```c
/*!
 * @file        main.c
 *
 * @brief       Main program body
 *
 * @version     V1.0.3
 *
 * @date        2022-09-20
 *
 * @attention
 */
/* Includes */
#include "main.h"

GPIO_Config_T       GPIO_ConfigStructure;
TMR_TimeBase_T      TMR_TimeBaseStructure;
TMR_OCConfig_T      OCConfigStructure;
ADC_Config_T        ADC_ConfigStructure;
DMA_Config_T        DMA_ConfigStructure;
EINT_Config_T       EINT_ConfigStructure;
USART_Config_T      USART_ConfigStructure;

Flag_Check          Flag;

uint32_t sysTick = 0;
#define BufSize 1
uint8_t DMA_USART_TxBuf[BufSize];
uint8_t DMA_USART_RxBuf[BufSize];

char Buffe_Re[100];
uint8_t x =0;

/******** ADC *********/
uint16_t Arr_Cov_Adc[6];
/******** Dimming *********/
volatile uint32_t Dimming = 0;
/******** FLASH *********/
uint32_t Addr = 0x08007000;
/*********Intialize_Parameter***********/
uint32_t Point_Float     = NULL;
uint32_t Point_Bulk      = NULL;
uint32_t SetPoin_DisPV   = NULL;
uint32_t Point_En_PV     = NULL;
uint32_t Recovery_Volt   = NULL;
uint32_t AC_SW_Volt      = NULL;
uint32_t Dis_BAT         = NULL;
uint32_t I_Load          = NULL;
uint16_t Prt_Battery12_24 = 3590;   // bao ve Battery > 16V/12V hoac 32V/24V
uint16_t Point_SW_ACDC    = 2300;   // Point xac dinh co nguon ACDC
uint16_t Flicker          = 5000;
```

```
51  /******** Available *********/
52  double Rt = 0.0;
53  float T = 0.0;
54  float Tc = 0.0;
55  uint16_t save_temp;
56
57  int main(void)
58  {
59       Config_RCC();
60       Config_GPIO();
61       Config_EINT();
62       Config_USART();
63       Config_TMR();
64       Config_OCTMR();
65       Config_ADC();
66       DMA_Disable(DMA1_CHANNEL_5);
67       Config_DMA();
68       USART_EnableDMA(USART1, USART_DMA_REQUEST_RX);
69       if(SysTick_Config(8000000 / 1000))
70      {
71          while(1);
72      }
73      IWDTInit();
74      Set_PID();
75      Load_OFF;
76      UPower_DC;
77      CHG_BAT_DIS;
78      TMR_SetCompare1(TMR1, 0);
79      TMR_SetCompare2(TMR3, 0);
80      Power_EN;
81      GPIOB->BSC = (1<<5); // Enable SET enters transmission/receive mode
82      GPIOB->BSC = (1<<4); // Enable CS enters working mode
83
84  /*********Parameter_Settings**********/
85      Point_Float    = *(volatile uint32_t*)Addr;
86      Point_Bulk     = *(volatile uint32_t*)(Addr+4);
87      SetPoin_DisPV  = *(volatile uint32_t*)(Addr+8);
88      Point_En_PV    = *(volatile uint32_t*)(Addr+12);
89
90      Recovery_Volt  = *(volatile uint32_t*)(Addr+16);
91      AC_SW_Volt     = *(volatile uint32_t*)(Addr+20);
92      Dis_BAT        = *(volatile uint32_t*)(Addr+24);
93      I_Load         = *(volatile uint32_t*)(Addr+28);
94
95  if( /*I_Load > 2250 || */ Dis_BAT > 4096 || AC_SW_Volt > 4096 || Recovery_Volt >
    4096 || Point_En_PV > 4096 || SetPoin_DisPV > 4096
96   || Point_Bulk > 4096 || Point_Float > 4096){
97      Set_Parameter(82);  // Reset parameter to default
98  }
99  /******************** Auto Set 12V/24V anh Protect Input Power ********************/
```

```
100     switch(Check_Bat12_24()){
101         case 0:
102             if( Dis_BAT > 1900){                            // use Dis_Bat de nhan biet
        truoc do set parameter cho power 12V hay 24V
103                 Point_Float = Point_Float/2;
