```c
/*!
 * @file        RF_usart.h
 *
 * @brief       This file contains all the functions for the charger PWM
 *
 * @version     V1.0
 *
 * @date        2023-12-24
 */

#include "RF_usart.h"
#include "main.h"

extern Flag_Check          Flag;
extern uint32_t Addr;
extern uint16_t Curr_Def;
uint16_t Prt_CurrMax = 2300;
float Pload = 0;

void Set_Parameter(uint8_t num_rf){
    if(USART_ReadStatusFlag(USART1, USART_FLAG_TXBE) != RESET){
        if(num_rf == 73 || Flag.Set_Curr == 1){       // I -> set current
            Flag.Set_Curr = Flag_ON;
            Flag.Set_AC_SW = Flag_OFF;
            Flag.Set_Dis_BAT = Flag_OFF;
            Set_Current(num_rf);
            Flash_Write();
        }
        else if(num_rf == 83 || Flag.Set_AC_SW == 1){     // S -> Switch AC-DC
            Flag.Set_Curr = Flag_OFF;
            Flag.Set_AC_SW = Flag_ON;
            Flag.Set_Dis_BAT = Flag_OFF;

            Set_ACDC(num_rf);
            Flash_Write();
        }
        else if(num_rf == 68 || Flag.Set_Dis_BAT == 1){        // D -> Cut-Off Battery
            Flag.Set_Curr = Flag_OFF;
            Flag.Set_AC_SW = Flag_OFF;
            Flag.Set_Dis_BAT = Flag_ON;

            Set_DisBattery(num_rf);
            Flash_Write();
        }
        else if(num_rf == 80){      // P -> check parameter
            Check_Para();
        }
        else if(num_rf == 82){      // R -> Reset parameter to default
            Rst_Default();
        }
        else if(num_rf == 81){       // Q-> Read data for Application
            Read_Para();
        }
        else {
            Flag.Set_Curr = Flag_OFF;
            Flag.Set_AC_SW = Flag_OFF;
            Flag.Set_Dis_BAT = Flag_OFF;
            printf("\n\rError! Import: I, S, D, P");
        }
    }
}
void Set_Current(uint8_t Curr){
    if(Curr == 73){
        printf("\n\rSet Current: import '+' or '-'");
        printf("\n\rCurrent: %.2fA", Val_CovCurr);      // Current <=> quy đổi ra điện áp sau cầu phân áp * 3 lần trừ đi hệ số tính toán
    }
    else if(Curr == 0x2d){
        I_Load -= 50;
        Curr_Def -= 50;
        if(I_Load <= 0){I_Load = 0; Curr_Def = 0;}
        printf("\n\rCurrent: %.2fA", Val_CovCurr);
        printf("@D%.2f#",   Val_CovCurr);

    }
    else if(Curr == 0x2b){
        I_Load += 50;
        Curr_Def += 50;
        if(I_Load >= Prt_CurrMax){I_Load = Prt_CurrMax; Curr_Def = Prt_CurrMax;}
        printf("\n\rCurrent: %.2fA", Val_CovCurr);
        printf("@D%.2f#",   Val_CovCurr);
    }
    else if(Curr == 66){
        Flag.Set_Curr = Flag_OFF;
        printf("\n\rReset OK!");
    }
    else{
        printf("\n\rFail; import 'B' -> Reset");
    }
}
void Set_ACDC(uint8_t Sw){
    if(Sw == 83){
        printf("\n\rSet DC to AC Voltage: import '+' or '-'");
        printf("\n\rDC to AC Voltage: %.2fV", AC_SW_Volt/Val_CovBat_DC);
    }
    else if(Sw == 0x2d){
