIB_PORTS_Clear(PORTS_ID_0, PORT_CHANNEL_G, 0x00C0);

    //init LCD
    lcd_init();
    lcd_bl_on();
    printf_lcd("Solution Ex 5_2 ");
    lcd_gotoxy(1,2);
    printf_lcd("SCA 12.2017");
    lcd_gotoxy(1,3);
    printf_lcd("%c %c %c %c %c", '-', '-', '-', '-', '-');

    //démarre timer 0
    DRV_TMR0_Start();

    appData.state = APP_STATE_WAIT;
    break;
}
```

Action **case APP_STATE_WAIT** :

- Si Elapsedms > 1000, éteindre le rétro-éclairage.

```
case APP_STATE_WAIT:
{
    //extinction BL après 1 s d'inactivité
    if(appData.Elapsedms > 1000)
        lcd_bl_off();

    break;
}
```

Actions case **APP_STATE_SERVICE_TASKS** :

- Remettre ON le rétro-éclairage et Elapsedms à 0
- Afficher état des 5 entrées CN
- Etablir appData.state à WAIT

```
case APP_STATE_SERVICE_TASKS:
{
    //nouvelle action ==> BL on
    appData.Elapsedms = 0;
    lcd_bl_on();

    //affiche états CN
    lcd_gotoxy(1,3);
    printf_lcd("%d %d %d %d %d",
          appData.EtatCol1, appData.EtatCol2,
          appData.EtatCol3, appData.EtatCol4,
          appData.EtatPec12Pb);

    appData.state = APP_STATE_WAIT;
    break;
}
```

La fonction APP_UpdateState() :

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

TESTS ET OBSERVATIONS

- Timer :
  Vérifiez à l'oscilloscope si le signal sur LED0 est bien à 500 Hz.
- CN :
  - Observez si une pression sur une des touches provoque une interruption (observation LED1)...
  - ...ainsi que l'affichage correspondant.
  - Vérifiez si la LED2 reflète l'état de PEC12_PB.

On peut observer le comportement souhaité.