104                 Point_Bulk = Point_Bulk/2;
105                 Recovery_Volt = Recovery_Volt/2;
106                 AC_SW_Volt = AC_SW_Volt/2;
107                 Dis_BAT = (Dis_BAT/2);
108                 I_Load = (((Val_CovCurr)/2.0)+2.055683594)/3*4096/3.3;
109                 Prt_Battery12_24 = Prt_Battery12_24/2;
110                 Point_SW_ACDC = Point_SW_ACDC/2;
111                 Flicker = 2500;
112             } break;
113         case 1:
114             if( Dis_BAT < 1900){
115                 Point_Float = Point_Float*2;
116                 Point_Bulk = Point_Bulk*2;
117                 Recovery_Volt = Recovery_Volt*2;
118                 AC_SW_Volt = AC_SW_Volt*2;
119                 Dis_BAT = Dis_BAT*2;
120                 I_Load = (((Val_CovCurr)*2.0)+2.055683594)/3*4096/3.3;
121                 Prt_Battery12_24 = Prt_Battery12_24*2;
122                 Point_SW_ACDC = Point_SW_ACDC*2;
123                 Flicker = 5000;
124             } break;
125         default: break;
126         }
127     if(Re_Adc_BAT > Prt_Battery12_24){    // BAT >= ~16V/32V
128        while(1){
129             Power_DIS;
130             LED3_ON;
131             Delay_ms(200);
132             LED3_OFF;
133             Delay_ms(200);
134             IWDT_Refresh();
135        }
136     }
137     /*************************************************************************************
        /
138
139         FMC_Unlock();
140         FMC_ErasePage(Addr);
141         for(uint32_t num=0;num<=28;num+=4){
142             switch(num){
143                 case 0:
144                     FMC_ProgramWord(Addr+num, Point_Float);
145                     break;
146                 case 4:
147                     FMC_ProgramWord(Addr+num, Point_Bulk);
```

```
148                 break;
149             case 8:
150                 FMC_ProgramWord(Addr+num, SetPoin_DisPV);
151                 break;
152             case 12:
153                 FMC_ProgramWord(Addr+num, Point_En_PV);
154                 break;
155             case 16:
156                 FMC_ProgramWord(Addr+num, Recovery_Volt);
157                 break;
158             case 20:
159                 FMC_ProgramWord(Addr+num, AC_SW_Volt);
160                 break;
161             case 24:
162                  FMC_ProgramWord(Addr+num, Dis_BAT);
163                 break;
164             case 28:
165                  FMC_ProgramWord(Addr+num, I_Load);
166         }
167     }
168     FMC_Lock();
169 
170 /*********************End-Flash*********************/
171     Delay_ms(1000);
172     IWDT_Refresh();
173     LED_ALL_OFF;
174     StartUp_LED();
175     IWDT_Refresh();
176 
177     /* USART1 Send data to PC, and you need to open serial assistant to observe */
178     USART_Write(USART1, "\n\rDRIVER HYBRID V2.1\r\n");
179 
180     Delay_ms(1);
181 
182     USART_Write(USART1,"\nModule RF M-BK2461U 2.4Ghz\r\n");
183 
184     printf("\n\n\n\rParameter:");
185     printf("\n\rBattery Cut-Off voltage: %.2fV", (Dis_BAT/Val_CovBat_DC));
186     printf("\n\rDC to AC Voltage: %.2fV", AC_SW_Volt/Val_CovBat_DC);
187     printf("\n\rCurrent: %.2fA", Val_CovCurr);
188     printf("\n\rRecovery_Volt: %.2fV", Recovery_Volt/Val_CovBat_DC);
189 
190     Flag.Cov_ACDC = Flag_OFF; // Set status Flag convert ACDC
