```
/**
 *
 * @copyright &copy; 2010 - 2020, Fraunhofer-Gesellschaft zur Foerderung der
 *  angewandten Forschung e.V. All rights reserved.
 *
 * BSD 3-Clause License
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 * 1.  Redistributions of source code must retain the above copyright notice,
 *     this list of conditions and the following disclaimer.
 * 2.  Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 * 3.  Neither the name of the copyright holder nor the names of its
 *     contributors may be used to endorse or promote products derived from
 *     this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 * We kindly request you to use one or more of the following phrases to refer
 * to foxBMS in your hardware, software, documentation or advertising
 * materials:
 *
 * &Prime;This product uses parts of foxBMS&reg;&Prime;
 *
 * &Prime;This product includes parts of foxBMS&reg;&Prime;
 *
 * &Prime;This product is derived from foxBMS&reg;&Prime;
 *
 */

/**
 * @file    diag_cfg.c
 * @author  foxBMS Team
 * @date    09.11.2015 (date of creation)
 * @ingroup ENGINE_CONF
 * @prefix  DIAG
 *
 * @brief   Diagnostic module configuration
 *
 * The configuration of the different diagnosis events defined in diag_cfg.h is set in the array
 * diag_ch_cfg[], e.g. initialization errors or runtime errors.
```

```
 *
 * Every entry of the diag_ch_cfg[] array consists of
 *  - name of the diagnosis event (defined in diag_cfg.h)
 *  - type of diagnosis event
 *  - diagnosis sensitivity (after how many occurrences event is counted as error)
 *  - enabling of the recording for diagnosis event
 *  - enabling of the diagnosis event
 *  - callback function for diagnosis event if wished, otherwise dummyfu
 *
 * The system monitoring configuration defined in diag_cfg.h is set in the array
 * diag_sysmon_ch_cfg[]. The system monitoring is at the moment only used for
 * supervising the cyclic/periodic tasks.
 *
 * Every entry of the diag_sysmon_ch_cfg[] consists of
 *  - enum of monitored object
 *  - type of monitored object (at the moment only DIAG_SYSMON_CYCLICTASK is supported)
 *  - maximum delay in [ms] in which the object needs to call the DIAG_SysMonNotify function defined in diag.c
 *  - enabling of the recording for system monitoring
 *  - enabling of the system monitoring for the monitored object
 *  - callback function if system monitoring notices an error if wished, otherwise dummyfu2
 */

/*================== Includes ==================================================*/
#include "diag_cfg.h"

#include "database.h"
#include "rtc.h"

/*================== Macros and Definitions ====================================*/

/*================== Static Constant and Variable Definitions ==================*/
static DATA_BLOCK_ERRORSTATE_s error_flags = { 0 };
static DATA_BLOCK_MOL_FLAG_s mol_flags = { 0 };
static DATA_BLOCK_RSL_FLAG_s rsl_flags = { 0 };
static DATA_BLOCK_MSL_FLAG_s msl_flags = { 0 };

/*================== Static Function Prototypes ==============
/* dummy functions */
static void dummyfu(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
static void dummyfu2(DIAG_SYSMON_MODULE_ID_e ch_id);

/* functions for SOA related events */
static void DIAG_overvoltage(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
static void DIAG_undervoltage(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
static void DIAG_overtemperature_charge(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
static void DIAG_overtemperature_discharge(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
static void DIAG_undertemperature_charge(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
static void DIAG_undertemperature_discharge(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
static void DIAG_overcurrent_charge(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
static void DIAG_overcurrent_discharge(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);

/* functions for system related events */
```

```
    DIAG_CH_LOW_COIN_CELL_VOLTAGE, /* coin cell voltage */
    DIAG_CH_CRIT_LOW_COIN_CELL_VOLTAGE, /* coin cell voltage */
    DIAG_CH_OPEN_WIRE, /* open-wire check */
    DIAG_CH_PLAUSIBILITY_CELL_VOLTAGE, /* plausibility checks */
    DIAG_CH_PLAUSIBILITY_CELL_TEMP, /* plausibility checks */
    DIAG_CH_PLAUSIBILITY_PACK_VOLTAGE, /* plausibility checks */
    DIAG_CH_DEEP_DISCHARGE_DETECTED, /* DoD was detected */
    DIAG_ID_MAX, /* MAX indicator - do not change */
} DIAG_CH_ID_e;
```

```
/** diagnosis check res
typedef enum {
    DIAG_EVENT_OK, /*!<
    DIAG_EVENT_NOK, /*!
    DIAG_EVENT_RESET, /
} DIAG_EVENT_e;
```

```
105  static void DIAG_error_cantiming(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
106  static void DIAG_error_ltc(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
107  static void DIAG_error_cancurrentsensor(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
108  static void DIAG_cont_feedback(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
109  static void DIAG_error_fuseState(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
110  static void DIAG_error_interlock(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
111  static void DIAG_error_insulation(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
112  static void DIAG_error_openWire(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
113  static void DIAG_error_deep_discharge_detected(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
114  static void DIAG_error_MCUdieTemperature(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
115  static void DIAG_error_coinCellVoltage(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
116
117  /* functions for plausibility related events */
118  static void DIAG_error_plausibility_check(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event);
119
120  /*================== Extern Constant and Variable Definitions ===============*/
121  /**
122   * Enable and Disable of Error Checks for Testing Purposes
123   *
124   * Each Bit enables or disables the diagnosis (checking of) the corresponding failure code
125   * FIXME struct isn't used anywhere in the code at the moment.
126   * FIXME delete if not needed
127  */
128  DIAG_CODE_s diag_mask = {
129          .GENERALmsk = 0xFFFFFFFF,
130          .CELLMONmsk = 0xFFFFFFFF,
131          .COMmsk = 0xFFFFFFFF,
132          .ADCmsk = 0xFFFFFFFF,
133  };
134
135  DIAG_CH_CFG_s  diag_ch_cfg[] = {
136      /* OS-Framework and startup events */
137      {DIAG_CH_FLASHCHECKSUM,                    "FLASHCHECKSUM",
       DIAG_ERROR_SENSITIVITY_HIGH,          DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
