**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