191     Flag.Cov_ACDC = Flag_OFF;
192 
193     while (1)
194     {
195         switch(Ctrl_Program()){
196 
197             case 0:                     // Mode Charger
```

```
198                 En_PWM_CHG();
199                 if(Re_Adc_BAT >= Recovery_Volt){    // Recovery Voltage
200                     Flag.Cov_ACDC = Flag_OFF;
201                 }
202                 break;
203             case 1:                         // Use PIN Power
204                 UPower_Pin;
205                 LED3_ON;
206                 LED4_OFF;
207                 Load_Enable();
208                 LED3_OFF;
209                 break;
210             case 2:                         // Use DC Power
211                 UPower_DC;
212                 LED4_ON;
213                 LED3_OFF;
214                 Load_Enable();
215                 LED4_OFF;
216                 break;
217 
218             case 3:                         // PIN < 22V => Disable system
219                 while(1){
220                     IWDT_Refresh();
221                     Load_OFF;
222                     TMR_SetCompare2(TMR3, 0);
223                     if(Ctrl_Program() == 0){
224                         En_PWM_CHG();
225                         break;
226                     }
227                     else if (Ctrl_Program() == 2){
228                         break;
229                     }
230 
231                     GPIOA->ODATA ^= GPIO_PIN_15;
232                     Delay_ms(1000);
233                 } break;
234             default:GPIOA->ODATA ^= GPIO_PIN_15;
235                 break;
236         }
237     }
238 }
239 
240 static void Config_RCC(void) {
241     RCM_EnableAHBPeriphClock(RCM_AHB_PERIPH_GPIOA | RCM_AHB_PERIPH_DMA1 |
    RCM_AHB_PERIPH_GPIOB);
242     RCM_EnableAPB2PeriphClock(RCM_APB2_PERIPH_TMR1 | RCM_APB2_PERIPH_ADC1 |
    RCM_APB2_PERIPH_SYSCFG | RCM_APB2_PERIPH_USART1);
243     RCM_EnableAPB1PeriphClock(RCM_APB1_PERIPH_TMR3 | RCM_APB1_PERIPH_TMR14);
244 }
245 static void Config_GPIO(void){
```

```
246
247     // Config Pin control PWM PA8, PA9 - Power PA10
248     GPIO_ConfigStructure.pin = GPIO_PIN_3 | GPIO_PIN_6 | GPIO_PIN_9 | GPIO_PIN_10 |
    GPIO_PIN_11 | GPIO_PIN_12 | GPIO_PIN_15;
249     GPIO_ConfigStructure.mode = GPIO_MODE_OUT;
250     GPIO_ConfigStructure.outtype = GPIO_OUT_TYPE_PP;
251     GPIO_ConfigStructure.speed = GPIO_SPEED_50MHz;
252     GPIO_Config(GPIOA, &GPIO_ConfigStructure);
253
254     // Led4, CS-module RF
255     GPIO_ConfigStructure.pin = GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5;
256     GPIO_Config(GPIOB, &GPIO_ConfigStructure);
257
258     // PWM Boost
259     GPIO_ConfigStructure.pin = GPIO_PIN_7 | GPIO_PIN_8;
260     GPIO_ConfigStructure.mode = GPIO_MODE_AF;
261     GPIO_ConfigStructure.outtype = GPIO_OUT_TYPE_PP;
262     GPIO_ConfigStructure.speed = GPIO_SPEED_50MHz;
263     GPIO_Config(GPIOA, &GPIO_ConfigStructure);
264
265     GPIO_ConfigPinAF(GPIOA, GPIO_PIN_SOURCE_7, GPIO_AF_PIN1);
266     GPIO_ConfigPinAF(GPIOA, GPIO_PIN_SOURCE_8, GPIO_AF_PIN2);
267
268     // USART_RF
269     GPIO_ConfigStructure.pin = GPIO_PIN_6 | GPIO_PIN_7;
270     GPIO_ConfigStructure.pupd = GPIO_PUPD_PU;
271     GPIO_Config(GPIOB, &GPIO_ConfigStructure);
