```
                Color code for code review:
                  * Blue for normal notes
                  * Dark green for proposed changes
                  * Red for bugs
                  * Yellow or cyon for highlights.
/**
 *
 * @copyright &copy; 2010 - 2019, 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    contactor.h
 * @author  foxBMS Team
 * @date    23.09.2015 (date of creation)
 * @ingroup DRIVERS
 * @prefix  CONT
 *
 * @brief   Headers for the driver for the contactors.
 *
 */

#ifndef CONTACTOR_H_
#define CONTACTOR_H_

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

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

/*================== Constant and Variable Definitions =====================*/

```

All function prototypes on this page should be moved to the end of the file, preferrable in alphabetical order.

```
63 /*================== Function Prototypes ==================================*/
64 /**
65  * @brief   Checks the configuration of the contactor-module
66  *
67  * @return  retVal (type: STD_RETURN_TYPE_e)
68  */
69 extern STD_RETURN_TYPE_e CONT_Init(void);   !!! Not defined and never used. Placeholder?
70
71 /**
72  * @brief   Gets the latest value (TRUE, FALSE) the contactors were set to.
73  *
74  * @param   contactor (type: CONT_NAMES_e)
75  *
76  * @return  returns CONT_SWITCH_OFF or CONT_SWITCH_ON
77  */
78 extern CONT_ELECTRICAL_STATE_TYPE_e CONT_GetContactorSetValue(CONT_NAMES_e
    contactor);
79
80
81 /**
82  * @brief   Reads the feedback pin of every contactor and returns its current value
83  *          (CONT_SWITCH_OFF/CONT_SWITCH_ON).
84  *
85  * @details If the contactor has a feedback pin the measured feedback is returned.
    If the contactor
86  *          has no feedback pin, it is assumed that after a certain time the
    contactor has reached
87  *          the requested state.
88  *
89  * @param   contactor (type: CONT_NAMES_e)
90  *
91  * @return  measuredContactorState (type: CONT_ELECTRICAL_STATE_TYPE_s)
92  */
93 extern CONT_ELECTRICAL_STATE_TYPE_e CONT_GetContactorFeedback(CONT_NAMES_e
    contactor);
94
95 /**
96  * @brief   Reads the feedback pins of all contactors and updates the cont_contactor_states[]
    contactors_cfg[] array with
97  *          their current states.
98  *
99  * @return  Returns E_OK if all feedbacks could be acquired (type:
    STD_RETURN_TYPE_e)
100  */
101 extern STD_RETURN_TYPE_e CONT_AcquireContactorFeedbacks(void);
102
103 /**
104  * @brief   Sets the contactor state to its requested state, if the contactor is at
    that time not
105  *          in the requested state.
106  *
107  * @details If the new state was already requested, but not reached (meaning the
    measured feedback
108  *          does not return the requested state), there are two states: it can be
    still ok (E_OK),
109  *          because the contactor has some time left to get physically in the
    requested state
110  *          (passed time since the request is lower than the limit) or it can be
    not ok (E_NOT_OK),
111  *          because there is timing violation, i.e. the contactor has surpassed the
    maximum time
112  *          for getting in the requested state. It returns E_OK if the requested
    state was
```

All function prototypes on this page should be moved to the end of the file, preferrable in alphabetical order.

```
 *          successfully set or if the contactor was at the requested state before.
 *
 * @param   contactor (type: CONT_NAMES_e)
 * @param   requestedContactorState (type: CONT_ELECTRICAL_STATE_TYPE_s)
 *
 * @return  retVal (type: STD_RETURN_TYPE_e)
 */
extern STD_RETURN_TYPE_e CONT_SetContactorState(CONT_NAMES_e contactor,
CONT_ELECTRICAL_STATE_TYPE_s requestedContactorState);
                            e
STD_RETURN_TYPE_e  CONT_SetContractorState_pulse(CONT_NAMES_e contactor)
/**
 * @brief   Iterates over the contactor array and switches all contactors off
 *
 * @return  E_OK if all contactors were opened, E_NOT_OK if not all contactors
could be opened
 *          (type: STD_RETURN_TYPE_e)
 */
extern STD_RETURN_TYPE_e CONT_SwitchAllContactorsOff(void);


/*================== Function Implementations ==========================*/


/*================== Constant and Variable Definitions ====================*/

     Macros and type definitions should be moved to the beginning of the file.