138      {DIAG_CH_BKPDIAG_FAILURE,                  "BKPDIAG",
       DIAG_ERROR_SENSITIVITY_HIGH,          DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
139      {DIAG_CH_WATCHDOGRESET_FAILURE,            "WATCHDOGRESET",
       DIAG_ERROR_SENSITIVITY_HIGH,          DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
140      {DIAG_CH_POSTOSINIT_FAILURE,               "POSTOSINIT",
       DIAG_ERROR_SENSITIVITY_HIGH,          DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
141      {DIAG_CH_CALIB_EEPR_FAILURE,               "CALIB_EEPR",
       DIAG_ERROR_SENSITIVITY_HIGH,          DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
142      {DIAG_CH_CAN_INIT_FAILURE,                 "CAN_INIT",
       DIAG_ERROR_SENSITIVITY_HIGH,          DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
143      {DIAG_CH_VIC_INIT_FAILURE,                 "VIC_INIT",
       DIAG_ERROR_SENSITIVITY_HIGH,          DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
144
145      /* HW-/SW-Runtime events */
146      {DIAG_CH_DIV_BY_ZERO_FAILURE,              "DIV_BY_ZERO",
       DIAG_ERROR_SENSITIVITY_HIGH,          DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
147      {DIAG_CH_UNDEF_INSTRUCTION_FAILURE,        "UNDEF_INSTRUCTION",
       DIAG_ERROR_SENSITIVITY_HIGH,          DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
```

```
305  typedef struct {
306      DIAG_CH_ID_e id;                          /*!< diagnos
307      uint8_t description[40];
308      uint16_t thresholds;                       /*!< thresh
         generating a notification in both direction (OK or N
309                                                  *   threshc
                                                    1:reports t
310      DIAG_TYPE_RECORDING_e enablerecording;    /*!< if enab
311      DIAG_ENABLE_STATE_e state;                /*!< if enab
312      void (*callbackfunc)(DIAG_CH_ID_e, DIAG_EVENT_e);
         (in both direction) with parameter DIAG_EVENT_e */
313  } DIAG_CH_CFG_s;
```

```
148       {DIAG_CH_DATA_BUS_FAILURE,                    "DATA_BUS_FAILURE",
          DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
149       {DIAG_CH_INSTRUCTION_BUS_FAILURE,             "INSTRUCTION_BUS",
          DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
150       {DIAG_CH_HARDFAULT_NOTHANDLED,                "HARDFAULT_NOTHANDLED",
          DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
151
152       {DIAG_CH_CONFIGASSERT,                        "CONFIGASSERT",
          DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
153       {DIAG_CH_SYSTEMMONITORING_TIMEOUT,            "SYSTEMMONITORING_TIMEOUT",
          DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
154
155
156       /* Measurement events */
157       {DIAG_CH_CANS_MAX_VALUE_VIOLATE,              "CANS_MAX_VALUE_VIOLATE",
          DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
158       {DIAG_CH_CANS_MIN_VALUE_VIOLATE,              "CANS_MIN_VALUE_VIOLATE",
          DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
159       {DIAG_CH_CANS_CAN_MOD_FAILURE,                "CANS_CAN_MOD_FAILURE",
          DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
160
161   #if BUILD_MODULE_ENABLE_ISOGUARD == 1
162       {DIAG_CH_ISOMETER_TIM_ERROR,                  "ISOMETER_TIM_ERROR",
          DIAG_ERROR_SENSITIVITY_MID,               DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
163       {DIAG_CH_ISOMETER_GROUNDERROR,                "ISOMETER_GROUNDERROR",
          DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
164       {DIAG_CH_ISOMETER_ERROR,                      "ISOMETER_ERROR",
          DIAG_ERROR_SENSITIVITY_MID,               DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
165       {DIAG_CH_ISOMETER_MEAS_INVALID,               "ISOMETER_MEAS_INVALID",
          DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
166       {DIAG_CH_INSULATION_ERROR,                    "INSULATION_ERROR",
          DIAG_ERROR_INSULATION_SENSITIVITY,        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_insulation},
167   #else
168       {DIAG_CH_ISOMETER_TIM_ERROR,                  "ISOMETER_TIM_ERROR",
          DIAG_ERROR_SENSITIVITY_MID,               DIAG_RECORDING_DISABLED, DIAG_DISABLED, dummyfu},
169       {DIAG_CH_ISOMETER_GROUNDERROR,                "ISOMETER_GROUNDERROR",
          DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_DISABLED, DIAG_DISABLED, dummyfu},
170       {DIAG_CH_ISOMETER_ERROR,                      "ISOMETER_ERROR",
          DIAG_ERROR_SENSITIVITY_MID,               DIAG_RECORDING_DISABLED, DIAG_DISABLED, dummyfu},
171       {DIAG_CH_ISOMETER_MEAS_INVALID,               "ISOMETER_MEAS_INVALID",
          DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_DISABLED, DIAG_DISABLED, dummyfu},
172       {DIAG_CH_INSULATION_ERROR,                    "INSULATION_ERROR",
          DIAG_ERROR_INSULATION_SENSITIVITY,        DIAG_RECORDING_DISABLED, DIAG_DISABLED, dummyfu},
173   #endif
174
175       /* Under and over temperature, voltage and current at cell level */
176       {DIAG_CH_CELLVOLTAGE_OVERVOLTAGE_MSL,             "CELLVOLT_OVERVOLT_MSL",
          DIAG_ERROR_VOLTAGE_SENSITIVITY_MSL,         DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overvoltage},
177       {DIAG_CH_CELLVOLTAGE_OVERVOLTAGE_RSL,             "CELLVOLT_OVERVOLT_RSL",
          DIAG_ERROR_VOLTAGE_SENSITIVITY_RSL,         DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overvoltage},
178       {DIAG_CH_CELLVOLTAGE_OVERVOLTAGE_MOL,          "CELLVOLT_OVERVOLT_MOL",
          DIAG_ERROR_VOLTAGE_SENSITIVITY_MOL,         DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overvoltage},
```

```
179
180     {DIAG_CH_CELLVOLTAGE_UNDERVOLTAGE_MSL,          "CELLVOLT_UNDERVOLT_MSL",
        DIAG_ERROR_VOLTAGE_SENSITIVITY_MSL,         DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_undervoltage},
181     {DIAG_CH_CELLVOLTAGE_UNDERVOLTAGE_RSL,          "CELLVOLT_UNDERVOLT_RSL",
        DIAG_ERROR_VOLTAGE_SENSITIVITY_RSL,         DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_undervoltage},
182     {DIAG_CH_CELLVOLTAGE_UNDERVOLTAGE_MOL,          "CELLVOLT_UNDERVOLT_MOL",
        DIAG_ERROR_VOLTAGE_SENSITIVITY_MOL,         DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_undervoltage},
183
184     {DIAG_CH_TEMP_OVERTEMPERATURE_CHARGE_MSL,       "OVERTEMP_CHARGE_MSL",
        DIAG_ERROR_TEMPERATURE_SENSITIVITY_MSL,     DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overtemperature_charge},
