**项目开发基本配置说明**

编制：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

审核：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

批准：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**修订记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 版本 | 修订日期 | 修订内容 | 修订者 |
| 1 | V1.0 | 2018-01-20 | 初版 | XXX |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[一、 calibrate.json的配置说明 3](#_Toc504723594)

[二、 config.js配置说明 6](#_Toc504723595)

[三、 ChargerComm\_LCfg.c配置说明 11](#_Toc504723596)

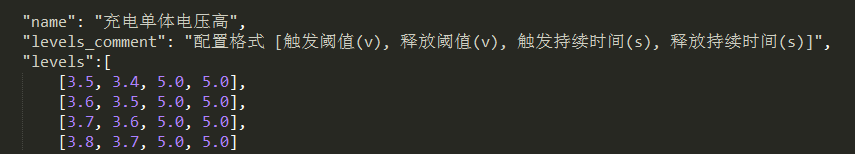
[四、 VcuComm\_LCfg.c配置说明 12](#_Toc504723597)

[五、 SocCalib\_Lcfg.c配置说明 13](#_Toc504723598)

[六、 SocOcvCalib\_Lcfg.c配置说明 14](#_Toc504723599)

[七、 PowerM\_Lcfg.c配置说明 15](#_Toc504723600)

# calibrate.json的配置说明



故障等级配置，以及触发时间配置。配置可以按照协议进行配置，需要注意的是无效值的配置。放电单体高三级应该比充电单体高三级高一些，充电单体低三级应该比放电低三级还要低一些。

单体电压无效值为 65.535

电流无效值为 0

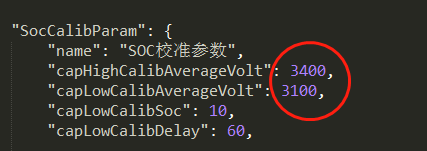
总压无效值为 6553.5

温度无效值为 205

若想读取这些配置值，则通过rte层函数

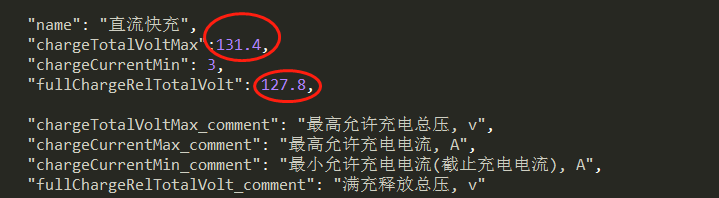
uint16 ParameterM\_DiagCalibRead(Diagnosis\_ItemType index, uint8 level, uint8 offset);来获取，其中Diagnosis\_ItemType为枚举类型