/**
 * States of the CONT state machine
 */
typedef enum {
    /* Init-Sequence */
    CONT_STATEMACH_UNINITIALIZED            = 0,    /*!<    */
    CONT_STATEMACH_INITIALIZATION           = 1,    /*!<    */
    CONT_STATEMACH_INITIALIZED              = 2,    /*!<    */
    CONT_STATEMACH_IDLE                     = 3,    /*!<    */
    CONT_STATEMACH_STANDBY                  = 4,    /*!<    */
    CONT_STATEMACH_PRECHARGE                = 5,    /*!<    */
    CONT_STATEMACH_NORMAL                   = 6,    /*!<    */
    CONT_STATEMACH_CHARGE_PRECHARGE         = 7,    /*!<    */
    CONT_STATEMACH_CHARGE                   = 8,    /*!<    */
                                                    CONT_STATMACH_ENGINE_PRECHARGE   = 9,
                                                    CONT_STATMACH_ENGINE             = 10,
    CONT_STATEMACH_UNDEFINED                = 20,   /*!< undefined state
                      */
    CONT_STATEMACH_RESERVED1                = 0x80, /*!< reserved state
                       */
    CONT_STATEMACH_ERROR                    = 0xF0, /*!< Error-State:  */
} CONT_STATEMACH_e;

/**
 * Substates of the CONT state machine
 */
typedef enum {
    CONT_ENTRY                                  = 0,    /*!< Substate entry state
      */
    CONT_OPEN_FIRST_CONTACTOR                   = 1,    /*!< Open-sequence: first
contactor */
    CONT_OPEN_SECOND_CONTACTOR_MINUS            = 2,    /*!< Open-sequence:
second contactor */
    CONT_OPEN_SECOND_CONTACTOR_PLUS             = 3,    /*!< Open-sequence:
second contactor */
    CONT_STANDBY                                = 4,    /*!< Substate stanby */
    CONT_PRECHARGE_CLOSE_MINUS                  = 5,    /*!< Begin of precharge
sequence: close main minus */
```

```
166     CONT_PRECHARGE_CLOSE_PRECHARGE                = 6,    /*!< Next step of
precharge sequence: close precharge */
167     CONT_PRECHARGE_CLOSE_PLUS                     = 7,    /*!< Next step of
precharge sequence: close main plus */
168     CONT_PRECHARGE_CHECK_VOLTAGES_TIME            = 8,    /*!< Next step of
precharge sequence: check if voltages OK */
169     CONT_PRECHARGE_OPEN_PRECHARGE                 = 9,    /*!< Next step of
precharge sequence: open precharge */
170     CONT_ERROR                                    = 10,   /*!< Error state */
171 } CONT_STATEMACH_SUB_e;
172
173 /**
174  * State requests for the CONT statemachine
175  */
176 typedef enum {   We should not use this Init_Request. This request should be done automatically.
177     CONT_STATE_INIT_REQUEST                   = CONT_STATEMACH_INITIALIZATION,
   /*!<    */
178     CONT_STATE_STANDBY_REQUEST                = CONT_STATEMACH_STANDBY,
      /*!<    */
179     CONT_STATE_NORMAL_REQUEST                 = CONT_STATEMACH_NORMAL,
 /*!<    */
180     CONT_STATE_CHARGE_REQUEST                 = CONT_STATEMACH_CHARGE,
 /*!<    */      CONT_STATMACH_ENGINE_REQUEST = CONT_STATMACH_ENGINE,
181     CONT_STATE_ERROR_REQUEST                  = CONT_STATEMACH_ERROR,   /*!<    */
182     CONT_STATE_NO_REQUEST                     = CONT_STATEMACH_RESERVED1,
   /*!<    */
183 } CONT_STATE_REQUEST_e;
184
185 /**
186  * Possible return values when state requests are made to the CONT statemachine
187  */
188 typedef enum {
189     CONT_OK                                    = 0,    /*!< CONT --> ok
             */
190     CONT_BUSY_OK                               = 1,    /*!< CONT under load --> ok
             */
191     CONT_REQUEST_PENDING                       = 2,    /*!< requested to be executed
             */
192     CONT_REQUEST_IMPOSSIBLE                    = 3,    /*!< requested not possible
             */
193     CONT_ILLEGAL_REQUEST                       = 4,    /*!< Request can not be
executed            */
194     CONT_INIT_ERROR                            = 5,    /*!< Error state: Source:
Initialization    */
195     CONT_OK_FROM_ERROR                         = 6,    /*!< Return from error --> ok
             */
196     CONT_ALREADY_INITIALIZED                   = 30,   /*!< Initialization of LTC
already finished */
197     CONT_ILLEGAL_TASK_TYPE                     = 99,   /*!< Illegal
             */
198 } CONT_RETURN_TYPE_e;
199
200 /**
201  * @brief Names for connected powerlines.
