```
//*****************************************************************************
//  Nom:              main.c
//
// Projet:            FitLux
//
// Description:   Programme de gestion du FitLux                                *
//
//
// Auteur:              Gregory Layaz - JDC Electronic SA                       *
// Date:                02.02.09
//
// Dernière modif:   20.02.12
//
//
//*****************************************************************************
#include <msp430x22x2.h>
#include <stdio.h>
#include <stdlib.h>
#include "main.h"

//*****************************************************************************
// Déclaration des variables globales
//*****************************************************************************
unsigned char STATE_MACHINE = DEFAULT; // Machine d'état du système

unsigned int ovf_counter = 0;    // Compteur d'overflow lors de capture du temps
unsigned int counterTime = 0;   // Compteur du temps écoulé
unsigned int endTime = 0;       // Temps total maximum d'allumage en fonction de l'énergie fournie
unsigned int endTimeTmp = 0;    // Temps intermediaire d'allumage en fonction de l'énergie fournie

unsigned int calculSlope = 0;  // Flag pour le calcul de la droite pour l'extinction de la LED
float slope = 0;                // Pente de la droite
unsigned int offset = 0;       // Offset de la droite

//*****************************************************************************
// Fonctions
//*****************************************************************************
/*****************************************************************************
 * Name      : timerAInit
 * Goal      : Il est utilisé pour calculer le temps que l'utilisateur produit
 *             de l'énergie. Ce temps peut être plus important que un overflow
 *             donc ils sont compris dans le calcul total du temps de fabrication
 *             d'énergie.
 * Author    : GLZ
 * Parameter : -
 * Output    : -
 *****************************************************************************/
void timerAInit(void)
{
  // Initialisation du Timer A
  TACTL |= TASSEL_1;          // Clk. = ACLK
  // Initialisation du mode capture pour CCR0
  TACCTL0 |= (CM_2+SCS+CAP);  // Cap. mode, falling edge, CCI0A, sync.
  TACCTL0 &= ~CCIE;             // Bloquage des interruptions TACCR0
}

/*****************************************************************************
 * Name      : startTimerA
 * Goal      : Mise en route du timerA
 * Author    : GLZ
 * Parameter : -
 * Output    : -
 *****************************************************************************/
void startTimerA(void)
{
  // Initialisation du port P1 pour TIMER
  P1SEL |= BIT1;         // P1.1 = fonction secondaire, CCI0A
  P1DIR &= ~BIT1;
  // Autorisation des interruptions du système de mesure (Timer A)
  TACTL |= TACLR;             // Clear Timer A
  TACTL &= ~TAIFG;            // Effacement du flag d'overflow du Timer A
  TACCTL0 &= ~(CCIFG+COV);  // Effacement du flag d'interruption TACCR0
  TACCTL0 |= CCIE;            // Autorisation des interruptions TACCR0
  TACTL |= TAIE;              // Autorisation des interruptions d'overflow Timer A

  // Enclenchement des Timer A et B (mesure et stop)
  TACTL |= MC_2;            // Timer A ON in Continous Mode
}
```

```
/*****************************************************************************
 * Name      : timerBInit
 * Goal      : Il est utiliser en mode PWM pour commander le driver de led
 * Author    : GLZ
 * Parameter : -
 * Output    : -
 *****************************************************************************/
unsigned int PWM_TB = PWM_MAX;
void timerBInit(void)
{
  P4DIR |= BIT1 + BIT2;                          // P4.1 - P4.2 output
  P4SEL |= BIT1 + BIT2;                          // P4.1 - P4.2 TBx options

  TBCCR0  = PWM_MAX-1;                         // PWM Period
#ifdef REF_200
  TBCCTL1 = OUTMOD_7;                         // TBCCR1 reset/set
  TBCCR1  = PWM_TB;                          // TBCCR1 PWM duty cycle
#else
  TBCCTL2 = OUTMOD_7;                         // TBCCR1 reset/set
  TBCCR2  = PWM_TB;                          // TBCCR1 PWM duty cycle
#endif /*REF_200*/

  TBCTL   = TBSSEL_2 + MC_1;                 // SMCLK, up mode
}

/*****************************************************************************
 * Name      : watchDogInit
 * Goal      : Créer une interruption toutes les secondes
 * Author    : GLZ
 * Parameter : -
 * Output    : -