185     {DIAG_CH_TEMP_OVERTEMPERATURE_CHARGE_RSL,       "OVERTEMP_CHARGE_RSL",
        DIAG_ERROR_TEMPERATURE_SENSITIVITY_RSL,     DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overtemperature_charge},
186     {DIAG_CH_TEMP_OVERTEMPERATURE_CHARGE_MOL,       "OVERTEMP_CHARGE_MOL",
        DIAG_ERROR_TEMPERATURE_SENSITIVITY_MOL,     DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overtemperature_charge},
187
188     {DIAG_CH_TEMP_OVERTEMPERATURE_DISCHARGE_MSL,    "OVERTEMP_DISCHARGE_MSL",
        DIAG_ERROR_TEMPERATURE_SENSITIVITY_MSL,     DIAG_RECORDING_ENABLED, DIAG_ENABLED,
        DIAG_overtemperature_discharge},
189     {DIAG_CH_TEMP_OVERTEMPERATURE_DISCHARGE_RSL,    "OVERTEMP_DISCHARGE_RSL",
        DIAG_ERROR_TEMPERATURE_SENSITIVITY_MSL,     DIAG_RECORDING_ENABLED, DIAG_ENABLED,
        DIAG_overtemperature_discharge},
190     {DIAG_CH_TEMP_OVERTEMPERATURE_DISCHARGE_MOL,    "OVERTEMP_DISCHARGE_MOL",
        DIAG_ERROR_TEMPERATURE_SENSITIVITY_MSL,     DIAG_RECORDING_ENABLED, DIAG_ENABLED,
        DIAG_overtemperature_discharge},
191
192     {DIAG_CH_TEMP_UNDERTEMPERATURE_CHARGE_MSL,      "UNDERTEMP_CHARGE_MSL",
        DIAG_ERROR_TEMPERATURE_SENSITIVITY_MSL,     DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_undertemperature_charge},
193     {DIAG_CH_TEMP_UNDERTEMPERATURE_CHARGE_RSL,      "UNDERTEMP_CHARGE_RSL",
        DIAG_ERROR_TEMPERATURE_SENSITIVITY_MSL,     DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_undertemperature_charge},
194     {DIAG_CH_TEMP_UNDERTEMPERATURE_CHARGE_MOL,      "UNDERTEMP_CHARGE_MOL",
        DIAG_ERROR_TEMPERATURE_SENSITIVITY_MSL,     DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_undertemperature_charge},
195
196     {DIAG_CH_TEMP_UNDERTEMPERATURE_DISCHARGE_MSL,   "UNDERTEMP_DISCHARGE_MSL",
        DIAG_ERROR_TEMPERATURE_SENSITIVITY_MSL,     DIAG_RECORDING_ENABLED, DIAG_ENABLED,
        DIAG_undertemperature_discharge},
197     {DIAG_CH_TEMP_UNDERTEMPERATURE_DISCHARGE_RSL,   "UNDERTEMP_DISCHARGE_RSL",
        DIAG_ERROR_TEMPERATURE_SENSITIVITY_MSL,     DIAG_RECORDING_ENABLED, DIAG_ENABLED,
        DIAG_undertemperature_discharge},
198     {DIAG_CH_TEMP_UNDERTEMPERATURE_DISCHARGE_MOL,   "UNDERTEMP_DISCHARGE_MOL",
        DIAG_ERROR_TEMPERATURE_SENSITIVITY_MSL,     DIAG_RECORDING_ENABLED, DIAG_ENABLED,
        DIAG_undertemperature_discharge},
199
200     {DIAG_CH_OVERCURRENT_PL_NONE,           "OVERCUR_NO_POWERLINE",     DIAG_ERROR_CURRENT_SENSITIVITY_MSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_charge},
201
202     {DIAG_CH_OVERCURRENT_CHARGE_CELL_MSL,   "OVERCUR_CHRG_CELL_MSL",    DIAG_ERROR_CURRENT_SENSITIVITY_MSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_charge},
203     {DIAG_CH_OVERCURRENT_CHARGE_CELL_RSL,   "OVERCUR_CHRG_CELL_RSL",    DIAG_ERROR_CURRENT_SENSITIVITY_RSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_charge},
204     {DIAG_CH_OVERCURRENT_CHARGE_CELL_MOL,   "OVERCUR_CHRG_CELL_MOL",    DIAG_ERROR_CURRENT_SENSITIVITY_MOL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_charge},
205
```

```
206     {DIAG_CH_OVERCURRENT_DISCHARGE_CELL_MSL,  "OVERCUR_DCHRG_CELL_MSL",    DIAG_ERROR_CURRENT_SENSITIVITY_MSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_discharge},
207     {DIAG_CH_OVERCURRENT_DISCHARGE_CELL_RSL,  "OVERCUR_DCHRG_CELL_RSL",    DIAG_ERROR_CURRENT_SENSITIVITY_RSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_discharge},
208     {DIAG_CH_OVERCURRENT_DISCHARGE_CELL_MOL,  "OVERCUR_DCHRG_CELL_MOL",    DIAG_ERROR_CURRENT_SENSITIVITY_MOL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_discharge},
209
210     {DIAG_CH_OVERCURRENT_CHARGE_PL0_MSL,       "OVERCUR_CHRG_PL0_MSL",     DIAG_ERROR_CURRENT_SENSITIVITY_MSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_charge},
211     {DIAG_CH_OVERCURRENT_CHARGE_PL0_RSL,       "OVERCUR_CHRG_PL0_RSL",     DIAG_ERROR_CURRENT_SENSITIVITY_RSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_charge},
212     {DIAG_CH_OVERCURRENT_CHARGE_PL0_MOL,       "OVERCUR_CHRG_PL0_MOL",     DIAG_ERROR_CURRENT_SENSITIVITY_MOL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_charge},
213
214     {DIAG_CH_OVERCURRENT_DISCHARGE_PL0_MSL,    "OVERCUR_DCHRG_PL0_MSL",    DIAG_ERROR_CURRENT_SENSITIVITY_MSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_discharge},
215     {DIAG_CH_OVERCURRENT_DISCHARGE_PL0_RSL,    "OVERCUR_DCHRG_PL0_RSL",    DIAG_ERROR_CURRENT_SENSITIVITY_RSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_discharge},
216     {DIAG_CH_OVERCURRENT_DISCHARGE_PL0_MOL,    "OVERCUR_DCHRG_PL0_MOL",    DIAG_ERROR_CURRENT_SENSITIVITY_MOL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_discharge},
217
218     {DIAG_CH_OVERCURRENT_CHARGE_PL1_MSL,       "OVERCUR_CHRG_PL1_MSL",     DIAG_ERROR_CURRENT_SENSITIVITY_MSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_charge},
219     {DIAG_CH_OVERCURRENT_CHARGE_PL1_RSL,       "OVERCUR_CHRG_PL1_RSL",     DIAG_ERROR_CURRENT_SENSITIVITY_RSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_charge},
220     {DIAG_CH_OVERCURRENT_CHARGE_PL1_MOL,       "OVERCUR_CHRG_PL1_MOL",     DIAG_ERROR_CURRENT_SENSITIVITY_MOL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_charge},
221
222     {DIAG_CH_OVERCURRENT_DISCHARGE_PL1_MSL,    "OVERCUR_DCHRG_PL1_MSL",    DIAG_ERROR_CURRENT_SENSITIVITY_MSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_discharge},
223     {DIAG_CH_OVERCURRENT_DISCHARGE_PL1_RSL,    "OVERCUR_DCHRG_PL1_RSL",    DIAG_ERROR_CURRENT_SENSITIVITY_RSL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_discharge},
224     {DIAG_CH_OVERCURRENT_DISCHARGE_PL1_MOL,    "OVERCUR_DCHRG_PL1_MOL",    DIAG_ERROR_CURRENT_SENSITIVITY_MOL,
        DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_overcurrent_discharge},
225
226     {DIAG_CH_LTC_SPI,                                  "LTC_SPI",
        DIAG_ERROR_LTC_SPI_SENSITIVITY,           DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_ltc},
227     {DIAG_CH_LTC_PEC,                                  "LTC_PEC",
        DIAG_ERROR_LTC_PEC_SENSITIVITY,           DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_ltc},
228     {DIAG_CH_LTC_MUX,                                  "LTC_MUX",
        DIAG_ERROR_LTC_MUX_SENSITIVITY,           DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_ltc},
229     {DIAG_CH_LTC_CONFIG,                               "LTC_CONFIG",
        DIAG_ERROR_SENSITIVITY_HIGH,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_ltc},
230
231     /* Communication events */
232     {DIAG_CH_CAN_TIMING,                               "CAN_TIMING",