        AC_SW_Volt -= 10;
        printf("\n\rDC to AC Voltage: %.2fV", (AC_SW_Volt/Val_CovBat_DC));
        Flag.Cov_ACDC = Flag_OFF;      // Reset Recovery flag
    }
    else if(Sw == 0x2b){
        AC_SW_Volt += 10;
        printf("\n\rDC to AC Voltage: %.2fV", AC_SW_Volt/Val_CovBat_DC);
        Flag.Cov_ACDC = Flag_OFF;      // Reset Recovery flag
    }
    else if(Sw == 66){
        Flag.Set_AC_SW = Flag_OFF;
        printf("\n\rReset OK!");
    }
    else{
        printf("\n\rFail; import 'B' -> Reset");
    }
}
void Set_DisBattery(uint8_t Dis_Bat){
    if(Dis_Bat == 68){
        printf("\n\rSet Battery Cut-Off voltage: import '+' or '-'");
        printf("\n\rBattery Cut-Off voltage: %.2fV", (Dis_BAT/Val_CovBat_DC));

    }
    else if(Dis_Bat == 0x2d){
        Dis_BAT -= 10;
        printf("\n\rBattery Cut-Off voltage: %.2fV", (Dis_BAT/Val_CovBat_DC));
    }
    else if(Dis_Bat == 0x2b){
        Dis_BAT += 10;
        printf("\n\rBattery Cut-Off voltage: %.2fV", Dis_BAT/Val_CovBat_DC);
    }
    else if(Dis_Bat == 66){
        Flag.Set_Dis_BAT = Flag_OFF;
        printf("\n\rReset OK!");
    }
    else{
        printf("\n\rFail; import 'B' -> Reset");
    }
}
void Check_Para(void){
    Pload = (Val_CovCurr)*(Re_Adc_LED/Val_CovLoad);
    printf("\n\rParameter:");
    printf("\n\rBattery Cut-Off voltage: %.2fV", (Dis_BAT/Val_CovBat_DC));
    printf("\n\rDC to AC Voltage: %.2fV", AC_SW_Volt/Val_CovBat_DC);
    printf("\n\rCurrent: %.2fA", Val_CovCurr);
    printf("\n\rRecovery_Volt: %.2fV", Recovery_Volt/Val_CovBat_DC);
    printf("\n\rTemp: %.2foC", Re_TempNTC());
    printf("\n\rVoltage PV: %.2fV", Re_Adc_PV/1.0);
    printf("\n\rVoltage LED: %.2fV", Re_Adc_LED/Val_CovLoad);
    printf("\n\rVoltage ACDC: %.2fV", Re_Adc_DC/Val_CovBat_DC);
    printf("\n\rVoltage Battery: %.2fV", Re_Adc_BAT/Val_CovBat_DC);
    printf("\n\rCong suat LOAD: %.2fV", Pload);
}
void Read_Para(void){
    Pload = (Val_CovCurr)*(Re_Adc_LED/Val_CovLoad);
    printf("@B%.2f#\n", (Dis_BAT/Val_CovBat_DC));
    printf("@C%.2f#\n",  AC_SW_Volt/Val_CovBat_DC);
    printf("@D%.2f#\n",  Val_CovCurr);
    printf("@E%.2f#\n", Recovery_Volt/Val_CovBat_DC);
    printf("@U%.2f#\n", Re_TempNTC());
    printf("@I%.2f#\n", Re_Adc_PV/1.0);
    printf("@O%.2f#\n", Re_Adc_LED/Val_CovLoad);
    printf("@T%.2f#\n", Re_Adc_DC/Val_CovBat_DC);
    printf("@X%.2f#\n", Re_Adc_BAT/Val_CovBat_DC);
    printf("@Y%.2f#\n", Pload);
    Delay(0x2fff);
}
void Rst_Default(void){
    switch(Check_Bat12_24()){
        case 0:
            FMC_Unlock();
            FMC_ErasePage(Addr);
            FMC_ProgramWord(Flash_Float, 3040/2);     // default 27V (2700 = 24V)
            FMC_ProgramWord(Flash_Bulk, 3200/2);      // default 28.5V = 3200 (2810 = 25V)
            FMC_ProgramWord(Flash_DisPv, 555);      // PV = ~9V
            FMC_ProgramWord(Flash_EnPv, 620);       // PV  = ~10V