在rte层 Diagnosis\_Cfg.h中可以查看枚举成员



关于SOC满校准和空校准的电压配置参数，模板默认为磷酸铁锂

三元材料的参数为 满校准3900 空校准3400



最高允许充电电压为前面参数的值，释放值为(单体电压 -500mv)\*电池串数

这里的最小电流设置一定要注意要比充电末端降流的值要大，要不SOC标定不了100%

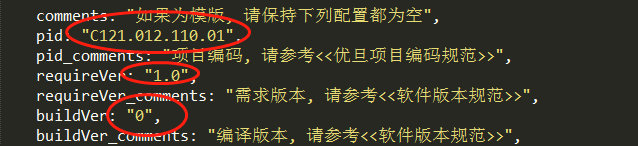
协议需要充电加热功能的，需要在下图中配置



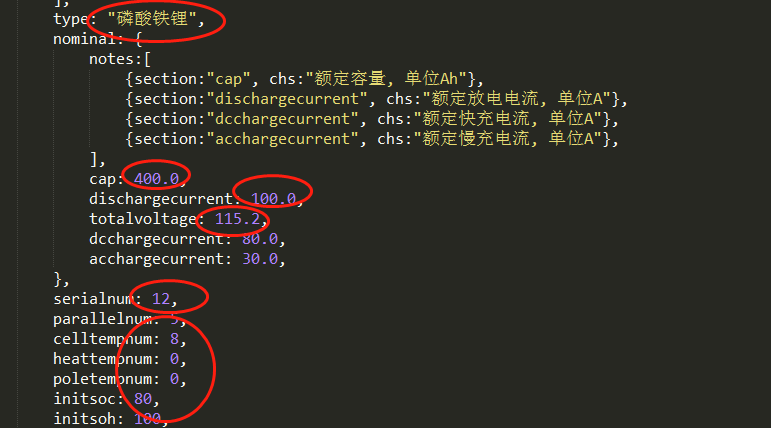
红字部分一般需要修改，在程序中若要读这些值可以通过函数

ParameterM\_CalibRead(PARAMETERM\_GET\_APP\_CALIB\_PARA\_ADDR(TemperatureMPara.chgHeatPara.temperatureOn), (uint8\*)&temp, 2U)

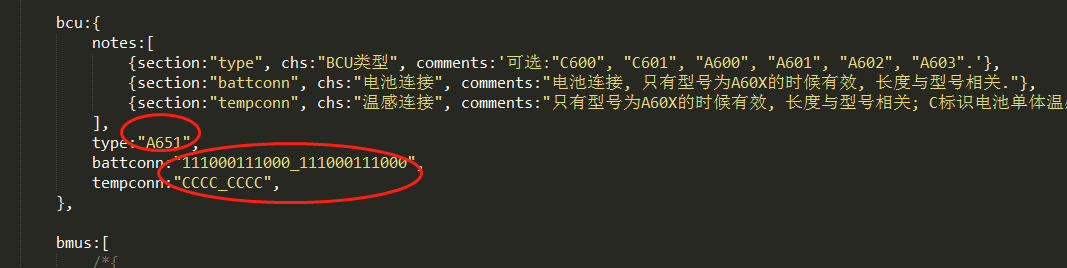
# config.js配置说明



红圈部分填写项目编号，版本号，第一次版本号都是1.0.0



对应相应的协议要求进行配置



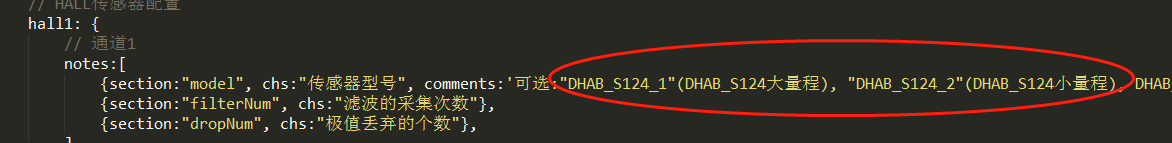
A表示一体机，C开头表示主机，M开头表示从机，battconn表示电池串数，tempconn表示温度，具体数量可以参考《UDAN iU6智能电池管理系统产品规格书.pdf》

