```
/*******************************************************************************
  MPLAB Harmony Application Source File

  Company:
    Microchip Technology Inc.

  File Name:
    app.c

  Summary:
    This file contains the source code for the MPLAB Harmony application.

  Description:
    This file contains the source code for the MPLAB Harmony application.  It
    implements the logic of the application's state machine and it may call
    API routines of other MPLAB Harmony modules in the system, such as drivers,
    system services, and middleware.  However, it does not call any of the
    system interfaces (such as the "Initialize" and "Tasks" functions) of any of
    the modules in the system or make any assumptions about when those functions
    are called.  That is the responsibility of the configuration-specific system
    files.
 *******************************************************************************/

// DOM-IGNORE-BEGIN
/*******************************************************************************
Copyright (c) 2013-2014 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES

(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
 *******************************************************************************/
// DOM-IGNORE-END
```

```
// *****************************************************************************
// *****************************************************************************
// Section: Included Files
// *****************************************************************************
// *****************************************************************************

#include "app.h"
#include "Mc32DriverLcd.h"
#include "Mc32Debounce.h"
#include "peripheral/oc/plib_oc.h"
#include "driver/tmr/drv_tmr_static.h"

MENU_STATE MenuState;
INF_CODAGE InfCodage;

int valHour = 0;
int valMinute = 0;
int valSeconde = 0;

// *****************************************************************************
// *****************************************************************************
// Section: Global Data Definitions
// *****************************************************************************
// *****************************************************************************

// *****************************************************************************
/* Application Data

  Summary:
    Holds application data

  Description:
    This structure holds the application's data.

  Remarks:
    This structure should be initialized by the APP_Initialize function.

    Application strings and buffers are be defined outside this structure.
*/


APP_DATA appData;

// *****************************************************************************
// *****************************************************************************
// Section: Application Callback Functions
// *****************************************************************************
// *****************************************************************************

/* TODO:  Add any necessary callback functions.
```

```
*/

// *****************************************************************************
// *****************************************************************************
// Section: Application Local Functions
// *****************************************************************************
// *****************************************************************************


/* TODO:  Add any necessary local functions.
*/


// *****************************************************************************
// *****************************************************************************
// Section: Application Initialization and State Machine Functions
// *****************************************************************************
// *****************************************************************************

/*******************************************************************************
  Function:
    void APP_Initialize ( void )

  Remarks:
    See prototype in app.h.
 */

void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    //Initialisation du LCD

       // Init debounce
    DebounceInit(&DescrSW1);

    DebounceInit(&DescrSW2);
    DebounceInit(&DescrSW3);
    DebounceInit(&DescrSW4);

    //Etat suivant
    appData.state = APP_STATE_INIT;

    /* TODO: Initialize your application's state machine and other
     * parameters.
     */
}

/******************************************************************************
  Function:
    void APP_Tasks ( void )
```

```
  Remarks:
    See prototype in app.h.
 */

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

            //Strating Timer
            DRV_TMR0_Start();
            DRV_TMR1_Start();
            DRV_TMR2_Start();

            //Starting OC
            DRV_OC0_Start();

            //Affichage initial
            lcd_gotoxy(1,1);
            printf_lcd("Emetteur DCF 18/19");
            lcd_gotoxy(1,2);

            printf_lcd("Julie Culand");

            MenuState = MENU_STATE_INIT;

            //Etat suivant
            appData.state = APP_STATE_WAIT;