        DIAG_ERROR_CAN_TIMING_SENSITIVITY,        DIAG_RECORDING_ENABLED, DIAG_CAN_TIMING, DIAG_error_cantiming},
233     {DIAG_CH_CAN_CC_RESPONDING,                        "CAN_CC_RESPONDING",
        DIAG_ERROR_CAN_TIMING_CC_SENSITIVITY,     DIAG_RECORDING_ENABLED, DIAG_CAN_SENSOR_PRESENT, DIAG_error_cantiming},
234     {DIAG_CH_CURRENT_SENSOR_RESPONDING,                "CURRENT_SENSOR_RESPONDING",
        DIAG_ERROR_CAN_SENSOR_SENSITIVITY,        DIAG_RECORDING_ENABLED, DIAG_CAN_SENSOR_PRESENT,
        DIAG_error_cancurrentsensor},
```

```
235
236 #if BUILD_MODULE_ENABLE_CONTACTOR == 1
237     /* Contactor Damage Error */
238     {DIAG_CH_CONTACTOR_DAMAGED,                    "CONTACTOR_DAMAGED",
      DIAG_ERROR_SENSITIVITY_HIGH,             DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
239     {DIAG_CH_CONTACTOR_OPENING,                    "CONTACTOR_OPENING",
      DIAG_ERROR_SENSITIVITY_HIGH,             DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
240     {DIAG_CH_CONTACTOR_CLOSING,                    "CONTACTOR_CLOSING",
      DIAG_ERROR_SENSITIVITY_HIGH,             DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
241
242     /* Contactor Feedback Error */
243     {DIAG_CH_CONTACTOR_MAIN_PLUS_FEEDBACK,             "CONT_MAIN_PLUS_FEED",
      DIAG_ERROR_MAIN_PLUS_SENSITIVITY,      DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_cont_feedback },
244     {DIAG_CH_CONTACTOR_MAIN_MINUS_FEEDBACK,            "CONT_MAIN_MINUS_FEED",
      DIAG_ERROR_MAIN_MINUS_SENSITIVITY,     DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_cont_feedback },
245     {DIAG_CH_CONTACTOR_PRECHARGE_FEEDBACK,             "CONT_PRECHARGE_FEED",
      DIAG_ERROR_PRECHARGE_SENSITIVITY,      DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_cont_feedback },
246     {DIAG_CH_CONTACTOR_CHARGE_MAIN_PLUS_FEEDBACK,      "CONT_CHRGE_MAIN_PLUS_FEED",
      DIAG_ERROR_MAIN_PLUS_SENSITIVITY,      DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_cont_feedback },
247     {DIAG_CH_CONTACTOR_CHARGE_MAIN_MINUS_FEEDBACK,     "CONT_CHRGE_MAIN_MINUS_FEED",
      DIAG_ERROR_MAIN_MINUS_SENSITIVITY,     DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_cont_feedback },
248     {DIAG_CH_CONTACTOR_CHARGE_PRECHARGE_FEEDBACK,      "CONT_CHRGE_PRECHARGE_FEED",
      DIAG_ERROR_PRECHARGE_SENSITIVITY,      DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_cont_feedback },
249
250     /* Fuse state */
251     {DIAG_CH_FUSE_STATE_NORMAL,                    "FUSE_STATE_NORMAL",
      DIAG_ERROR_SENSITIVITY_LOW,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_fuseState },
252     {DIAG_CH_FUSE_STATE_CHARGE,                    "FUSE_STATE_CHARGE",
      DIAG_ERROR_SENSITIVITY_LOW,              DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_fuseState },
253 #else
254     /* Contactor Damage Error */
255     {DIAG_CH_CONTACTOR_DAMAGED,                    "CONTACTOR_DAMAGED",
      DIAG_ERROR_SENSITIVITY_HIGH,             DIAG_RECORDING_DISABLED, DIAG_DISABLED, dummyfu},
256     {DIAG_CH_CONTACTOR_OPENING,                    "CONTACTOR_OPENING",
      DIAG_ERROR_SENSITIVITY_HIGH,             DIAG_RECORDING_DISABLED, DIAG_DISABLED, dummyfu},
257     {DIAG_CH_CONTACTOR_CLOSING,                    "CONTACTOR_CLOSING",
      DIAG_ERROR_SENSITIVITY_HIGH,             DIAG_RECORDING_DISABLED, DIAG_DISABLED, dummyfu},
258
259     /* Contactor Feedback Error */
260     {DIAG_CH_CONTACTOR_MAIN_PLUS_FEEDBACK,             "CONT_MAIN_PLUS_FEED",
      DIAG_ERROR_MAIN_PLUS_SENSITIVITY,      DIAG_RECORDING_DISABLED, DIAG_DISABLED, DIAG_cont_feedback},
261     {DIAG_CH_CONTACTOR_MAIN_MINUS_FEEDBACK,            "CONT_MAIN_MINUS_FEED",
      DIAG_ERROR_MAIN_MINUS_SENSITIVITY,     DIAG_RECORDING_DISABLED, DIAG_DISABLED, DIAG_cont_feedback},
262     {DIAG_CH_CONTACTOR_PRECHARGE_FEEDBACK,             "CONT_PRECHARGE_FEED",
      DIAG_ERROR_PRECHARGE_SENSITIVITY,      DIAG_RECORDING_DISABLED, DIAG_DISABLED, DIAG_cont_feedback},
263     {DIAG_CH_CONTACTOR_CHARGE_MAIN_PLUS_FEEDBACK,      "CONT_CHRGE_MAIN_PLUS_FEED",
      DIAG_ERROR_MAIN_PLUS_SENSITIVITY,      DIAG_RECORDING_DISABLED, DIAG_DISABLED, DIAG_cont_feedback},
264     {DIAG_CH_CONTACTOR_CHARGE_MAIN_MINUS_FEEDBACK,     "CONT_CHRGE_MAIN_MINUS_FEED",
      DIAG_ERROR_MAIN_MINUS_SENSITIVITY,     DIAG_RECORDING_DISABLED, DIAG_DISABLED, DIAG_cont_feedback},
265     {DIAG_CH_CONTACTOR_CHARGE_PRECHARGE_FEEDBACK,      "CONT_CHRGE_PRECHARGE_FEED",
      DIAG_ERROR_PRECHARGE_SENSITIVITY,      DIAG_RECORDING_DISABLED, DIAG_DISABLED, DIAG_cont_feedback},
266
```

```
267         /* Fuse state */
268         {DIAG_CH_FUSE_STATE_NORMAL,                     "FUSE_STATE_NORMAL",
            DIAG_ERROR_SENSITIVITY_LOW,           DIAG_RECORDING_DISABLED, DIAG_DISABLED, DIAG_error_fuseState},
269         {DIAG_CH_FUSE_STATE_CHARGE,                     "FUSE_STATE_CHARGE",
            DIAG_ERROR_SENSITIVITY_LOW,           DIAG_RECORDING_DISABLED, DIAG_DISABLED, DIAG_error_fuseState},
270
271     #endif
272
273     #if BUILD_MODULE_ENABLE_ILCK == 1
274         /* Interlock Feedback Error */
275         {DIAG_CH_INTERLOCK_FEEDBACK,                    "INTERLOCK_FEEDBACK",
            DIAG_ERROR_INTERLOCK_SENSITIVITY,       DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_interlock},
276     #else
277         {DIAG_CH_INTERLOCK_FEEDBACK,                    "INTERLOCK_FEEDBACK",
            DIAG_ERROR_INTERLOCK_SENSITIVITY,       DIAG_RECORDING_DISABLED, DIAG_DISABLED, DIAG_error_interlock},
278     #endif
279
280         /* Slave PCB temperature errors for under and over temperature */
281         {DIAG_CH_SLAVE_PCB_UNDERTEMPERATURE_MSL,          "SLAVE_PCB_UNDERTEMP_MSL",
            DIAG_ERROR_SLAVE_TEMP_SENSITIVITY_MSL,   DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
282         {DIAG_CH_SLAVE_PCB_UNDERTEMPERATURE_RSL,          "SLAVE_PCB_UNDERTEMP_RSL",
            DIAG_ERROR_SLAVE_TEMP_SENSITIVITY_RSL,   DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
283         {DIAG_CH_SLAVE_PCB_UNDERTEMPERATURE_MOL,        "SLAVE_PCB_UNDERTEMP_MOL",
            DIAG_ERROR_SLAVE_TEMP_SENSITIVITY_MOL, DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
284
285         {DIAG_CH_SLAVE_PCB_OVERTEMPERATURE_MSL,           "SLAVE_PCB_OVERTEMP_MSL",
            DIAG_ERROR_SLAVE_TEMP_SENSITIVITY_MSL,   DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
286         {DIAG_CH_SLAVE_PCB_OVERTEMPERATURE_RSL,           "SLAVE_PCB_OVERTEMP_RSL",
            DIAG_ERROR_SLAVE_TEMP_SENSITIVITY_RSL,   DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
287         {DIAG_CH_SLAVE_PCB_OVERTEMPERATURE_MOL,         "SLAVE_PCB_OVERTEMP_MOL",
