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_cfg.h
 * @author  foxBMS Team
 * @date    23.09.2015 (date of creation)
 * @ingroup DRIVERS_CONF
 * @prefix  CONT
 *
 * @brief   Header for the configuration for the driver for the contactors
 *
 */

#ifndef CONTACTOR_CFG_H_
#define CONTACTOR_CFG_H_

/*================== Includes ===============================================*/
#include "general.h"
#include "batterysystem_cfg.h"
#include "io.h"

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

```

```
/**
 * @ingroup CONFIG_CONTACTOR
 * defines the number of bad countings of opening contactors at too
 *  high current
 * \par Type:
 * int
 * \par Default:
 * 10
 * \par Range:
 * [9,11]
*/
#define CONT_NUMBER_OF_BAD_COUNTINGS 10

/**
 * @ingroup CONFIG_CONTACTOR
 * This macro describes the limiting current from the the positive to
 * negative side of the contactor at which a damaging of the
 * contactor occurs. If this limit is exceeded the contactor
 * module makes an entry in the diagnosis module indicating a
 * switching off of the contactors under a bad  condition
 *
 * \par Type:
 * float
 * \par Default:
 * 1.0
 * \par Range:
 * [1.0,2.0]
*/
#define BAD_SWITCHOFF_CURRENT_POS 100000.0f

/**
 * @ingroup CONFIG_CONTACTOR
 * This macro describes the limiting current from the the negative to
 * positive side of the contactor at which a damaging of the
 * contactor occurs. If this limit is exceeded the contactor
 * module makes an entry in the diagnosis module indicating a
 * switching off of the contactors under a bad  condition
 *
 * \par Type:
 * float
 * \par Default:
 * -1.0
 * \par Range:
 * [-2.0,-1.0]
*/
#define BAD_SWITCHOFF_CURRENT_NEG -1000000.0f

/*
 * The number of defines per contactor must be the same as the length
 *  of the array cont_contactors_cfg in contactor_cfg.c
 * Every contactor consists of 1 control pin and 1 feedback pin
 * counting together as 1 contactor.
 * E.g. if you have 1 contactor your define has to be:
 *      #define CONT_MAIN_PLUS_CONTROL      IO_PIN_CONTACTOR_0_CONTROL
 *      #define CONT_MAIN_PLUS_FEEDBACK     IO_PIN_CONTACTOR_0_FEEDBACK
 */
#define CONT_MAIN_PLUS_CONTROL_CLOSE            IO_PIN_CONTACTOR_0_CONTROL
#define CONT_MAIN_PLUS_FEEDBACK                 IO_PIN_CONTACTOR_0_FEEDBACK

#define CONT_PRECHARGE_PLUS_CONTROL             IO_PIN_CONTACTOR_1_CONTROL
#define CONT_PRECHARGE_PLUS_FEEDBACK            IO_PIN_CONTACTOR_1_FEEDBACK

#define CONT_MAIN_MINUS_CONTROL_CLOSE           IO_PIN_CONTACTOR_2_CONTROL
```

Guidelines for the relay control:
* Use the first 4 relay control/feedback lines to close the relays and get the feedbacks on the 4 relays.
* Use the last two relay control/feedback lines to open the latching type relays. No feedback is provided. The feedback lines are just place holder.