需要注意的是！ 若有跨铜牌的配置配置battconn需要注意，跨铜牌位置配置成0

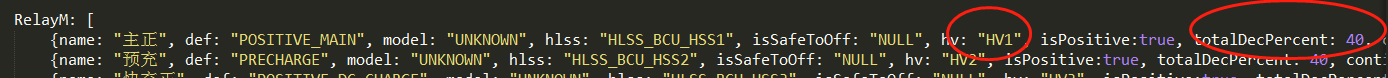
加热片温度？？什么情况用？



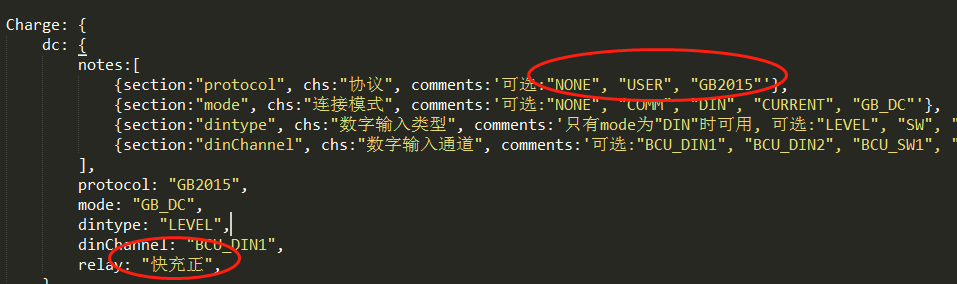
OTA远端升级功能若开启 配置成true



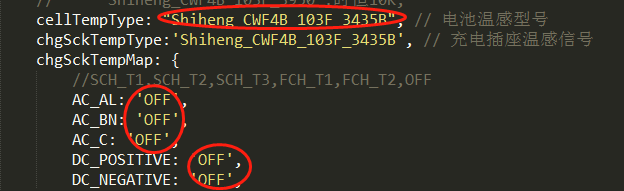
关于传感器配置，注意双量程，如果单量程，那么hall1和hall2配置一样



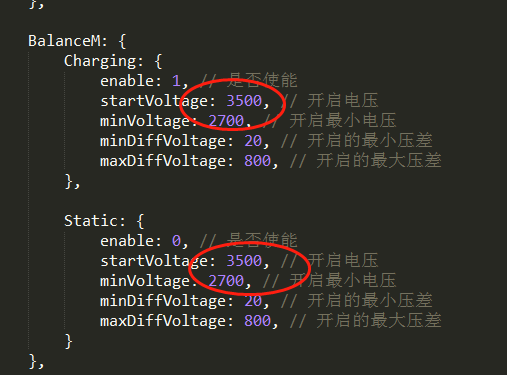
继电器的配置注意黏连的配置以及粘连的检测位置，如果没有预充，记得在PrechargeM\_Lcfg.c中把预充改成RELAYM\_FN\_NONE



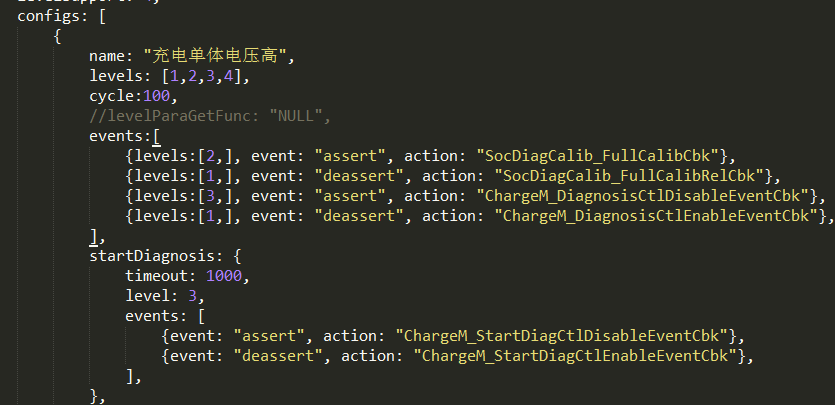
如果没有就配置成none，快充和慢充起码有一个是使能的。如果充电和放电共用继电器，那么relay就配置成主正。



温感型号中没有对应型号，需要驱动层配置相应参数。如果要开启插座温感检测，相应的OFF改成SCH\_T1,SCH\_T2,SCH\_T3，FCH\_T1，FCH\_T2. 如果慢充和快充的插座温感不一致，应该如何配置