            DIAG_ERROR_SLAVE_TEMP_SENSITIVITY_MOL, DIAG_RECORDING_ENABLED, DIAG_ENABLED, dummyfu},
288
289         {DIAG_CH_ERROR_MCU_DIE_TEMPERATURE,     "MCU_DIE_TEMPERATURE",      DIAG_ERROR_SENSITIVITY_LOW,
            DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_MCUdieTemperature},
290         {DIAG_CH_LOW_COIN_CELL_VOLTAGE,         "COIN_CELL_VOLT_LOW",       DIAG_ERROR_SENSITIVITY_LOW,
            DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_coinCellVoltage},
291         {DIAG_CH_CRIT_LOW_COIN_CELL_VOLTAGE,    "COIN_CELL_VOLT_CRITICAL",  DIAG_ERROR_SENSITIVITY_LOW,
            DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_coinCellVoltage},
292
293         {DIAG_CH_OPEN_WIRE,       "OPEN_WIRE",         DIAG_ERROR_SENSITIVITY_HIGH, DIAG_RECORDING_DISABLED,
            DIAG_DISABLED, DIAG_error_openWire},
294         {DIAG_CH_DEEP_DISCHARGE_DETECTED,    "DEEP-DISCHARGE detected", DIAG_ERROR_SENSITIVITY_HIGH,
            DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_deep_discharge_detected},
295
296         /* Plausibility checks */
297         {DIAG_CH_PLAUSIBILITY_CELL_VOLTAGE,    "PL_CELL_VOLT",    DIAG_ERROR_SENSITIVITY_HIGH, DIAG_RECORDING_ENABLED,
            DIAG_ENABLED, DIAG_error_plausibility_check},
298         {DIAG_CH_PLAUSIBILITY_CELL_TEMP,       "PL_CELL_TEMP",    DIAG_ERROR_SENSITIVITY_HIGH, DIAG_RECORDING_ENABLED,
            DIAG_ENABLED, DIAG_error_plausibility_check},
299         {DIAG_CH_PLAUSIBILITY_PACK_VOLTAGE,    "PL_PACK_VOLT",    DIAG_ERROR_PLAUSIBILITY_PACK_SENSITIVITY,
            DIAG_RECORDING_ENABLED, DIAG_ENABLED, DIAG_error_plausibility_check},
300     };
```

```
301
302                                                    We cannot switch off the connectors
303  DIAG_SYSMON_CH_CFG_s diag_sysmon_ch_cfg[] = {
304      {DIAG_SYSMON_DATABASE_ID,        DIAG_SYSMON_CYCLICTASK,  10, DIAG_RECORDING_E
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_ENABLED, dummyfu2},
305      {DIAG_SYSMON_SYS_ID,             DIAG_SYSMON_CYCLICTASK,  20, DIAG_RECORDING_E
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_ENABLED, dummyfu2},
306      {DIAG_SYSMON_BMS_ID,             DIAG_SYSMON_CYCLICTASK,  20, DIAG_RECORDING_ENABLED,
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_ENABLED, dummyfu2},
307
308  #if BUILD_MODULE_ENABLE_CONTACTOR == 1
309      {DIAG_SYSMON_CONT_ID,            DIAG_SYSMON_CYCLICTASK,  20, DIAG_RECORDING_ENABLED,
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_ENABLED, dummyfu2},
310  #else
311      {DIAG_SYSMON_CONT_ID,            DIAG_SYSMON_CYCLICTASK,  20, DIAG_RECORDING_DISABLED,
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_DISABLED, dummyfu2},
312  #endif
313
314  #if BUILD_MODULE_ENABLE_ILCK == 1
315      {DIAG_SYSMON_ILCK_ID,            DIAG_SYSMON_CYCLICTASK,  20, DIAG_RECORDING_ENABLED,
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_ENABLED, dummyfu2},
316  #else
317      {DIAG_SYSMON_ILCK_ID,            DIAG_SYSMON_CYCLICTASK,  20, DIAG_RECORDING_DISABLED,
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_DISABLED, dummyfu2},
318  #endif
319      {DIAG_SYSMON_LTC_ID,             DIAG_SYSMON_CYCLICTASK,   5, DIAG_RECORDING_ENABLED,
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_ENABLED, dummyfu2},
320
321  #if BUILD_MODULE_ENABLE_ISOGUARD == 1
322      {DIAG_SYSMON_ISOGUARD_ID,        DIAG_SYSMON_CYCLICTASK, 400, DIAG_RECORDING_ENABLED,
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_ENABLED, dummyfu2},
323  #else
324      {DIAG_SYSMON_ISOGUARD_ID,        DIAG_SYSMON_CYCLICTASK, 400, DIAG_RECORDING_DISABLED,
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_DISABLED, dummyfu2},
325  #endif
326
327      {DIAG_SYSMON_CANS_ID,            DIAG_SYSMON_CYCLICTASK,  20, DIAG_RECORDING_ENABLED,
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_ENABLED, dummyfu2},
328      {DIAG_SYSMON_APPL_CYCLIC_1ms,    DIAG_SYSMON_CYCLICTASK,  20, DIAG_RECORDING_ENABLED,
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_ENABLED, dummyfu2},
329      {DIAG_SYSMON_APPL_CYCLIC_10ms,   DIAG_SYSMON_CYCLICTASK,  20, DIAG_RECORDING_ENABLED,
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_ENABLED, dummyfu2},
330      {DIAG_SYSMON_APPL_CYCLIC_100ms,  DIAG_SYSMON_CYCLICTASK, 200, DIAG_RECORDING_ENABLED,
     DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR, DIAG_ENABLED, dummyfu2},
331  };
332
333
334  DIAG_DEV_s  diag_dev = {
335      .nr_of_ch    = sizeof(diag_ch_cfg)/sizeof(DIAG_CH_CFG_s),
336      .ch_cfg      = &diag_ch_cfg[0],
337  };
338
```

```
336  typedef struct {
337      DIAG_SYSMON_MODULE_ID_e id;
338      DIAG_SYSMON_TYPE_e type;
339      uint16_t threshold;
340      DIAG_TYPE_RECORDING_e enablerecording;
341      DIAG_SYSMON_HANDLING_TYPE_e handlingtype;
342      DIAG_ENABLE_STATE_e state;
343      void (*callbackfunc)(DIAG_SYSMON_MODULE_ID_e);
344  } DIAG_SYSMON_CH_CFG_s;
```

```
266  typedef enum {
267      DIAG_SYSMON_HANDLING_DONOTHING,
268      DIAG_SYSMON_HANDLING_SWITCHOFFCONTACTOR,
269  } DIAG_SYSMON_HANDLING_TYPE_e;
```

```
/*================== Static Function Implementations ========================*/
/**
 * @brief  dummy callback function of diagnosis events
 */
void dummyfu(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
    /* Dummy function -> empty */
}

/**
 * @brief  dummy callback function of system monitoring error events
 */
void dummyfu2(DIAG_SYSMON_MODULE_ID_e ch_id) {
    /* Dummy function -> empty */
}

/**
 * @brief  diagnosis callback function for overvoltage events
 */
static void DIAG_overvoltage(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
    if (ch_id == DIAG_CH_CELLVOLTAGE_OVERVOLTAGE_MSL) {
        if (event == DIAG_EVENT_RESET) {
            msl_flags.over_voltage = 0;
        } else if (...) {
        if (event == DIAG_EVENT_NOK) {
            msl_flags.over_voltage = 1;
        }
    } else if (ch_id == DIAG_CH_CELLVOLTAGE_OVERVOLTAGE_RSL) {
        if (event == DIAG_EVENT_RESET) {
            rsl_flags.over_voltage = 0;
        }
        if (event == DIAG_EVENT_NOK) {
            rsl_flags.over_voltage = 1;
        }
    } else if (ch_id == DIAG_CH_CELLVOLTAGE_OVERVOLTAGE_MOL) {
        if (event == DIAG_EVENT_RESET) {
            mol_flags.over_voltage = 0;
        }
        if (event == DIAG_EVENT_NOK) {
            mol_flags.over_voltage = 1;
        }
    }
}

/**
 * @brief  diagnosis callback function for undervoltage events
 */
static void DIAG_undervoltage(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
    //return; // JHL to disable the undervoltage test.