202  */
203 typedef enum CONT_POWER_LINE_e {
204     CONT_POWER_LINE_NONE,    /*!< no power line is connected, contactors are open
          */
205     CONT_POWER_LINE_0,       /*!< power line 0, e.g. used for the power train
          */
206 #if BS_SEPARATE_POWERLINES == 1
207     CONT_POWER_LINE_1,       /*!< power line 1, e.g. used for charging
           */
```

```
208 #endif
209 } CONT_POWER_LINE_e;
210
211 /**
212  * This structure contains all the variables relevant for the CONT state machine.
213  * The user can get the current state of the CONT state machine with this variable
214  */
215 typedef struct {
216     uint16_t timer;                         /*!< time in ms before the state
    machine processes the next state, e.g. in counts of 1ms    */
217     CONT_STATE_REQUEST_e statereq;          /*!< current state request made to the
    state machine                                                */
218     CONT_STATEMACH_e state;                 /*!< state of Driver State Machine
                                                                 */
219     CONT_STATEMACH_SUB_e substate;          /*!< current substate of the state
    machine                                                      */
220     CONT_STATEMACH_e laststate;             /*!< previous state of the state
    machine                                                      */
221     CONT_STATEMACH_SUB_e lastsubstate;      /*!< previous substate of the state
    machine                                                      */
222     uint32_t ErrRequestCounter;             /*!< counts the number of illegal
    requests to the LTC state machine                            */
223     STD_RETURN_TYPE_e initFinished;         /*!< #E_OK if the initialization has
    passed, #E_NOT_OK otherwise                                  */
224     uint16_t OscillationCounter;            /*!< timeout to prevent oscillation of
    contactors                                                   */
225     uint8_t PrechargeTryCounter;            /*!< timeout to prevent oscillation of
    contactors                                                   */
226     uint16_t PrechargeTimeOut;              /*!< time to wait when precharge has
    been closed for voltages to settle                           */
227     uint8_t triggerentry;                   /*!< counter for re-entrance
    protection (function running flag)                           */
228     uint8_t counter;                        /*!< general purpose counter
                                                                 */
229     CONT_POWER_LINE_e activePowerLine;      /*!< tracks the currently connected
    power line                                                   */
230 } CONT_STATE_s;
231
232
233 /*================== Function Prototypes ==================================*/
234
235 /**
236  * @brief   Sets the current state request of the state variable cont_state.
237  *
238  * @details This function is used to make a state request to the state machine,e.g,
    start voltage
239  *          measurement, read result of voltage measurement, re-initialization.
240  *          It calls CONT_CheckStateRequest() to check if the request is valid. The
    state request
241  *          is rejected if is not valid. The result of the check is returned
    immediately, so that
242  *          the requester can act in case it made a non-valid state request.
243  *
244  * @param   state request to set
245  *
246  * @return  #CONT_OK if a state request was made, #CONT_STATE_NO_REQUEST if no
    state request was made
247  */
248 extern CONT_RETURN_TYPE_e CONT_SetStateRequest(CONT_STATE_REQUEST_e statereq);
249
250 /**
251  * @brief   Gets the current state.
252  *
```

```
253  * @details This function is used in the functioning of the CONT state machine.
254  *
255  * @return  current state, taken from #CONT_STATEMACH_e
256  */
257 extern  CONT_STATEMACH_e CONT_GetState(void);
258
259 /**
260  * @brief   Gets the initialization state.
261  *
262  * This function is used for getting the CONT initialization state.
263  *
264  * @return  #E_OK if initialized, otherwise #E_NOT_OK
265  */
266 STD_RETURN_TYPE_e CONT_GetInitializationState(void);
267
268 /**
269  * @brief Returns the active power line.
270  *
271  * This function returns the value of #cont_state.activePowerLine
272  *
273  * @return value of #cont_state.activePowerLine
274  */
275 extern CONT_POWER_LINE_e CONT_GetActivePowerLine(void);
276
277 /**
278  * @brief   Trigger function for the CONT driver state machine.
279  *
280  * @details This function contains the sequence of events in the CONT state
    machine. It must be
281  *          called time-triggered, every 1ms. It exits without effect, if the
    function call is
282  *          a reentrance.
283  */
284 extern void CONT_Trigger(void);
285
286 #endif /* CONTACTOR_H_ */
287
```