Annexe S.5

```c
//  gestionBatterie.c
//
//  Description : fonctions liées à la gestion de la batterie
//  Auteur : Perret Mélissa
//  Création : 09/06/2024
//  Modifications : --

//  Version   : V1.0
/*-----------------------------------------------------------*/


#include "main.h"
#include "stdbool.h"
#include "fonctionsADC.h"
#include "gestionBatterie.h"


//// Fonction GestionCheckBatterie (regarder si une nouvelle vérification de la batterie est nécessaire)
//// Description: décrémente le compteur lié à la vérification de la batterie, déclenche le lancement de la vérification batterie si besoin
//// Entrées: Pointeur:ADC_HandleTypeDef hadc, Pointeur:TIM_HandleTypeDef htim, Pointeur:InfoBatterie infoBatterie
//// Sorties: bool (true si besoin de mesurer l'état de la batterie)
bool GestionCheckBatterie(ADC_HandleTypeDef *hadc, TIM_HandleTypeDef *htim, InfoBatterie *infoBatterie)
{
  if(infoBatterie->compteurCheckBatterie > 0)
  {
    infoBatterie->compteurCheckBatterie--; // Décrementer le compteur à chaque réveil
  }

  if(infoBatterie->compteurCheckBatterie == 0)
  {
    // Si le compteur vaut 0, il est temps de procéder à une nouvelle vérification de l'état de la batterie
    LancerCheckBatterie(htim, infoBatterie);
    return true;
  }
  else
  {
    return false; // Pas de vérification batterie nécessaire pour le moment
  }
}

//// Fonction MesurerEtatBatterie (détermine l'état de la batterie)
//// Description: utilise ControleEtatChargeBatterie pour récupérer l'état de la batterie et ArreterLectureTensionBatterie pour terminer l'opération
//// Entrées: Pointeur:ADC_HandleTypeDef hadc, Pointeur:TIM_HandleTypeDef htim, Pointeur:InfoBatterie infoBatterie
//// Sorties: Etat (état de la batterie)
Etat MesurerEtatBatterie(ADC_HandleTypeDef *hadc, TIM_HandleTypeDef *htim, InfoBatterie *infoBatterie)
{
  Etat nouvelEtatBatterie = ControleEtatChargeBatterie(hadc, infoBatterie);
  ArreterLectureTensionBatterie(htim, infoBatterie);
  return nouvelEtatBatterie;
}


```

```
53  //// Fonction LancerCheckBatterie (étape préliminaire avant de pouvoir mesurer l'état de la batterie)
54  //// Description: enclenche le transitor nécessaire pour pouvoir lire l'etat de la batterie, met en place le timer pour le délai avant mesure
55  //// Entrées: Pointeur:TIM_HandleTypeDef htim, Pointeur:InfoBatterie infoBatterie
56  //// Sorties: -
57  void LancerCheckBatterie(TIM_HandleTypeDef* htim, InfoBatterie* infoBatterie)
58  {
59    // Enclenche le transistor nécessaire pour pouvoir lire l'état de la batterie
60    HAL_GPIO_WritePin(EN_VBAT_GPIO_Port,EN_VBAT_Pin,GPIO_PIN_SET);
61
62    // Activation de l'interruption timer pour pouvoir attendre un certain délai (DELAI_ACTIVATION_TRANSISTOR_MS) avant de lire l'état de la batterie
63    HAL_TIM_Base_Start_IT(htim);
64
65    // Compteur cadencé sur l'interruption timer
66    // Nombre de fois que l'on doit décrémenter le compteur dans l'interruption timer avant de procéder à la mesure
67    infoBatterie->compteurCheckBatterie = DELAI_ACTIVATION_TRANSISTOR_MS / DELAI_TIMER6_MS;
68  }
69
70  //// Fonction ControleEtatChargeBatterie
71  //// Description: Contôle l'etat de charge de la batterie (allume une LED en cas de problème détecté)
72  //// Entrées: Pointeur:ADC_HandleTypeDef hadc, Pointeur:InfoBatterie infoBatterie
73  //// Sorties: Etat (état de la batterie)
74  Etat ControleEtatChargeBatterie(ADC_HandleTypeDef* hadc, InfoBatterie* infoBatterie)
75  {
76    float valeurBruteADC_VBAT = 0;
77    float valeurVBat = 0;
78
79    // Lecture de valeur du ADC et sauvegarde