    // Results: Cannot stop the under voltage error by by-passing this function.
    if (ch_id == DIAG_CH_CELLVOLTAGE_UNDERVOLTAGE_MSL) {
        if (event == DIAG_EVENT_RESET) {
            msl_flags.under_voltage = 0;
```

```
391             }
392             if (event == DIAG_EVENT_NOK) {
393                 msl_flags.under_voltage = 1;
394             }
395         } else if (ch_id == DIAG_CH_CELLVOLTAGE_UNDERVOLTAGE_RSL) {
396             if (event == DIAG_EVENT_RESET) {
397                 rsl_flags.under_voltage = 0;
398             }
399             if (event == DIAG_EVENT_NOK) {
400                 rsl_flags.under_voltage = 1;
401             }
402         } else if (ch_id == DIAG_CH_CELLVOLTAGE_UNDERVOLTAGE_MOL) {
403             if (event == DIAG_EVENT_RESET) {
404                 mol_flags.under_voltage = 0;
405             }
406             if (event == DIAG_EVENT_NOK) {
407                 mol_flags.under_voltage = 1;
408             }
409         }
410     }
411
412     /**
413      * @brief  diagnosis callback function for overtemperature charge events
414      */
415     static void DIAG_overtemperature_charge(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
416         if (ch_id == DIAG_CH_TEMP_OVERTEMPERATURE_CHARGE_MSL) {
417             if (event == DIAG_EVENT_RESET) {
418                 msl_flags.over_temperature_charge = 0;
419             }
420             if (event == DIAG_EVENT_NOK) {
421                 msl_flags.over_temperature_charge = 1;
422             }
423         } else if (ch_id == DIAG_CH_TEMP_OVERTEMPERATURE_CHARGE_RSL) {
424             if (event == DIAG_EVENT_RESET) {
425                 rsl_flags.over_temperature_charge = 0;
426             }
427             if (event == DIAG_EVENT_NOK) {
428                 rsl_flags.over_temperature_charge = 1;
429             }
430         } else if (ch_id == DIAG_CH_TEMP_OVERTEMPERATURE_CHARGE_MOL) {
431             if (event == DIAG_EVENT_RESET) {
432                 mol_flags.over_temperature_charge = 0;
433             }
434             if (event == DIAG_EVENT_NOK) {
435                 mol_flags.over_temperature_charge = 1;
436             }
437         }
438     }
439
440     /**
441      * @brief  diagnosis callback function for overtemperature discharge events
442      */
```

```
443  static void DIAG_overtemperature_discharge(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
444      if (ch_id == DIAG_CH_TEMP_OVERTEMPERATURE_DISCHARGE_MSL) {
445          if (event == DIAG_EVENT_RESET) {
446              msl_flags.over_temperature_discharge = 0;
447          }
448          if (event == DIAG_EVENT_NOK) {
449              msl_flags.over_temperature_discharge = 1;
450          }
451      } else if (ch_id == DIAG_CH_TEMP_OVERTEMPERATURE_DISCHARGE_RSL) {
452          if (event == DIAG_EVENT_RESET) {
453              rsl_flags.over_temperature_discharge = 0;
454          }
455          if (event == DIAG_EVENT_NOK) {
456              rsl_flags.over_temperature_discharge = 1;
457          }
458      } else if (ch_id == DIAG_CH_TEMP_OVERTEMPERATURE_DISCHARGE_MOL) {
459          if (event == DIAG_EVENT_RESET) {
460              mol_flags.over_temperature_discharge = 0;
461          }
462          if (event == DIAG_EVENT_NOK) {
463              mol_flags.over_temperature_discharge = 1;
464          }
465      }
466  }
467
468  /**
469   * @brief  diagnosis callback function for undertemperature charge events
470   */
471  static void DIAG_undertemperature_charge(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
472      if (ch_id == DIAG_CH_TEMP_UNDERTEMPERATURE_CHARGE_MSL) {
473          if (event == DIAG_EVENT_RESET) {
474              msl_flags.under_temperature_charge = 0;
475          }
476          if (event == DIAG_EVENT_NOK) {
477              msl_flags.under_temperature_charge = 1;
478          }
479      } else if (ch_id == DIAG_CH_TEMP_UNDERTEMPERATURE_CHARGE_RSL) {
480          if (event == DIAG_EVENT_RESET) {
481              rsl_flags.under_temperature_charge = 0;
482          }
483          if (event == DIAG_EVENT_NOK) {
484              rsl_flags.under_temperature_charge = 1;
485          }
486      } else if (ch_id == DIAG_CH_TEMP_UNDERTEMPERATURE_CHARGE_MOL) {
487          if (event == DIAG_EVENT_RESET) {
488              mol_flags.under_temperature_charge = 0;
489          }
490          if (event == DIAG_EVENT_NOK) {
491              mol_flags.under_temperature_charge = 1;
492          }
493      }
494  }
```

```
495
496 /**
497  * @brief  diagnosis callback function for undertemperature discharge events
498  */
499 static void DIAG_undertemperature_discharge(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
500     if (ch_id == DIAG_CH_TEMP_UNDERTEMPERATURE_DISCHARGE_MSL) {
501         if (event == DIAG_EVENT_RESET) {
502             msl_flags.under_temperature_discharge = 0;
503         }
504         if (event == DIAG_EVENT_NOK) {
505             msl_flags.under_temperature_discharge = 1;
506         }
507     } else if (ch_id == DIAG_CH_TEMP_UNDERTEMPERATURE_DISCHARGE_RSL) {
508         if (event == DIAG_EVENT_RESET) {
509             rsl_flags.under_temperature_discharge = 0;
510         }
511         if (event == DIAG_EVENT_NOK) {
512             rsl_flags.under_temperature_discharge = 1;
513         }
514     } else if (ch_id == DIAG_CH_TEMP_UNDERTEMPERATURE_DISCHARGE_MOL) {
515         if (event == DIAG_EVENT_RESET) {
516             mol_flags.under_temperature_discharge = 0;
517         }
518         if (event == DIAG_EVENT_NOK) {
519             mol_flags.under_temperature_discharge = 1;
520         }
521     }
522 }
523
524 /**
525  * @brief  diagnosis callback function for overcurrent charge events
526  */
527 static void DIAG_overcurrent_charge(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
528     switch (ch_id) {
529     case DIAG_CH_OVERCURRENT_CHARGE_CELL_MSL:
530         if (event == DIAG_EVENT_RESET) {
531             msl_flags.over_current_charge_cell = 0;
532         } else if (event == DIAG_EVENT_NOK) {
533             msl_flags.over_current_charge_cell = 1;
534         } else {
535             /* no relevant event, do nothing */
536         }
537         break;
538     case DIAG_CH_OVERCURRENT_CHARGE_CELL_RSL:
539         if (event == DIAG_EVENT_RESET) {
540             rsl_flags.over_current_charge_cell = 0;
541         } else if (event == DIAG_EVENT_NOK) {
542             rsl_flags.over_current_charge_cell = 1;
543         } else {
544             /* no relevant event, do nothing */
545         }
546         break;
```

```
547         case DIAG_CH_OVERCURRENT_CHARGE_CELL_MOL:
548             if (event == DIAG_EVENT_RESET) {
549                 mol_flags.over_current_charge_cell = 0;
550             } else if (event == DIAG_EVENT_NOK) {
551                 mol_flags.over_current_charge_cell = 1;
552             } else {
553                 /* no relevant event, do nothing */
554             }
555             break;
556         case DIAG_CH_OVERCURRENT_CHARGE_PL0_MSL:
557             if (event == DIAG_EVENT_RESET) {
558                 msl_flags.over_current_charge_pl0 = 0;
559             } else if (event == DIAG_EVENT_NOK) {
560                 msl_flags.over_current_charge_pl0 = 1;
561             } else {
562                 /* no relevant event, do nothing */
563             }
564             break;
565         case DIAG_CH_OVERCURRENT_CHARGE_PL0_RSL:
566             if (event == DIAG_EVENT_RESET) {
567                 rsl_flags.over_current_charge_pl0 = 0;
568             } else if (event == DIAG_EVENT_NOK) {
569                 rsl_flags.over_current_charge_pl0 = 1;
570             } else {
571                 /* no relevant event, do nothing */
572             }
573             break;
574         case DIAG_CH_OVERCURRENT_CHARGE_PL0_MOL:
575             if (event == DIAG_EVENT_RESET) {
576                 mol_flags.over_current_charge_pl0 = 0;