272
273     GPIO_ConfigPinAF(GPIOB, GPIO_PIN_SOURCE_6, GPIO_AF_PIN0);
274     GPIO_ConfigPinAF(GPIOB, GPIO_PIN_SOURCE_7, GPIO_AF_PIN0);
275
276     // Config Pin ADC
277     GPIO_ConfigStructure.pin = GPIO_PIN_0 | GPIO_PIN_2 | GPIO_PIN_4 |
    GPIO_PIN_5;          // PA0 = ADC_LED, PA2 = ADC_DC, PA4 = ADC_BAT
278     GPIO_ConfigStructure.mode = GPIO_MODE_AN;
279     GPIO_Config(GPIOA, &GPIO_ConfigStructure);
280
281     GPIO_ConfigStructure.pin = GPIO_PIN_0 | GPIO_PIN_1;          // Read Current
282     GPIO_ConfigStructure.mode = GPIO_MODE_AN;
283     GPIO_Config(GPIOB, &GPIO_ConfigStructure);
284
285     // EINT PA1 Configuration
286     GPIO_ConfigStructure.pin = GPIO_PIN_1;
287     GPIO_ConfigStructure.mode = GPIO_MODE_IN;
288     GPIO_ConfigStructure.pupd = GPIO_PUPD_PU;
289     GPIO_Config(GPIOA, &GPIO_ConfigStructure);
290
291 }
292 static void Config_EINT(void){
293
```

```
294     SYSCFG_EINTLine(SYSCFG_PORT_GPIOA, SYSCFG_PIN_1);
295     EINT_ClearStatusFlag(EINT_LINE1);
296 
297     EINT_ConfigStructure.line = EINT_LINE1;
298     EINT_ConfigStructure.lineCmd = ENABLE;
299     EINT_ConfigStructure.mode = EINT_MODE_INTERRUPT;
300     EINT_ConfigStructure.trigger = EINT_TRIGGER_RISING;
301     EINT_Config(&EINT_ConfigStructure);
302 
303     NVIC_EnableIRQRequest(EINT0_1_IRQn, 0);
304 }
305 static void Config_USART(void){
306 
307     /* BaudRate = 9600 baud */
308     USART_ConfigStructure.baudRate = 19200;
309     /* Receive and transmit enabled */
310     USART_ConfigStructure.mode     = USART_MODE_TX_RX;
311     /* Hardware flow control disabled (RTS and CTS signals) */
312     USART_ConfigStructure.hardwareFlowCtrl = USART_FLOW_CTRL_NONE;
313     /* No parity */
314     USART_ConfigStructure.parity   = USART_PARITY_NONE;
315     /* One Stop Bit */
316     USART_ConfigStructure.stopBits =  USART_STOP_BIT_1;
317     /* Word Length = 8 Bits */
318     USART_ConfigStructure.wordLength = USART_WORD_LEN_8B;
319     /* USART_Config */
320     USART_Config(USART1, &USART_ConfigStructure);
321 
322     /* Enable USART_Interrupt_RXBNEIE */
323     USART_EnableInterrupt(USART1, USART_INT_RXBNEIE);
324 
325     NVIC_EnableIRQRequest(USART1_IRQn, 3);
326 
327     /* Enable USART */
328     USART_Enable(USART1);
329     /* Remap USART1_RX Channel to DMA channel 5*/
330     SYSCFG_EnableDMAChannelRemap(SYSCFG_DAM_REMAP_USART1RX);
331 }
332 
333 static void Config_TMR(void){
334     TMR_TimeBaseStructure.clockDivision =  TMR_CKD_DIV1;
335     TMR_TimeBaseStructure.counterMode =  TMR_COUNTER_MODE_UP;
336     // PWM Charger
337     TMR_TimeBaseStructure.div = 26;
338     TMR_TimeBaseStructure.period = 9999;
339     TMR_TimeBaseStructure.repetitionCounter = 0;
340     TMR_ConfigTimeBase(TMR1, &TMR_TimeBaseStructure);
341     // PWM Discharger
342     TMR_TimeBaseStructure.div = 0;
343     TMR_TimeBaseStructure.period = 160;
```

```
344     TMR_ConfigTimeBase(TMR3, &TMR_TimeBaseStructure);
345     // Timer Dimming
346     TMR_TimeBaseStructure.div = 799;
347     TMR_TimeBaseStructure.period = 9999;
348     TMR_ConfigTimeBase(TMR14, &TMR_TimeBaseStructure);
349 
350     TMR_ClearIntFlag(TMR14, TMR_INT_FLAG_UPDATE);
351     TMR_EnableInterrupt(TMR14, TMR_INT_UPDATE);
352     NVIC_EnableIRQRequest(TMR14_IRQn, 4);