            break;
        }
         case APP_STATE_WAIT:
        {
            //nothing to do
            break;
        }
        case APP_STATE_SERVICE_TASKS:
        {
            //Affichage initial (Menu)
            GESTION_MENU();

            //Bouton OK est pressé -> on rentre dans le menu pour régler l'heure
            if(DebounceIsPressed(&DescrSW1))
            {
```

```
            {
                DebounceClearPressed(&DescrSW1);
                MenuState = MENU_STATE_HOUR_AFF;
            }
            appData.state = APP_STATE_WAIT;
            break;
        }

        /* TODO: implement your application state machine.*/


        /* The default state should never be executed. */
        default:
        {
            /* TODO: Handle error in application's state machine. */
            break;
        }
    }
}
//Fonction APP_UpdateState
void APP_UpdateState (APP_STATES newState)
{
    appData.state = newState;
}


//******* FONCTION POUR L'HEURE *******
//Fonction qui permet de régler l'heure
void GESTION_MENU ()
{
    bool FlagChange = false;

    switch(MenuState)
    {
        //Affichage initial des menus
        case MENU_STATE_INIT :
        {
            lcd_gotoxy(1, 1);
            printf_lcd("******* MENU *******");
            lcd_gotoxy(1, 2);
            printf_lcd("1. S1-Reglages heure");
            lcd_ClearLine(3);
            //Affichage de l'heure réglée
            lcd_gotoxy(1, 4);
            printf_lcd("Heure: %2d:%2d:%2d ",valHour, valMinute, valSeconde);
            break;
        }
        //Affichage du menu du réglage de l'heure
        case MENU_STATE_HOUR_AFF :
        {
            //Etat '0' pour bloquer l'incrémentation de l'heure
```

```
            etatReglHour = 0;
            //Affichage LCD
            lcd_gotoxy(1, 1);
            printf_lcd("** REGLAGE HEURE **");
            lcd_gotoxy(1, 2);
            printf_lcd("Heure Minute Seconde");
            lcd_gotoxy(1, 3);
            printf_lcd("      %2d:%2d:%2d    ", valHour, valMinute, valSeconde);

            lcd_gotoxy(1, 4);
            printf_lcd("S1 pour valider");
            //Etat suivant
            MenuState = MENU_STATE_HOUR;
            break;
        }
        //Réglages de l'heure
        case MENU_STATE_HOUR :
        {

            //Bouton "SW2" incrémente les heures
            if(DebounceIsPressed(&DescrSW2))
            {
               DebounceClearPressed(&DescrSW2);
               valHour += 1;
               FlagChange = true;
            }
            rebouclementHour (&valHour);

            //Bouton "SW3" incrémente les minutes
            if(DebounceIsPressed(&DescrSW3))
            {
                DebounceClearPressed(&DescrSW3);
                valMinute += 1;
                FlagChange = true;
            }
            rebouclementMinute (&valMinute);

            //Bouton "SW4" incrémente les secondes
            if(DebounceIsPressed(&DescrSW4))
            {
                DebounceClearPressed(&DescrSW4);
                valSeconde += 1;
                FlagChange = true;
            }
            rebouclementSeconde (&valSeconde);

            //Affichage des réglages
            if(FlagChange == true)
            {
                lcd_gotoxy(1, 3);
```

```
                printf_lcd("      %2d:%2d:%2d    ", valHour, valMinute, valSeconde);
                lcd_gotoxy(1, 4);
                printf_lcd("S1 pour valider");
            }

            //Bouton "SW1" valide les réglages de l'heure
            if(DebounceIsPressed(&DescrSW1))
            {
                DebounceClearPressed(&DescrSW1);
                //Retour au menu initial
                MenuState = MENU_STATE_INIT;
                etatReglHour = 1;
            }

            break;
        }
    }
}
// ********** FONCTION REBOUCLEMENT HEURE ********** //
//Fonction de rebouclement pour l'heure
void rebouclementHour (int *valHour)
{
    if(*valHour > 23)
    {
        *valHour = 0;
    }
    else if (*valHour <= 0)
    {
        *valHour = 0;
    }
}

//Fonction de rebouclement pour les minutes
void rebouclementMinute (int *valMinute)
{
    if(*valMinute == 59)
    {
        *valMinute = 0;
    }
    else if(*valMinute <=0)
    {
        *valMinute = 0;
    }
}

//Fonction de rebouclement pour les secondes
void rebouclementSeconde (int *valSeconde)
{
    if(*valSeconde == 60)
    {
```

```
        *valSeconde = 0;
    }
    else if(*valSeconde <= 0)
    {
        *valSeconde = 0;
    }
}


// ********** FONCTION INCREMENTATION HEURE ********** //
//Fonction qui incrémente les heures
void incrementHour()
{
    //Incrémente les heures
    valHour ++;
    //Lorsque l'on arrive à 23heures, on repasse à 0
    if(valHour == 24)
    {
        valHour = 0;
    }
}

//Fonction qui incrémente les minutes
void incrementMinute()
{
    //Incrémente les minutes
    valMinute ++;
    //Lorsque l'on arrive à 59 minutes, on repasse à 0
    if(valMinute == 59)
    {
        valMinute = 0;
        //Active les heures après 59 minutes
        incrementHour();
    }
}

//Fonction qui incrémente les secondes