577             } else if (event == DIAG_EVENT_NOK) {
578                 mol_flags.over_current_charge_pl0 = 1;
579             } else {
580                 /* no relevant event, do nothing */
581             }
582             break;
583         case DIAG_CH_OVERCURRENT_CHARGE_PL1_MSL:
584             if (event == DIAG_EVENT_RESET) {
585                 msl_flags.over_current_charge_pl1 = 0;
586             } else if (event == DIAG_EVENT_NOK) {
587                 msl_flags.over_current_charge_pl1 = 1;
588             } else {
589                 /* no relevant event, do nothing */
590             }
591             break;
592         case DIAG_CH_OVERCURRENT_CHARGE_PL1_RSL:
593             if (event == DIAG_EVENT_RESET) {
594                 rsl_flags.over_current_charge_pl1 = 0;
595             } else if (event == DIAG_EVENT_NOK) {
596                 rsl_flags.over_current_charge_pl1 = 1;
597             } else {
598                 /* no relevant event, do nothing */
```

```
599             }
600             break;
601         case DIAG_CH_OVERCURRENT_CHARGE_PL1_MOL:
602             if (event == DIAG_EVENT_RESET) {
603                 mol_flags.over_current_charge_pl1 = 0;
604             } else if (event == DIAG_EVENT_NOK) {
605                 mol_flags.over_current_charge_pl1 = 1;
606             } else {
607                 /* no relevant event, do nothing */
608             }
609             break;
610         case DIAG_CH_OVERCURRENT_PL_NONE:
611             if (event == DIAG_EVENT_RESET) {
612                 error_flags.currentOnOpenPowerline = 0;
613             } else if (event == DIAG_EVENT_NOK) {
614                 error_flags.currentOnOpenPowerline = 1;
615             } else {
616                 /* no relevant event, do nothing */
617             }
618             break;
619         default:
620             /* no relevant channel, do nothing */
621             break;
622         }
623     }
624
625     /**
626      * @brief  diagnosis callback function for overcurrent discharge events
627      */
628     static void DIAG_overcurrent_discharge(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
629         switch (ch_id) {
630         case DIAG_CH_OVERCURRENT_DISCHARGE_CELL_MSL:
631             if (event == DIAG_EVENT_RESET) {
632                 msl_flags.over_current_discharge_cell = 0;
633             } else if (event == DIAG_EVENT_NOK) {
634                 msl_flags.over_current_discharge_cell = 1;
635             } else {
636                 /* no relevant event, do nothing */
637             }
638             break;
639         case DIAG_CH_OVERCURRENT_DISCHARGE_CELL_RSL:
640             if (event == DIAG_EVENT_RESET) {
641                 rsl_flags.over_current_discharge_cell = 0;
642             } else if (event == DIAG_EVENT_NOK) {
643                 rsl_flags.over_current_discharge_cell = 1;
644             } else {
645                 /* no relevant event, do nothing */
646             }
647             break;
648         case DIAG_CH_OVERCURRENT_DISCHARGE_CELL_MOL:
649             if (event == DIAG_EVENT_RESET) {
650                 mol_flags.over_current_discharge_cell = 0;
```

```
651             } else if (event == DIAG_EVENT_NOK) {
652                 mol_flags.over_current_discharge_cell = 1;
653             } else {
654                 /* no relevant event, do nothing */
655             }
656             break;
657         case DIAG_CH_OVERCURRENT_DISCHARGE_PL0_MSL:
658             if (event == DIAG_EVENT_RESET) {
659                 msl_flags.over_current_discharge_pl0 = 0;
660             } else if (event == DIAG_EVENT_NOK) {
661                 msl_flags.over_current_discharge_pl0 = 1;
662             } else {
663                 /* no relevant event, do nothing */
664             }
665             break;
666         case DIAG_CH_OVERCURRENT_DISCHARGE_PL0_RSL:
667             if (event == DIAG_EVENT_RESET) {
668                 rsl_flags.over_current_discharge_pl0 = 0;
669             } else if (event == DIAG_EVENT_NOK) {
670                 rsl_flags.over_current_discharge_pl0 = 1;
671             } else {
672                 /* no relevant event, do nothing */
673             }
674             break;
675         case DIAG_CH_OVERCURRENT_DISCHARGE_PL0_MOL:
676             if (event == DIAG_EVENT_RESET) {
677                 mol_flags.over_current_discharge_pl0 = 0;
678             } else if (event == DIAG_EVENT_NOK) {
679                 mol_flags.over_current_discharge_pl0 = 1;
680             } else {
681                 /* no relevant event, do nothing */
682             }
683             break;
684         case DIAG_CH_OVERCURRENT_DISCHARGE_PL1_MSL:
685             if (event == DIAG_EVENT_RESET) {
686                 msl_flags.over_current_discharge_pl1 = 0;
687             } else if (event == DIAG_EVENT_NOK) {
688                 msl_flags.over_current_discharge_pl1 = 1;
689             } else {
690                 /* no relevant event, do nothing */
691             }
692             break;
693         case DIAG_CH_OVERCURRENT_DISCHARGE_PL1_RSL:
694             if (event == DIAG_EVENT_RESET) {
695                 rsl_flags.over_current_discharge_pl1 = 0;
696             } else if (event == DIAG_EVENT_NOK) {
697                 rsl_flags.over_current_discharge_pl1 = 1;
698             } else {
699                 /* no relevant event, do nothing */
700             }
701             break;
702         case DIAG_CH_OVERCURRENT_DISCHARGE_PL1_MOL:
```

```
703             if (event == DIAG_EVENT_RESET) {
704                 mol_flags.over_current_discharge_pl1 = 0;
705             } else if (event == DIAG_EVENT_NOK) {
706                 mol_flags.over_current_discharge_pl1 = 1;
707             } else {
708                 /* no relevant event, do nothing */
709             }
710             break;
711         default:
712             /* no relevant channel, do nothing */
713             break;
714         }
715     }
716 
717     /**
718      * @brief  diagnosis callback function for can related events
719      */
720     void DIAG_error_cantiming(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
721         if (ch_id == DIAG_CH_CAN_TIMING) {
722             if (event == DIAG_EVENT_RESET) {
723                 error_flags.can_timing = 0;
724             }
725             if (event == DIAG_EVENT_NOK) {
726                 error_flags.can_timing = 1;
727             }
728         } else if (ch_id == DIAG_CH_CAN_CC_RESPONDING) {
729             if (event == DIAG_EVENT_RESET) {
730                 error_flags.can_timing_cc = 0;
731             }
732             if (event == DIAG_EVENT_NOK) {
733                 error_flags.can_timing_cc = 1;
734             }
735         }
736     }
737 
738     /**
739      * @brief  diagnosis callback function for current sensor related events
740      */
741     void DIAG_error_cancurrentsensor(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
742         if (ch_id == DIAG_CH_CURRENT_SENSOR_RESPONDING) {
743             if (event == DIAG_EVENT_RESET) {
744                 error_flags.currentsensorresponding = 0;
745             }
746             if (event == DIAG_EVENT_NOK) {
747                 error_flags.currentsensorresponding = 1;
748             }
749         }
750     }
751 
752     /**
753      * @brief  diagnosis callback function for LTC module related events
754      */
```

```
755  static void DIAG_error_ltc(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
756      if (ch_id == DIAG_CH_LTC_SPI) {
757          if (event == DIAG_EVENT_RESET) {
758              error_flags.spi_error = 0;
759          }
760          if (event == DIAG_EVENT_NOK) {
761              error_flags.spi_error = 1;
762          }
763      } else if (ch_id == DIAG_CH_LTC_PEC) {
764          if (event == DIAG_EVENT_RESET) {
765              error_flags.crc_error = 0;
766          }