353 }
354 
355 static void Config_OCTMR(void){
356     OCConfigStructure.OC_Mode = TMR_OC_MODE_PWM1;
357     OCConfigStructure.OC_Polarity = TMR_OC_POLARITY_HIGH;
358     OCConfigStructure.OC_OutputState = TMR_OUTPUT_STATE_ENABLE;
359     OCConfigStructure.Pulse = 0;
360     TMR_OC1Config(TMR1, &OCConfigStructure);
361     TMR_OC2Config(TMR3, &OCConfigStructure);
362 
363     TMR_EnablePWMOutputs(TMR3);
364     TMR_EnablePWMOutputs(TMR1);
365 
366     TMR_Enable(TMR1);
367     TMR_Enable(TMR3);
368     TMR_Enable(TMR14);
369 }
370 
371 static void Config_ADC(void){
372 
373     ADC_Reset();
374 
375     ADC_ConfigStructure.resolution = ADC_RESOLUTION_12B;
376     ADC_ConfigStructure.convMode = ADC_CONVERSION_CONTINUOUS;
377     ADC_ConfigStructure.scanDir = ADC_SCAN_DIR_UPWARD;
378     ADC_ConfigStructure.extTrigConv = ADC_EXT_TRIG_CONV_TRG0;
379     ADC_ConfigStructure.extTrigEdge = ADC_EXT_TRIG_EDGE_NONE;
380     ADC_ConfigStructure.dataAlign = ADC_DATA_ALIGN_RIGHT;
381     ADC_Config(&ADC_ConfigStructure);
382 
383     ADC_ConfigChannel(ADC_CHANNEL_0, ADC_SAMPLE_TIME_239_5); // Read Voltage LED
384     ADC_ConfigChannel(ADC_CHANNEL_2, ADC_SAMPLE_TIME_239_5); // Read Voltage Power DC
385     ADC_ConfigChannel(ADC_CHANNEL_4, ADC_SAMPLE_TIME_239_5); // Read Voltage Power
    Battery
386     ADC_ConfigChannel(ADC_CHANNEL_5, ADC_SAMPLE_TIME_239_5); // Read Voltage PV
387     ADC_ConfigChannel(ADC_CHANNEL_8, ADC_SAMPLE_TIME_239_5); // Read Current LED
388     ADC_ConfigChannel(ADC_CHANNEL_9, ADC_SAMPLE_TIME_239_5); // Sensor Temp
389 
390     ADC_ReadCalibrationFactor();
391     ADC_DMARequestMode(ADC_DMA_MODE_CIRCULAR);
392     ADC_EnableDMA();
```

```
393     ADC_Enable();
394     while (!ADC_ReadStatusFlag(ADC_FLAG_ADRDY))
395     ;
396     ADC_StartConversion();
397 }
398 
399 static void Config_DMA(void){
400     /**********************DMA_ADC - Init***************************/
401     DMA_ConfigStructure.peripheralAddress = (uint32_t)ADC1_DR_Address;
402     DMA_ConfigStructure.memoryAddress = (uint32_t)&Arr_Cov_Adc;
403     DMA_ConfigStructure.direction = DMA_DIR_PERIPHERAL;
404     DMA_ConfigStructure.bufferSize = 6;
405     DMA_ConfigStructure.peripheralInc = DMA_PERIPHERAL_INC_DISABLE;
406     DMA_ConfigStructure.memoryInc = DMA_MEMORY_INC_ENABLE;
407     DMA_ConfigStructure.peripheralDataSize = DMA_PERIPHERAL_DATASIZE_HALFWORD;
408     DMA_ConfigStructure.memoryDataSize = DMA_MEMORY_DATASIZE_HALFWORD;
409     DMA_ConfigStructure.circular = DMA_CIRCULAR_ENABLE;
410     DMA_ConfigStructure.priority = DMA_PRIORITY_LEVEL_HIGHT;
411     DMA_ConfigStructure.memoryTomemory = DMA_M2M_DISABLE;
412 
413     DMA_Config(DMA1_CHANNEL_1, &DMA_ConfigStructure);
414     DMA_Enable(DMA1_CHANNEL_1);
415 
416     /**********************DMA_USART - Init***************************/
417     DMA_ConfigStructure.peripheralAddress = (uint32_t)&USART1->RXDATA;
418     DMA_ConfigStructure.memoryAddress = (uint32_t)&DMA_USART_RxBuf;
419     DMA_ConfigStructure.direction = DMA_DIR_PERIPHERAL;
420     DMA_ConfigStructure.bufferSize = BufSize;
421     DMA_ConfigStructure.peripheralInc = DMA_PERIPHERAL_INC_DISABLE;
422     DMA_ConfigStructure.memoryInc = DMA_MEMORY_INC_ENABLE;