 *****************************************************************************/
void watchDogInit(void)
{
  // Activation du timer watchdog pour interruption toutes les 1[s]
  WDTCTL = WDT_ADLY_1000;       // clock = ACLK 32k
  IE1   |= WDTIE;               // Activation des interruptions
}

/*****************************************************************************
 * Name      : clockInit
 * Goal      : Initialisation des clock
 * Author    : GLZ
 * Parameter : -
 * Output    : -
 *****************************************************************************/
void clockInit(void)
{
  BCSCTL1 = CALBC;                    // Set DCO
  DCOCTL = CALDC;
}

/*****************************************************************************
 * Name      : PortInit
 * Goal      : Initialisation des portrs
 * Author    : GLZ
 * Parameter : -
 * Output    : -
 *****************************************************************************/
void PortInit(void)
{
  P1DIR = 0xFD;
  P2DIR = 0xFE;
  P3DIR = 0xFF;
  P4DIR = 0xFF;

  P1OUT = 0x00;
  P2OUT = 0x00;
  P3OUT = 0x00;
  P4OUT = 0x00;

  // Initialisation pour le compteur
  P1DIR &= ~BIT1;
  P1IFG &= ~BIT1;
  P1SEL &= ~BIT1;
  P1IES &= ~BIT1;
  P1IE  |= BIT1;

  // Initialisation de la sortie du PWM
```

```
  P4OUT |=  PWM;
  P1OUT &= ~CS;
}

/*****************************************************************************
 * Name      : comparatorInit
 * Goal      : Initialisation du comparateur
 * Author    : GLZ
 * Parameter : -
 * Output    : -
 *****************************************************************************/
void comparatorInit(void)
{
  // Initialisation pour le comparateur branché sur la batterie
  P2DIR &= ~BIT0;
  P2IFG &= ~BIT0;
  P2SEL &= ~BIT0;
  P2IES |= BIT0;
  P2IE  &= ~BIT0;
}

/*****************************************************************************
 * Name      : Port1Init
 * Goal      : Initialisation du port 1
 * Author    : GLZ
 * Parameter : -
 * Output    : -
 *****************************************************************************/
void Port1Init(void)
{
  P1IFG &= ~BIT1;
  P1SEL &= ~BIT1;
  P1IES &= ~BIT1;
  P1IE  |= BIT1;
}

/*****************************************************************************
 * Name      : main
 * Goal      : Main loop
 * Author    : GLZ
 * Parameter : -
 * Output    : -
 *****************************************************************************/
void main(void)
{
  // Arret du watchdog
  WDTCTL = WDTPW + WDTHOLD;   // Arrêt du watchdog

  // Bloquage général des interruptions
  _BIC_SR(GIE);
  // Initialisation des clocks
  clockInit();
  // Initialisation du port
  PortInit();
  // Initialisation du Timer A
  timerAInit();
  // Initialisation du comparateur de la batterie
  comparatorInit();

  while(1){

    switch(STATE_MACHINE)
    {
    case START:
     P2IE  &= ~BIT0;          // Désactive l'interruption du comparateur de la batterie
     startTimerA();
     calculSlope = 0;
//      P1IES |= BIT1;         // Met le déclenchement en high-to-low
//      _BIS_SR(GIE+LPM3_bits);
      _BIS_SR(GIE+LPM0_bits);
     _NOP();
      break;
    case STOP:
      P1SEL &= ~BIT1;
      P1IFG &= ~BIT1;         // Effacement du Flag d'interruption
     P1IES &= ~BIT1;       // Met le déclenchement en low-to-high
     P1IE  |= BIT1;      // Active l'interruption pour la détection de la tension
     P2IE  |= BIT0;        // Active l'interruption du comparateur de la batterie
     //_BIS_SR(GIE+LPM3_bits);
```

```
    _BIS_SR(GIE+LPM0_bits);
    _NOP();
     break;
   case COUNTER:
     break;
   case DEFAULT:
   default:
     P1SEL &= ~BIT1;
     P1IFG &= ~BIT1;          // Effacement du Flag d'interruption
    P1IES &= ~BIT1;        // Met le déclenchement en low-to-high
    P1IE  |= BIT1;       // Active l'interruption pour la détection de la tension
    P2IFG &= ~BIT0;        // Effacement du Flag d'interruption
    P2IE  |= BIT0;         // Active l'interruption du comparateur de la batterie
     // Autorisation générale des interruptions
     _BIS_SR(GIE+LPM4_bits);
     _NOP();
     break;
   }
  };
}

//*****************************************************************************
// Routines d'interruption
//*****************************************************************************

//*****************************************************************************
// Fonction:    wake_up
// Paramètres:  In:  -
//              Out: -
// Description: Routine d'interruption du watchdog chaque seconde pour
//              prise de mesure.