767          if (event == DIAG_EVENT_NOK) {
768              error_flags.crc_error = 1;
769          }
770      } else if (ch_id == DIAG_CH_LTC_MUX) {
771          if (event == DIAG_EVENT_RESET) {
772              error_flags.mux_error = 0;
773          }
774          if (event == DIAG_EVENT_NOK) {
775              error_flags.mux_error = 1;
776          }
777      }  else if (ch_id == DIAG_CH_LTC_CONFIG) {
778          if (event == DIAG_EVENT_RESET) {
779              error_flags.ltc_config_error = 0;
780          }
781          if (event == DIAG_EVENT_NOK) {
782              error_flags.ltc_config_error = 1;
783          }
784      }
785  }
786
787  /**
788   * @brief  diagnosis callback function for contactor feedback events
789   */
790  void DIAG_cont_feedback(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
791      if (ch_id == DIAG_CH_CONTACTOR_MAIN_PLUS_FEEDBACK) {
792          if (event == DIAG_EVENT_RESET) {
793              error_flags.main_plus = 0;
794          }
795          if (event == DIAG_EVENT_NOK) {
796              error_flags.main_plus = 1;
797          }
798      } else if (ch_id == DIAG_CH_CONTACTOR_MAIN_MINUS_FEEDBACK) {
799          if (event == DIAG_EVENT_RESET) {
800              error_flags.main_minus = 0;
801          }
802          if (event == DIAG_EVENT_NOK) {
803              error_flags.main_minus = 1;
804          }
805      } else if (ch_id == DIAG_CH_CONTACTOR_PRECHARGE_FEEDBACK) {
806          if (event == DIAG_EVENT_RESET) {
```

```
807             error_flags.precharge = 0;
808         }
809         if (event == DIAG_EVENT_NOK) {
810             error_flags.precharge = 1;
811         }
812     } else if (ch_id == DIAG_CH_CONTACTOR_CHARGE_MAIN_PLUS_FEEDBACK) {
813         if (event == DIAG_EVENT_RESET) {
814             error_flags.charge_main_plus = 0;
815         }
816         if (event == DIAG_EVENT_NOK) {
817             error_flags.charge_main_plus = 1;
818         }
819     } else if (ch_id == DIAG_CH_CONTACTOR_CHARGE_MAIN_MINUS_FEEDBACK) {
820         if (event == DIAG_EVENT_RESET) {
821             error_flags.charge_main_minus = 0;
822         }
823         if (event == DIAG_EVENT_NOK) {
824             error_flags.charge_main_minus = 1;
825         }
826     } else if (ch_id == DIAG_CH_CONTACTOR_CHARGE_PRECHARGE_FEEDBACK) {
827         if (event == DIAG_EVENT_RESET) {
828             error_flags.charge_precharge = 0;
829         }
830         if (event == DIAG_EVENT_NOK) {
831             error_flags.charge_precharge = 1;
832         }
833     }
834 }
835
836 /**
837  * @brief  diagnosis callback function for fuse related events
838  */
839 void DIAG_error_fuseState(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
840     if (ch_id == DIAG_CH_FUSE_STATE_NORMAL) {
841         if (event == DIAG_EVENT_RESET) {
842             error_flags.fuse_state_normal = 0;
843         }
844         if (event == DIAG_EVENT_NOK) {
845             error_flags.fuse_state_normal = 1;
846         }
847     } else if (ch_id == DIAG_CH_FUSE_STATE_CHARGE) {
848         if (event == DIAG_EVENT_RESET) {
849             error_flags.fuse_state_charge = 0;
850         }
851         if (event == DIAG_EVENT_NOK) {
852             error_flags.fuse_state_charge = 1;
853         }
854     }
855 }
856
857 /**
858  * @brief  diagnosis callback function for interlock events
```

```
 */
void DIAG_error_interlock(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
    if (ch_id == DIAG_CH_INTERLOCK_FEEDBACK) {
        if (event == DIAG_EVENT_RESET) {
            error_flags.interlock = 0;
        }
        if (event == DIAG_EVENT_NOK) {
            error_flags.interlock = 1;
        }
    }
}

/**
 * @brief  diagnosis callback function for insulation events
 */
void DIAG_error_insulation(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
    if (ch_id == DIAG_CH_INSULATION_ERROR) {
        if (event == DIAG_EVENT_RESET) {
            error_flags.insulation_error = 0;
        }
        if (event == DIAG_EVENT_NOK) {
            error_flags.insulation_error = 1;
        }
    }
}

/**
 * @brief  diagnosis callback function for MCU die temperature events
 */
void DIAG_error_MCUdieTemperature(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
    if (ch_id == DIAG_CH_ERROR_MCU_DIE_TEMPERATURE) {
        if (event == DIAG_EVENT_RESET) {
            error_flags.mcuDieTemperature = 0;
        }
        if (event == DIAG_EVENT_NOK) {
            error_flags.mcuDieTemperature = 1;
        }
    }
}

/**
 * @brief  diagnosis callback function for coin cell voltage events
 */
void DIAG_error_coinCellVoltage(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
    if (ch_id == DIAG_CH_LOW_COIN_CELL_VOLTAGE) {
        if (event  ==  DIAG_EVENT_RESET) {
            error_flags.coinCellVoltage &= 0xFE;
        }
        if (event == DIAG_EVENT_NOK) {
            error_flags.coinCellVoltage |= 0x01;
        }
    } else if (ch_id == DIAG_CH_CRIT_LOW_COIN_CELL_VOLTAGE) {
```

```
911             if (event  ==  DIAG_EVENT_RESET) {
912                 error_flags.coinCellVoltage &= 0xFD;
913             }
914             if (event == DIAG_EVENT_NOK) {
915                 error_flags.coinCellVoltage |= 0x02;
916             }
917         }
918     }
919
920     /**
921      * @brief  diagnosis callback function for plausibility events
922      */
923     void DIAG_error_plausibility_check(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
924         if (ch_id == DIAG_CH_PLAUSIBILITY_CELL_VOLTAGE) {
925             if (event  ==  DIAG_EVENT_RESET) {
926                 error_flags.plausibilityCheck &= 0xFE;
927             }
928             if (event == DIAG_EVENT_NOK) {
929                 error_flags.plausibilityCheck |= 0x01;
930             }
931         } else if (ch_id == DIAG_CH_PLAUSIBILITY_CELL_TEMP) {
932             if (event  ==  DIAG_EVENT_RESET) {
933                 error_flags.plausibilityCheck &= 0xFD;
934             }
935             if (event == DIAG_EVENT_NOK) {
936                 error_flags.plausibilityCheck |= 0x02;
937             }
938         } else if (ch_id == DIAG_CH_PLAUSIBILITY_PACK_VOLTAGE) {
939             if (event  ==  DIAG_EVENT_RESET) {
940                 error_flags.plausibilityCheck &= 0xFB;
941             }
942             if (event == DIAG_EVENT_NOK) {
943                 error_flags.plausibilityCheck |= 0x04;
944             }
945         }
946     }
947
948     /**
949      * @brief  diagnosis callback function for open-wire events
950      */
951     void DIAG_error_openWire(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
952         if (ch_id == DIAG_CH_OPEN_WIRE) {
953             if (event  ==  DIAG_EVENT_RESET) {
954                 error_flags.open_wire = 0;
955             }
956             if (event == DIAG_EVENT_NOK) {
957                 error_flags.open_wire = 1;
958             }
959         }
960     }
961
962     /**
```

```
963     * @brief  diagnosis callback function for deep-discharge related events
964     */
965    void DIAG_error_deep_discharge_detected(DIAG_CH_ID_e ch_id, DIAG_EVENT_e event) {
966        if (ch_id == DIAG_CH_DEEP_DISCHARGE_DETECTED) {
967            if (event == DIAG_EVENT_RESET) {
968                error_flags.deepDischargeDetected = 0;
969                RTC_DEEP_DISCHARGE_DETECTED = 0;
970            }
971            if (event == DIAG_EVENT_NOK) {
972                error_flags.deepDischargeDetected = 1;
973                RTC_DEEP_DISCHARGE_DETECTED = 1;
974            }
975        }
976    }
977
978    /*================== Extern Function Implementations ========================*/
979    void DIAG_updateFlags(void) {
980        DB_WriteBlock(&error_flags, DATA_BLOCK_ID_ERRORSTATE);
981        DB_WriteBlock(&msl_flags, DATA_BLOCK_ID_MSL);
982        DB_WriteBlock(&rsl_flags, DATA_BLOCK_ID_RSL);
983        DB_WriteBlock(&mol_flags, DATA_BLOCK_ID_MOL);
984    }
985
```