这个是磷酸的均衡配置，三元材料的均衡配置是开启电压3900 最小电压是3100，默认是充电开启均衡功能。



Levels：等级配置，如果需要里面就不用填写数字了，startDiagnosis中配置自检条件。

Assert代表触发条件，Deassert代表释放条件。如果故障等级释放参数配置成无效值。则故障就算释放，但是can总线的故障报文不解除。

以下使用默认配置

1，启动初识化

2，电压排线脱落

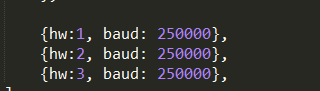
3，温感排线脱落

4，内网

5，电流异常

6，供电电压

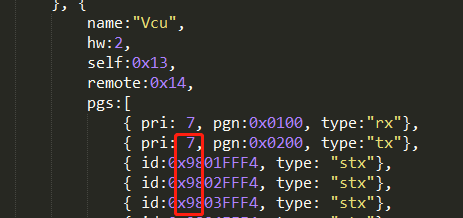
快充连接器过温一般指快充正极高温和快充负极高温



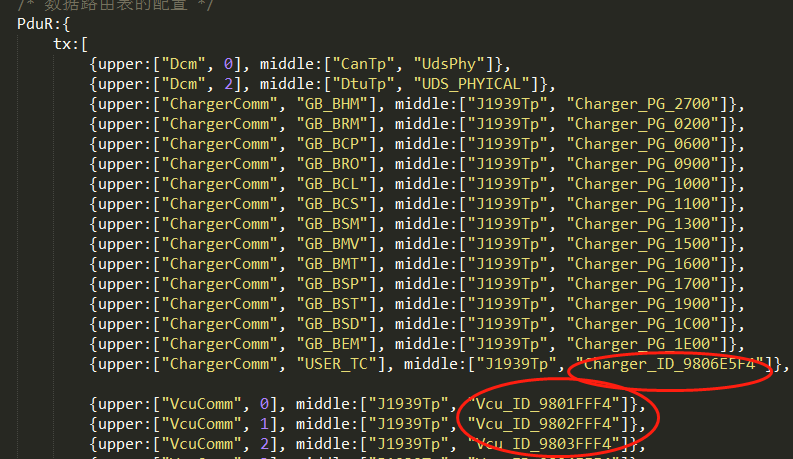
Can波特率配置，内网一般不需要做任何修改



充电can配置，国标一般不需做改动，用户充电通信协议可能做改动，若用户充电通信使用的是整车can那么需要在name:”Vcu”中配置相关充电报文ID，id高位9是1001，最高位是表示扩展的意思

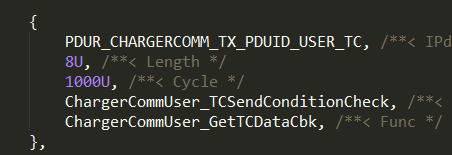


等报文配置完毕后以下发送报文记得修改，接收一样需要修改。

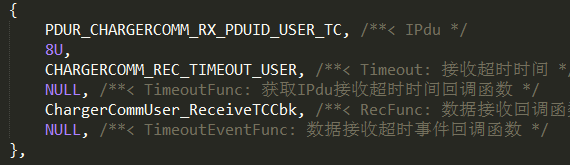


上图中第一个绕圈标注（Charger\_ID\_9806E5F4）为充电机通信用户自定义协议对应的发送报文ID相关，若用户自定义协议在整车CAN，则只需将Charger\_ID\_9806E5F4修改为Vcu\_ID\_9806E5F4即可（需要同步将ID定义到整车CAN）。