//*****************************************************************************
#pragma vector=WDT_VECTOR
__interrupt void wake_up (void)
{
  IFG1 &= ~WDTIFG;            // Effacement du flag du watchdog

  // Contrôle d'overflow
  if(counterTime < 65535) {
    counterTime++;  // Mise à jour de la variable temps
  }

  // Si l'utilisateur est en train de fournir de l'énergie alors il ne faut
  // pas baisser le niveau d'intensité lumineuse
  if((P1IN & BIT1) == 0) {
   // Si le temps maximum est ateint.. alors il faut éteindre la diode pas-à-pas.
   if(counterTime >= (endTime*0.5)) {
     if(calculSlope == 0) {
       offset = PWM_TB;
       slope = ((float)PWM_HALF - (float)offset) / ((float)endTime - (float)counterTime);
       calculSlope = 1;
     }
     PWM_TB = (unsigned int)(((float)counterTime - (float)endTime/2.0) * slope + (float)offset);
     if(counterTime >= endTime) {
      // Il faut éteindre la led...
       PWM_TB = PWM_MAX;
       P1OUT &= ~CS;
       counterTime = 0;
       endTime = 0;
       IE1 &= ~WDTIE;               // désactivation de l'interruption du watchdog
       P2IE&= ~BIT0;
       LPM3_EXIT;
        STATE_MACHINE = DEFAULT;
     }
      timerBInit();
    }
  }
  else {
   // Contrôle si la batterie est OK
   if((P2IN & BIT0) && (ovf_counter > PREAMBLE_TIME)){
      // Changement de la valeur du PWM, afin que l'éclairage augmente pas-à-pas
      if(PWM_TB >= PWM_PART_1) {
         PWM_TB -= PWM_DELTA_PART_1;
      }
      else {
         if(PWM_TB >= PWM_DELTA_PART_2) {
            PWM_TB -= PWM_DELTA_PART_2;
         }
         else if(PWM_TB > 0) {
```

```
            PWM_TB = 0;
          }
        }
        timerBInit();
    }
  }
}

//*************************************************************************
// Fonction:    mesure
// Paramètres:  In:  -
//              Out: -
// Description: Routine d'interruption du timer A pour TACCR0
//*************************************************************************
#pragma vector=TIMERA0_VECTOR
__interrupt void mesure (void)
{
  // Si on crée au moins pour 4 secondes d'énergie...
  if(ovf_counter >= PREAMBLE_TIME) {
    // Sauvegarde du dernier top de référence capturé
    // La valeur du compteur est concaténée au compteur d'overflow
    // => compteur de 32 bits
    // Contrôle d'overflow
    if(endTimeTmp < (65535 - (2*FACTOR_TIME))) {
      endTimeTmp += (unsigned int)(((float)(TACCR0)/32768.0) * (float)(FACTOR_TIME));
    }
    // Tant que la valeur du compteur dépasse pas la valeur maximum
    if(endTimeTmp <= MAX_TIME) {
      // Contrôle d'overflow
      if(endTime < (65535 - (2*FACTOR_TIME))) {
        endTime += (unsigned int)(((float)(TACCR0)/32768.0) * (float)(FACTOR_TIME));
      }
    }
    if(counterTime >= endTime) {
      counterTime = MAX_TIME / 2;
    }
  }

  // Il n'y a plus d'énérgie fournie par la génératrice
  if(!(P1IN & BIT1)) {
    // Remise à jour de la valeur du compteur d'overflow
    ovf_counter = 0;

    // Remise à zéro de la valeur du temps d'allumage intermédiaire
    endTimeTmp = 0;

    // Initialise le système pour qu'il soit pret pour recalculer les temps d'énergie fournie
    STATE_MACHINE = STOP;

    ////////////////////////////////////////////////////////////////////////
    // TODO: Pourquoi c'est fait de cette manière??? A corriger
    ////////////////////////////////////////////////////////////////////////
    // Contrôle si la batterie est vide...
    if(P2IN & BIT0) {
      P2IFG &= ~BIT0;
    }
    else {
      P2IFG |= BIT0;
    }

    // Stop le timer A
    TACCTL0 &= ~CCIE;   // Bloquage des interruption de TACCR0 (freq)
    TACTL &= ~TAIE;     // Bloquage des interruption d'overflow Timer A

    // Stop le timer en mode PWM si la led est allumée au maximum
    if((PWM_TB == 0)) {
      TBCTL = MC_0;
    }
    LPM3_EXIT;
  }
}

/*************************************************************************
 * Name       : phase_overflow
 * Goal       : Lorsqu'il y a un overflow sur le timerA. Cela veut dire que
 *              l'utilisateur a au moins créé durant 2 secondes de l'énérgie.