            FMC_ProgramWord(Flash_ReVol, 2810/2);     // BAT = 25V (Recovery Voltage)
            FMC_ProgramWord(Flash_ACDC, 2585/2);      // BAT = 23V (Set point Swiching DC-AC)
            FMC_ProgramWord(Flash_DisBat, 2475/2);    // BAT < 22V ( Disable PIN)
            FMC_ProgramWord(Flash_Curr, 1310);      // Current Load I_load = 2A(1700) - 24V/5A/120W-Load/3.24A/35.3V(2180)
            Flicker = 3000;
            FMC_Lock();
            break;
        case 1:
            FMC_Unlock();
            FMC_ErasePage(Addr);
            FMC_ProgramWord(Flash_Float, 3040);     // default 27V (2700 = 24V)
            FMC_ProgramWord(Flash_Bulk, 3200);      // default 28.5V = 3200 (2810 = 25V)
            FMC_ProgramWord(Flash_DisPv, 555);      // PV = ~9V
            FMC_ProgramWord(Flash_EnPv, 620);       // PV  = ~10V

            FMC_ProgramWord(Flash_ReVol, 2810);     // BAT = 25V (Recovery Voltage)
            FMC_ProgramWord(Flash_ACDC, 2585);      // BAT = 23V (Set point Swiching DC-AC)
            FMC_ProgramWord(Flash_DisBat, 2475);    // BAT < 22V ( Disable PIN)
            FMC_ProgramWord(Flash_Curr, 1700);      // Current Load I_load = 2A(1700) - 24V/5A/120W-Load/3.24A/35.3V(2180)
            Flicker = 5000;
            FMC_Lock();
            break;
        default:
            break;
    }
    Point_Float    = *(volatile uint32_t*)Addr;
    Point_Bulk     = *(volatile uint32_t*)(Addr+4);
    SetPoin_DisPV  = *(volatile uint32_t*)(Addr+8);
    Point_En_PV    = *(volatile uint32_t*)(Addr+12);

    Recovery_Volt  = *(volatile uint32_t*)(Addr+16);
    AC_SW_Volt     = *(volatile uint32_t*)(Addr+20);
    Dis_BAT        = *(volatile uint32_t*)(Addr+24);
    I_Load         = *(volatile uint32_t*)(Addr+28);
    printf("\n\rReset parameter to default: OK!");
}
void Flash_Write(void){
    FMC_Unlock();
    FMC_ErasePage(Addr);
    FMC_ProgramWord(Flash_Float, Point_Float);
    FMC_ProgramWord(Flash_Bulk, Point_Bulk);
    FMC_ProgramWord(Flash_DisPv, SetPoin_DisPV);
    FMC_ProgramWord(Flash_EnPv, Point_En_PV);
    FMC_ProgramWord(Flash_ReVol, Recovery_Volt);
    FMC_ProgramWord(Flash_ACDC, AC_SW_Volt);
    FMC_ProgramWord(Flash_DisBat, Dis_BAT);
    FMC_ProgramWord(Flash_Curr, I_Load);
    FMC_Lock();
}
void USART_Write(USART_T* usart, uint8_t* dat)
{
    while (*dat)
    {
        while (USART_ReadStatusFlag(usart, USART_FLAG_TXBE) == RESET);
        USART_TxData(usart, *dat++);
    }
}
#if defined (__CC_ARM) || defined (__ICCARM__) || (defined(__ARMCC_VERSION) && (__ARMCC_VERSION >= 6010050))

int fputc(int ch, FILE* f)
{
    /* send a byte of data to the serial port */
    USART_TxData(DEBUG_USART, (uint8_t)ch);
    /* wait for the data to be send  */
    while (USART_ReadStatusFlag(DEBUG_USART, USART_FLAG_TXBE) == RESET);
    return (ch);
}
#elif defined (__GNUC__)
#endif

```