423     DMA_ConfigStructure.peripheralDataSize = DMA_PERIPHERAL_DATASIZE_BYTE;
424     DMA_ConfigStructure.memoryDataSize = DMA_MEMORY_DATASIZE_BYTE;
425     DMA_ConfigStructure.circular = DMA_CIRCULAR_ENABLE;
426     DMA_ConfigStructure.priority = DMA_PRIORITY_LEVEL_HIGHT;
427     DMA_ConfigStructure.memoryTomemory = DMA_M2M_DISABLE;
428 
429     DMA_Config(DMA1_CHANNEL_5, &DMA_ConfigStructure);
430     DMA_ClearIntFlag(DMA1_INT_FLAG_TF5);
431     DMA_EnableInterrupt(DMA1_CHANNEL_5, DMA_INT_TFIE);
432     NVIC_EnableIRQRequest(DMA1_CH4_5_IRQn, 2);
433     USART_ClearStatusFlag(USART1, USART_FLAG_TXC);
434     DMA_Enable(DMA1_CHANNEL_5);
435 }
436 
437 static void IWDTInit(void){
438     /* clear IWDTRST Flag*/
439     if (RCM_ReadStatusFlag(RCM_FLAG_IWDTRST) != RESET)
440     {
441         RCM_ClearStatusFlag();
442     }
```

```
443     /* set IWDT Write Access */
444     IWDT_EnableWriteAccess();
445 
446     /* set IWDT Divider*/
447     IWDT_ConfigDivider(IWDT_DIV_64); //1.6ms - 6553.6ms
448 
449     /* set IWDT Reloader*/
450     IWDT_ConfigReload(2187); // Reloader() = ((timeout(s)*LSI(40K))/PSC)-1 <=> 3.5008s
451 
452     /* Refresh*/
453     IWDT_Refresh();
454 
455     /* Enable IWDT*/
456     IWDT_Enable();
457 }
458 void DMA_Isr(void)
459 {
460     if (DMA_ReadStatusFlag(DMA1_FLAG_TF5))
461     {
462         /* do something*/
463         DMA_ClearStatusFlag(DMA1_FLAG_TF5);
464 
465         for (int i = 0; i < BufSize; i++)
466         {
467             /* USART send data*/
468             while (USART_ReadStatusFlag(USART1, USART_FLAG_TXBE) == RESET);
469             DMA_USART_TxBuf[i] = DMA_USART_RxBuf[i];
470 
471 
472             Buffe_Re[x++] = DMA_USART_RxBuf[i];
473             if(x==100){ x=0;}
474 
475             if(DMA_USART_RxBuf[i] == 95){
476                Set_Parameter(Buffe_Re[x-2]);
477             }
478         }
479     }
480 }
481 void EINT_Protect(void){
482 
483     if(EINT_ReadStatusFlag(EINT_LINE1) == SET){
484         if(GPIO_ReadInputBit(GPIOA, GPIO_PIN_1) == 1){
485 
486             UPower_DC;
487             TMR_SetCompare2(TMR3, 0);
488             Load_OFF;
489 
490             for(uint8_t delay=0; delay < 50; delay++){
491                 LED_ALL_OFF;
492                 Delay(0x5ffff);
```

```
                LED_ALL_ON;
                Delay(0x5ffff);
                IWDT_Refresh();
            }
        }
        EINT_ClearStatusFlag(EINT_LINE1);
        NVIC_SystemReset();
    }
}
void TMR14Isr(void)
{
    if (TMR_ReadIntFlag(TMR14, TMR_INT_FLAG_UPDATE) == SET)
    {
        TMR_ClearIntFlag(TMR14, TMR_INT_FLAG_UPDATE);

        Dimming++;
    }
}
void Delay(uint32_t val)
{
    volatile uint32_t delay = val;

    while (delay--);
}

void Delay_ms(uint16_t time){
    sysTick = 0;
    while(sysTick <= time)
    {;}
}
float Re_TempNTC(void){

    Rt = ((4096/(float)Re_Temp)-1);
    T = 1/((1.00/298.15)-(1.00/3470.00)*(log(Rt)));
    Tc  = T-273.15;
    return Tc;
}
uint8_t Check_Bat12_24(void){
    if(Re_Adc_BAT >= 900 && Re_Adc_BAT <= 1910){    // 8V < Battery < 17V   =>
Battery 12V
        return 0;
    }
    else if( Re_Adc_BAT > 1910){                    // Battery > 17V => Battery 24V
        return 1;
    }
    return 2;       // Battery Cut-off BMS => parameter do not change
}
```