```c
/*!
 * @file        mode_charg.c
 *
 * @brief       This file contains all the functions for the charger PWM
 *
 * @version     V1.0
 *
 * @date        2023-11-04
 */
#include "mode_charg.h"
#include <stdlib.h>
#include "math.h"

extern Flag_Check          Flag;
extern uint16_t Prt_Battery12_24;

uint16_t num = 0;
int16_t Error = 0;
uint32_t Sum_Adc_Bat = 0;
uint16_t Bat_Tb = 0;
uint16_t AriMean_BAT = 0;
float _err_measure = 50;
float _err_estimate = 50;
float _q = 0.0009;
float _current_estimate = 0;
float _last_estimate = 0;
float _kalman_gain = 0;
uint16_t DutyMax = 10000;       // 10000 = 100% duty
uint8_t DelayUp = 0;            // xac dinh time dua tren delay bang systick
float Temp_Charger = 0.0;       // nhiet do trong qua trinh sac
extern uint16_t Save_Temp;      // luu gia tri nhiet do tam thoi

void En_PWM_CHG(void){
    IWDT_Refresh();
    CHG_BAT_DIS;
    LED1_OFF;
    LED3_ON;
    LED4_OFF;
    K= 3000;
    DutyMax = 10000;
    sysTick = 0;
    num = 0;
    Save_Temp = 0;
    DelayUp = 0;
    Flag.Charger = Flag_OFF;
    Delay_ms(3000);
    IWDT_Refresh();

    if(Re_Adc_PV > 3300 || Re_Adc_BAT > Prt_Battery12_24){   // PV = ~55V or BAT > 32V
        while(1){
```

```
51              Power_DIS;
52              if(Re_Adc_PV >= 3100){    // PV = ~50V
53                 while(1){
54                     LED2_ON;
55                     Delay_ms(500);
56                     LED2_OFF;
57                     Delay_ms(500);
58                     IWDT_Refresh();
59                 }
60              }
61              if(Re_Adc_BAT >= Prt_Battery12_24){    // BAT = 32V
62                 while(1){
63                     LED3_ON;
64                     Delay_ms(200);
65                     LED3_OFF;
66                     Delay_ms(200);
67                    IWDT_Refresh();
68                 }
69              }
70          }
71      }
72      CHG_BAT_EN;
73
74      while(1){
75          for(uint8_t j=0; j<=200; j++){
76              Sum_Adc_Bat += Re_Adc_BAT;
77          }
78          Bat_Tb = Sum_Adc_Bat/200;
79          Sum_Adc_Bat = 0;
80
81          AriMean_BAT = updateEstimate(Re_Adc_BAT);
82          Flag.Charger = Flag_ON;
83          Error = Point_Bulk - Re_Adc_BAT;
84          Cov_K = (double)Error / 8;
85          if(Re_Adc_BAT >= (Point_Bulk-2) && Re_Adc_BAT <= (Point_Bulk+2)){
86              K = K;
87          }else{
88              K += Cov_K;
89              if (K >= DutyMax){
90                  K = DutyMax;
91              } else if(K < 1500){
92                  while(1){
93                      AriMean_BAT = updateEstimate(Re_Adc_BAT);
94                      Error = Point_Float - AriMean_BAT;
95                      Cov_K = (double)Error / 10;
96                      if(AriMean_BAT >= (Point_Float-2) && AriMean_BAT <=
    (Point_Float+2)){
97                          K = K;
98                      }else{
99                          K += Cov_K;
```

```
100                         if (K >= DutyMax){
101                             K = DutyMax;
102                         } else if(K <= 500){
103                             while(1){
104                                 LED2_ON;
105                                 TMR_SetCompare1(TMR1, 0);
106                                 CHG_BAT_DIS;
107                                 if(sysTick >= 3000){
108                                     IWDT_Refresh();
109                                     sysTick = 0;
110                                     if(Re_Adc_PV <= SetPoin_DisPV){
111                                         LED2_OFF; return;
112                                     }
113                                     num++;
114                                 }
115                                 if(num >= 2000){    // ~10 minutes
116                                     num = 0;
117                                     K = 5000;
118                                     CHG_BAT_EN;
119                                     break;
120                                 }
121                             }
122                         }
123                     }
124                     TMR_SetCompare1(TMR1, K);
125                     if(Check_DIS_PV() == 1){ return;}
126                 }
127             }
128         }
129         TMR_SetCompare1(TMR1, K);
130         if(Check_DIS_PV() == 1){ return;}
131     }
132 }
133 uint8_t Check_DIS_PV(void){
134 
135     if((Temp_Charger = Re_TempNTC()) >= 110){
136         while(1){
137             TMR_SetCompare1(TMR1, 0);
138             CHG_BAT_DIS;
139             IWDT_Refresh();
140             if((Temp_Charger = Re_TempNTC()) <= 85){
141                 CHG_BAT_EN;
142                 sysTick = 0;
143                 break;
144             }
145             Delay_ms(1000);
146             LED2_ON;
147             LED3_ON;
148             Delay_ms(1000);
149             LED3_OFF;
```

```
150             LED2_OFF;
151         }
152     }
153     else if(Temp_Charger >= 100){
154         if((Temp_Charger - Save_Temp) >= 1 && DelayUp >= 20){
155             DutyMax -= 500;
156             DelayUp = 0;
157             Save_Temp = Temp_Charger;
158             if(DutyMax <= 0){DutyMax = 0;}
159         }
160     }
161     else
162        if(DelayUp >= 60){
163         DutyMax = 10000;
164         Save_Temp = 0;
165         DelayUp =0;
166     }
167 /****************************** Temperature ***************************************/
168 
169     if(sysTick >= 2500){
170         IWDT_Refresh();
171         CHG_BAT_DIS;
172         Delay_ms(40);
173         if(Re_Adc_PV <= SetPoin_DisPV){
174             CHG_BAT_DIS;
175             TMR_SetCompare1(TMR1, 0);
176             return 1;
177         }
178         CHG_BAT_EN;
179         for(uint8_t i = 0; i<5; i++){
180             LED2_ON;
181             Delay_ms(60);
182             LED2_OFF;
183             Delay_ms(60);
184         }
185         DelayUp++;
186         sysTick = 0;
187         IWDT_Refresh();
188     }
189     return 0;
190 }
191 float updateEstimate(float mea) {
192     _kalman_gain = _err_estimate / (_err_estimate + _err_measure);
193     _current_estimate = _last_estimate + _kalman_gain * (mea - _last_estimate);
194     _err_estimate = (1.0 - _kalman_gain) * _err_estimate
195             + fabs(_last_estimate - _current_estimate) * _q;
196     _last_estimate = _current_estimate;
197 
198     return _current_estimate;
199 }
```