# ChargerComm\_LCfg.c配置说明



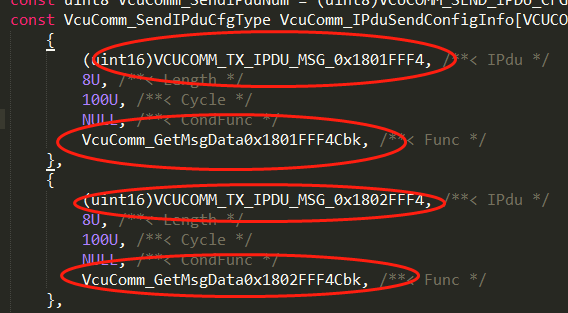
此配置是关于用户充电的函数配置，注意发送周期和字节的配置



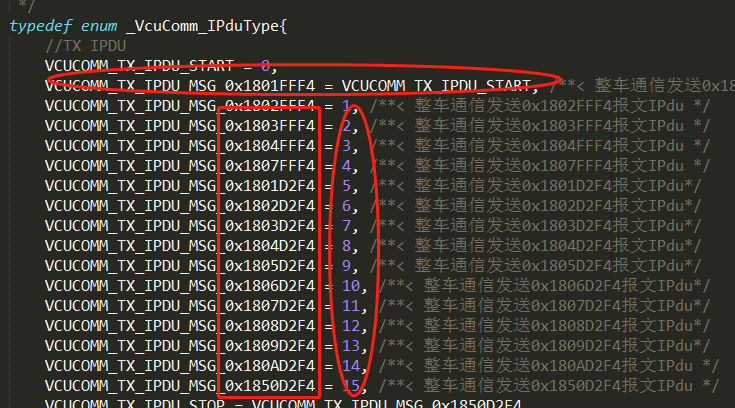
此配置是关于用户充电函数的接收配置。

一般国标充电和车载充电的配置不变。

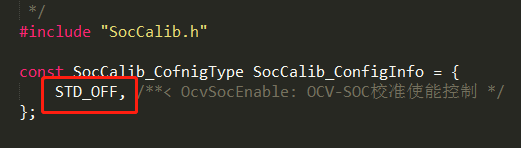
# VcuComm\_LCfg.c配置说明



IPdu名称在VcuComm\_Types.h配置，回调函数在VcuComm\_Messages.c定义。

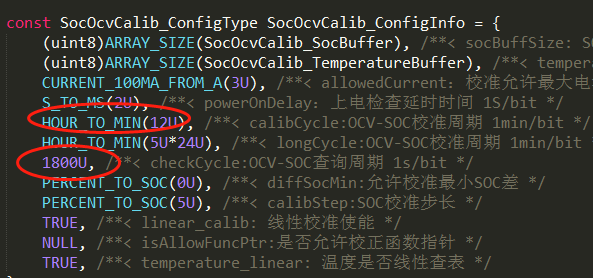


# SocCalib\_Lcfg.c配置说明

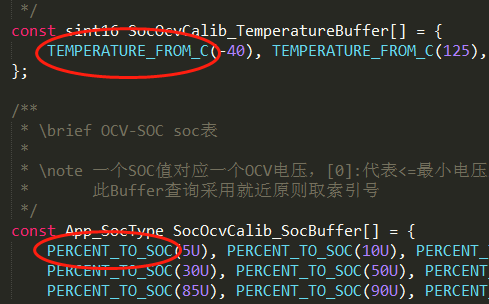


如果开启soc\_ocv功能，则改STD\_ON

# SocOcvCalib\_Lcfg.c配置说明



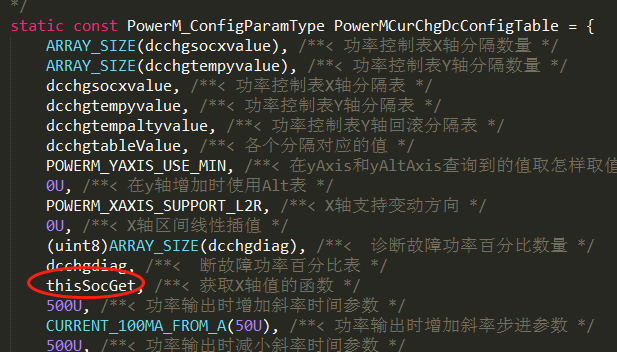
静置时间一般需要配置，为了测试方便有时候时间设置比较短，记得查询时间也应该相应改短