80    valeurBruteADC_VBAT = LectureValeurAdcBrute(hadc, ADC_CHANNEL_6);
81
82    // Convertie la valeur brute de l'ADC en volt
83    valeurVBat = ConversionValeurAdcEnVolt(valeurBruteADC_VBAT, R3_THEORIQUE, R6_THEORIQUE);
84
85    Etat nouvelEtatBatterie = INDEFINI;
86
87    // Enbranchement en fonction de la tension des piles
88    // Pour changer la tension de la charge, modifier la valeur de VALEUR_BATTERIE_VOLT dans gestionBatterie.h
89    if (VALEUR_BATTERIE_VOLT == VALEUR_3PILES_VOLT)
90    {
91      if((valeurVBat > VALEUR_BATTERIE_4_5V_MAX_VOLT ) || (valeurVBat < VALEUR_BATTERIE_4_5V_MIN_VOLT ))
92      {
93        // Problème détecté (mauvaise tension de batterie ou batterie déchargée)
94        HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_SET); // Allume la LED rouge
95        nouvelEtatBatterie = ALARME;
96      }
97      else
98      {
99        // Aucun problème (batterie avec la bonne tension et chargée)
100       HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_RESET); // Eteint la LED rouge
101       nouvelEtatBatterie = OK;
102     }
103   }
104   else if (VALEUR_BATTERIE_VOLT == VALEUR_4PILES_VOLT)
105   {
```

```
106       if((valeurVBat  > VALEUR_BATTERIE_6V_MAX_VOLT )||(valeurVBat  < VALEUR_BATTERIE_6V_MIN_VOLT ))
107       {
108         // Problème détecté (mauvaise tension de batterie ou batterie déchargée)
109         HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_SET); // Allume la LED rouge
110         nouvelEtatBatterie = ALARME;
111       }
112       else
113       {
114         // Aucun problème (batterie avec la bonne tension et chargée)
115         HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_RESET); // Eteint la LED rouge
116         nouvelEtatBatterie = OK;
117       }
118     }
119
120     // Code de test qui change l'état de la batterie à chaque réveil
121     // Permet de tester l'activation de la LED et l'envoi des trames UART vers l'ESP
122     if(DEBUG_ALARME_BATTERIE)
123     {
124       if(infoBatterie->etatBatterie == ALARME) // Si l'état était ALARME on passe en OK
125       {
126         nouvelEtatBatterie = OK;
127         HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_RESET); // Eteint la LED rouge
128       }
129       else // Si l'état n'était pas ALARME on passe en ALARME
130       {
131         nouvelEtatBatterie = ALARME;
132         HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_SET); // Allume la LED rouge
133       }
134     }
135
136     return nouvelEtatBatterie;
137   }
138
139
140   //// Fonction ArreterLectureTensionBatterie
141   //// Description: Réinitialise le transitor utilisé pour lire l'etat de la batterie, défini le compteur pour le délai avant la prochaine
      vérification
142   //// Entrées: Pointeur:TIM_HandleTypeDef htim, Pointeur:InfoBatterie infoBatterie
143   //// Sorties: -
144   void ArreterLectureTensionBatterie(TIM_HandleTypeDef* htim, InfoBatterie* infoBatterie)
145   {
146     // Réinitialiser le transistor utilisé pour lire l'état de la batterie
147     HAL_GPIO_WritePin(EN_VBAT_GPIO_Port,EN_VBAT_Pin,GPIO_PIN_RESET);
148
149     // Arrêt de l'interruption timer utilisée pour attendre entre l'enclenchement du transistor et la mesure de l'état de la batterie
150     HAL_TIM_Base_Stop_IT(htim);
151
152     // Compteur cadencé sur la fréquence de réveil du STM
153     // Nombre de fois que le STM doit se réveiller avant de procéder à une nouvelle vérification de l'état de la batterie
154     infoBatterie->compteurCheckBatterie = DELAI_VERIFICATION_BATTERIE_MS / DELAI_ENTRE_DEUX_REVEILS_MS;
155   }
156
```