void incrementSeconde()
{
    //Incrémente les secondes
    valSeconde ++;
    //Lorsque l'on arrive à 60 secondes, on repasse à 0
    if(valSeconde == 60)
    {
        valSeconde = 0;
        //Active les minutes après 60 secondes
        incrementMinute();
    }
}

// ******** CODE DES INTERRUPTIONS ********
```

```
// ******** CODAGE DES INFORMATIONS HORAIRES ********
//Fonction qui contient une machine d'état
// pour le codage des informations de l'heure

void codageHour ()
{
    int valEnvoiMinute = 0;
    int valEnvoiHour = 0;

    //Inactif (bit à 0) ; Actif (bit à 1)
    switch(InfCodage)
    {
        //0 : Début de trame, bit toujours à 0
        case INIT_INF :
            //Valeur des minutes et heures convertie à l'initialisation
            //Valeur des minutes récupérée en BCD
            valEnvoiMinute = DecimalToBCD(valMinute);
            //Valeur de l'heure récupérée en BCD
            valEnvoiHour = DecimalToBCD(valMinute);
            //Si les secondes valent '0' alors on débute l'envoi
            if (valSeconde == 0)
            {
                sendBit(0);
                InfCodage = RESERVED_INF; //Etat suivant
            }
            break;
        // 1 - 14 : Réservé pour une utilisation futur
        // 15 : Inactif
        case RESERVED_INF :
            sendBit(0);
            if(valSeconde == 15)
                InfCodage = FUSEAU_INF; //Etat suivant
            break;
        //16 : Annonce l'heure d'hiver ; inactif
        //17 - 18 : Fuseau horaire ; inactif
        case FUSEAU_INF :
            sendBit(0);
            if(valSeconde == 18)
                InfCodage = START_INF; //Etat suivant
            break;
        //19 : Suppression d'une seconde pour corriger irrégulatiré
        //rotation terre ; inactif
        //20 : Début de codage des informations horaire, bit toujours à 1
        case START_INF :
            if(valSeconde == 19)
                sendBit(0);
            else if (valSeconde == 20)
            {

            sendBit(1);
            InfCodage = MINUTES_INF; //Etat suivant
```

```
            }
            break;
        //21 -27 : Minutes codées en BCD
        // 28 : Bit de parité paire des minutes
        case MINUTES_INF :
            //Valeurs récupérées bit par bit et envoi des données
            sendBit(ExtractBit (valEnvoiMinute, valSeconde - 21));
            if(valSeconde == 28)
            {
                //Envoi de la partié
                sendBit(getValParite(valEnvoiMinute, 7));
                InfCodage = HEURES_INF; //Etat suivant
            }
            break;
        //29 - 34 : Heures codées en BCD
        // 35 : bit de parité paire des heures
        case HEURES_INF :
            //Valeurs récupérées bit par bit et envoi des données
            sendBit(ExtractBit(valEnvoiHour, valMinute - 29));
            if(valSeconde == 35)
            {
                //Envoi de la parité
                sendBit(getValParite(valEnvoiHour, 6));
                InfCodage = JOUR_INF; //Etat suivant
            }
            break;
        //36 - 41 : Jour ; inactif
        case JOUR_INF :
            sendBit(0);
            if(valSeconde == 41)
                InfCodage = JOUR_SEM_INF; //Etat suivant
            break;
        //42 - 44 : Jour de la semaine ; inactif
        case JOUR_SEM_INF :
            sendBit(0);
            if(valSeconde == 44)
                InfCodage = MOIS_INF; //Etat suivant
            break;
        //45 - 49 : Mois ; inactif
        case MOIS_INF :
            sendBit(0);
            if(valSeconde == 49)

                InfCodage = ANNEE_INF; //Etat suivant
            break;
        //50 - 57 : Année ; inactif
        //58 : Bit de parité paire de la date
        case ANNEE_INF :
            sendBit(0);
            if(valSeconde == 58)
```

```
                InfCodage = END_INF; //Etat suivant
            break;
        //59 : Pas d'impulsion ; indique fin de trame
        case END_INF :
            InfCodage = INIT_INF; //Etat suivant
            break;
    }
}
// Fonction de conversion Décimal -> BCD
int8_t DecimalToBCD (int valDecimal)
{
    return (((valDecimal / 10) << 4) | (valDecimal%10));
}

//Fonction d'extraction des bits
bool ExtractBit (int Data, int numBit)
{
    int MaskValBit = 0;

    MaskValBit = 1 << numBit;
    return (Data & MaskValBit) >> numBit;
}

// Fonction qui définit la durée de l'impulsion
void sendBit (bool etatVal)
{
    DRV_TMR2_CounterClear();
    if(etatVal == 1)
    {
        //Valeur pour une impulsion de 200ms
        PLIB_OC_Buffer16BitSet(OC_ID_4, 62500);
    }
    else
    {
        //Valeur pour une impulsion de 100ms
        PLIB_OC_Buffer16BitSet(OC_ID_4, 31250);
    }

    DRV_OC1_Enable();
}

//Fonction qui permet de vérifier si la donnée est paire ou impaire
bool getValParite(uint8_t Data, int NbBit)
{
    bool bitParite = 0;
    int i = 0;

    for(i = 0 ; i < NbBit ; i++)
    {
        bitParite ^= (Data & 0x01);
```

```
        Data >>= 1;
    }
    return bitParite;
}
/*******************************************************************************
 End of File
 */
```