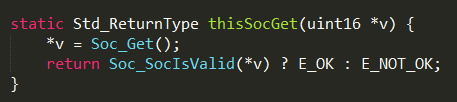
查表数据都应该转换成系统的默认模式，温感偏移量50。

分辨率：电流100mA，总压100mv，单体电压1mv。

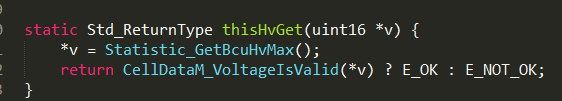
# PowerM\_Lcfg.c配置说明

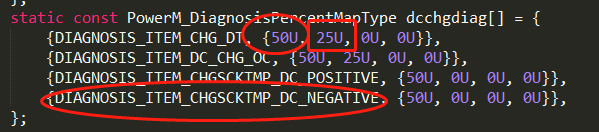


thisSocGet函数在

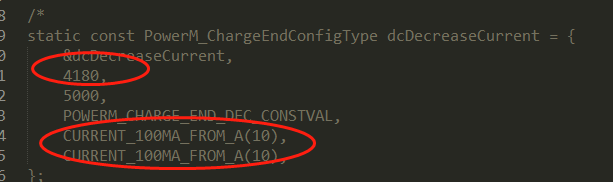


如果改成电压则 使用





这里配置相应故障出现做出的降功率处理，50U意思是降到50%，不是降了50%。



关于末端降流的配置配置，这里的降流链表即上图的dcDecreaseCurrent，这里为空没有末端降流功能。还需配置末端降流的比较值函数thisHvGet（通过电压判断降流）



以下是一个末端降流链表参考：

static const PowerM\_ChargeEndConfigType acDecreaseCurrent4 = {

&acDecreaseCurrent4, //四个链表

4161,

5000,

POWERM\_CHARGE\_END\_DEC\_TO\_CONSTVAL,

CURRENT\_100MA\_FROM\_A(3),

CURRENT\_100MA\_FROM\_A(3),

};

static const PowerM\_ChargeEndConfigType acDecreaseCurrent3 = {

&acDecreaseCurrent4, //四个链表

4140,

5000,

POWERM\_CHARGE\_END\_DEC\_TO\_CONSTVAL,

CURRENT\_100MA\_FROM\_A(5.4),

CURRENT\_100MA\_FROM\_A(3),

};

static const PowerM\_ChargeEndConfigType acDecreaseCurrent2 = {

&acDecreaseCurrent3, //四个链表

4100,

5000,

POWERM\_CHARGE\_END\_DEC\_TO\_CONSTVAL,

CURRENT\_100MA\_FROM\_A(6.5),

CURRENT\_100MA\_FROM\_A(3),

};

static const PowerM\_ChargeEndConfigType acDecreaseCurrent1 = {

&acDecreaseCurrent2, //四个链表

4050,

5000,

POWERM\_CHARGE\_END\_DEC\_TO\_CONSTVAL,

CURRENT\_100MA\_FROM\_A(12),

CURRENT\_100MA\_FROM\_A(3),

};

# 充放电流程说明

**充电流程**

充电流程采用状态机的形式实现，可以在HvProcess\_Chg\_Lcfg.c中配置

可以用以下结构体来定义

typedef struct{

HvProcess\_CondFunc Cond; /\*\*< 条件函数指针 \*/

HvProcess\_ActionFunc Action; /\*\*< 动作函数指针 \*/

uint16 Next; /\*\*< 下一状态 \*/

}HvProcess\_StateInfoType;

然后用

typedef struct{

uint8 Num; /\*\*< 状态信息个数 \*/

const HvProcess\_StateInfoType \*State; /\*\*< 状态信息指针 \*/

}HvProcess\_StateConfigType;

来配置状态中的个数和实现函数

可以在HvProcess\_Chg.h中的结构体

typedef enum{

HVPROCESS\_CHG\_START = 0, /\*\*< 启动状态 \*/

HVPROCESS\_CHG\_HV\_ON = 1, /\*\*< 高压上电状态 \*/

HVPROCESS\_CHG\_RELAY\_OFF\_DELAY = 2, /\*\*< 继电器断开延时状态 \*/

HVPROCESS\_CHG\_STATE\_MAX = 3, /\*\*< 充电高压流程控制状态最大值 \*/

}HvProcess\_ChgStateType;