 *              Le watchdog est activé et la led commence à être allumée.
 * Author     : GLZ
 * Parameter : -
```

```
 * Output    : -
 ******************************************************************************/
//unsigned char stateBattery[10] = {0,0,0,0,0,0,0,0,0,0};
unsigned char stateBattery = 0xFF;
#pragma vector=TIMERA1_VECTOR
__interrupt void phase_overflow (void)
{
  unsigned int int_vector;          // Variable de lecture du registre TAIV

  int_vector = TAIV;          // Lecture du registre TAIV et reset automatique
  // Test du vecteur d'interruptions
  if(int_vector == TAIV_TAIFG){
    // Overflow du Timer A => Timer A = max => ré-init à 0
    TACTL &= ~TAIFG;        // Effacement du flag d'overflow du Timer A
    ++ovf_counter;          // Incrémentation du cpt d'overflow

    // Mise en route de l'interruption du watchog... toutes les 1 secondes.
    if(ovf_counter == PREAMBLE_TIME) {
      watchDogInit();
      // Changement de la valeur du PWM, afin que l'éclairage augmente pas-à-pas
      if(PWM_TB > PWM_HALF) {
        PWM_TB = PWM_HALF;
      }
      P1OUT |= CS;  // Activation du chip select
      endTime = endTime - counterTime; // On garde seulement le temps restant...
      counterTime = 0; // Mise à 0 du compteur
      timerBInit();
    }
    else if(ovf_counter > PREAMBLE_TIME) {
      // Contrôle d'overflow
      if(endTimeTmp < (65535 - (2*FACTOR_TIME))) {
        endTimeTmp +=   (2 * FACTOR_TIME); // Mise à jour du compteur
      }
      // Si le temps de "pompage" est plus petit que le temps maximum
      if(endTimeTmp <= MAX_TIME) {
         // Contrôle d'overflow
        if(endTime < (65535 - (2*FACTOR_TIME))) {
          endTime += (2 * FACTOR_TIME); // Mise à jour du compteur
        }
      }
    }
  }
}

/******************************************************************************
 * Name      : TestPort
 * Goal      : Lorsque l'utilisateur crée de l'énergie, il y une interrupt sur
 *             le BIT1 du port 1. Il faut allumer le compteur (timerA) afin de
 *             connaître le temps d'apport d'énergie.
 * Author    : GLZ
 * Parameter : -
 * Output    : -
 ******************************************************************************/
#pragma vector=PORT1_VECTOR
__interrupt void TestPort (void)
{
  if(P1IFG & BIT1) {
   P1IFG &= ~BIT1;          // Effacement du Flag d'interruption
   P1IE  &= ~BIT1;          // Désactive l'interruption
   STATE_MACHINE = START;
   LPM4_EXIT;
   _NOP();
  }
}

/******************************************************************************
 * Name      : ComparatorPort
 * Goal      : Controle si la batterie est toujours chargée... Si elle devient
 *             trop déchargée et que le Led est allumée alors il faut eteindre
 *             la Led malgré que le temps d'allumage n'est pas atteint.
 * Author    : GLZ
 * Parameter : -
 * Output    : -
 ******************************************************************************/
#pragma vector=PORT2_VECTOR
__interrupt void ComparatorPort (void)
{
  if (P2IFG & BIT0) {
   P2IFG   &= ~BIT0; // Effacement du Flag d'interruption
```

```
   P2IE    &= ~BIT0; // Désactive l'interruption du comparateur de la batterie
   TACCTL0 &= ~CCIE;   // Bloquage des interruption de TACCR0 (freq)
    TACTL   &= ~TAIE;   // Bloquage des interruption d'overflow Timer A
    TACTL   &= ~TAIFG;  // Effacement du flag d'overflow du Timer A
   if(endTime != 0) {
     endTime = 30;
     counterTime = 15;
   }
  }
}
```