来对模板中的状态量进行增加和减少

可以在HvProcess\_Chg.h中的结构体

typedef struct{

uint8 chgFinishFlag; /\*\*< 充电完成标志 \*/

HvProcess\_ChgStateType State; /\*\*< 充电高压流程状态 \*/

uint32 RelayOffTick; /\*\*< 继电器断开计时 \*/

boolean RelayFaultCheckFlag; /\*\*< 继电器故障检查标志 \*/

}HvProcess\_ChgInnerDataType;

设置一些全局变量，如时间控制，故障标志

一，充电自检一般包括：

1. CHARGECONNECTM\_IS\_CONNECT()（无充电连接信号或盲冲就不需要判断了）,
2. Std\_ReturnType ChargeM\_ChargeIsAllowed(void);充电是否允许（无充电信号，也不需要判断，系统在无连接信号下进入的放电状态）
3. uint8 ChargerComm\_GetCommunicationStatus(void); 获取充电通信状态（无充电信号，不需要判断，如果充电机需要bms发开机指令的，也不需要判断这个函数）
4. if (HvProcess\_GetDchgState() == HVPROCESS\_DCHG\_START) 这条语句表达是等待放电下电结束，出于安全考虑。处理不当可能会引起充电时报放电继电器粘连故障。

5，if (OSTimeGet() >= 300UL)

{

HvProcess\_ChgInnerData.RelayFaultCheckFlag = TRUE;

(void)RelayM\_StartAdhesiveDetect(RELAYM\_FN\_POSITIVE\_MAIN, NULL);

}

粘连判断。

1. 自检完毕后闭合相应继电器。如果需要可以增加一个判断继电器常开故障的状态。
2. 继电器闭合后，断开的一些条件
3. void HvProcess\_ChgChargeConnectionAction(void);连接故障
4. Std\_ReturnType ChargeM\_ChargeIsFault(void); 充电故障，这里加上满充（也算故障）

满充记得置标志位，（OTA充电末端升级需要）

1. 延时断继电器（时间可以在上一步骤中赋值）
2. 断开继电器，并延时下电（放电状态下，整车中如果收到电机放电结束标志位最安全）

**放电流程**

放电流程和充电流程类似，需要注意的是有的项目需要开启预充。如果放电口和充电口共用，为了方便控制，只需一个放电流程开控制充放电即可。

**用户流程**

在UserStrategy.c文件中配置，比如控制风扇，蜂鸣器，自动下电等功能。

**总结**

实际项目中应针对客户的不同要求和实际的电气图纸灵活的开发最理想的流程。

**常用函数总结**

UserStrategy.c

App\_Tv100mvType UserStrategy\_GetChargeVoltMax(void)

可以获取充电机总压，可以在calibrate.json中配置。

Current\_CurrentType UserStrategy\_GetChargeCurrentMax(void)

可以通过查充电功率表，来确定电流。

Satistic.h

App\_Tv100mvType Statistic\_GetBcu100mvTotalVoltage(void);

获取当前回路电压

Hv.h

App\_Tv100mvType HV\_GetVoltage(HV\_ChannelType channel);

获取相应端口采集电压

CurrentM.h

Current\_CurrentType CurrentM\_GetCurrentCalibrated(CurrentM\_ChannelType channel);

获取某一路电流

Diagnosis.h

Diagnosis\_LevelType Diagnosis\_GetLevel(Diagnosis\_ItemType item);

获取当前状态 可以在Diagnosis\_Cfg.h 查看所有配置故障状态

ChargeM.h

Diagnosis\_StartDiagStateType ChargeM\_GetStartDiagCtl(Diagnosis\_ItemType num);

获取充电电自检诊断状态

ChargeM\_CtlTypeType ChargeM\_GetDiagnosisChargeCtlAction(Diagnosis\_ItemType item);

获取诊断故障当前对充电的动作控制

DischargeM.h

Diagnosis\_StartDiagStateType DischargeM\_GetStartDiagCtl(Diagnosis\_ItemType num);

获取放电启动诊断控制状态

DischargeM\_CtlTypeType DischargeM\_GetDiagnosisDchargeCtlAction(Diagnosis\_ItemType item);

获取诊断故障当前对放电的控制动作