```
126 #define CONT_MAIN_MINUS_FEEDBACK                IO_PIN_CONTACTOR_2_FEEDBACK
127
128 #if BS_SEPARATE_POWERLINES == 1
129 #define CONT_CHARGE_MAIN_PLUS_CONTROL_OPEN      IO_PIN_CONTACTOR_3_CONTROL
130 #define CONT_CHARGE_MAIN_PLUS_FEEDBACK2         IO_PIN_CONTACTOR_3_FEEDBACK
131
132 #define CONT_CHARGE_PRECHARGE_PLUS_CONTROL      IO_PIN_CONTACTOR_4_CONTROL
133 #define CONT_CHARGE_PRECHARGE_PLUS_FEEDBACK     IO_PIN_CONTACTOR_4_FEEDBACK
134
135 #define CONT_CHARGE_MAIN_MINUS_CONTROL_OPEN     IO_PIN_CONTACTOR_5_CONTROL
136 #define CONT_CHARGE_MAIN_MINUS_FEEDBACK 2       IO_PIN_CONTACTOR_5_FEEDBACK
137 #endif /* BS_SEPARATE_POWERLINES == 1 */
138 /*
139  * additional possible contactors from the io definition
140 #define CONT_X0_CONTROL                         PIN_CONTACTOR_3_CONTROL
141 #define CONT_X0_FEEDBACK                        PIN_CONTACTOR_3_FEEDBACK
142
143 #define CONT_X1_CONTROL                         PIN_CONTACTOR_4_CONTROL
144 #define CONT_X1_FEEDBACK                        PIN_CONTACTOR_4_FEEDBACK
145
146 #define CONT_X2_CONTROL                         PIN_CONTACTOR_5_CONTROL
147 #define CONT_X2_FEEDBACK                        PIN_CONTACTOR_5_FEEDBACK
148 */
149
150
151 /**
152  * This define MUST represent the cycle time of the task in which context the
153  * functions run, e.g., if the CONT_Trigger() is running in the 10 ms task
154  * then the define must be set to 10.
155  *
156  * This define also sets the minimum time.
157  */
158
159 #define CONT_TASK_CYCLE_CONTEXT_MS (10)
160
161 /**
162  * Counter limit used to prevent contactor oscillation
163  */
164
165 #define CONT_OSCILLATION_LIMIT 500
166
167
168 /**
169  * Number of allowed tries to close contactors
170  */
171 #define CONT_PRECHARGE_TRIES 3
172
173 /**
174  * Delay between open first and second contactor
175  */
176
177 #define CONT_DELAY_BETWEEN_OPENING_CONTACTORS_MS        ((50) *
    (CONT_TASK_CYCLE_CONTEXT_MS))
178
179 /**
180  * Delay after opening second contactor
181  */
182
183 #define CONT_DELAY_AFTER_OPENING_SECOND_CONTACTORS_MS   ((50) *
    (CONT_TASK_CYCLE_CONTEXT_MS))
184
185
186 /**
     * Width of the latching relay control pulses
     */
    #define CONT_LATCHING_PULSE_WIDTH_MS  (100)
    /** Time for precharge
     */
    #define CONT_PRECHARGE_TIME_MS        (2000)
```

```
/**
 * CONT statemachine short time definition in ms
 */

#define CONT_STATEMACH_SHORTTIME_MS
(CONT_TASK_CYCLE_CONTEXT_MS)


/**
 * CONT statemachine time to wait after contactors opened because precharge failed
in ms
                    ???
 */
#define CONT_STATEMACH_TIMEAFTERPRECHARGEFAIL_MS        ((100) *
(CONT_TASK_CYCLE_CONTEXT_MS))


/*================== Main precharge configuration ====================*/

/**
 * Precharge timeout in ms
 */
#define CONT_PRECHARGE_TIMEOUT_MS ((500) * (CONT_TASK_CYCLE_CONTEXT_MS))

/**
 * Delay after closing main minus in ms
 */
#define CONT_STATEMACH_WAIT_AFTER_CLOSING_MINUS_MS ((50) *
(CONT_TASK_CYCLE_CONTEXT_MS))

/**
 * Delay after closing precharge in ms
 */

#define CONT_STATEMACH_WAIT_AFTER_CLOSING_PRECHARGE_MS ((100) *
(CONT_TASK_CYCLE_CONTEXT_MS))

/**
 * Delay after closing main plus in ms
 */
#define CONT_STATEMACH_WAIT_AFTER_CLOSING_PLUS_MS ((100) *
(CONT_TASK_CYCLE_CONTEXT_MS))

/**
 * Delay after opening precharge in ms
 */
#define CONT_STATEMACH_WAIT_AFTER_OPENING_PRECHARGE_MS ((50) *
(CONT_TASK_CYCLE_CONTEXT_MS))

/**
 * @ingroup CONFIG_CONTACTOR
 * \par Type:
 * int
 * \par Default:
 * 1000
 * \par Range:
 * [1000,3000]
 * \par Unit:
 * V
*/
#define CONT_PRECHARGE_VOLTAGE_THRESHOLD_mV     1000  /* mV */

/**
 * @ingroup CONFIG_CONTACTOR
 * \par Type:
```

```
 * int
 * \par Default:
 * 10  50
 * \par Range:
 * [50,500]
 * \par Unit:
 * mA
*/
#define CONT_PRECHARGE_CURRENT_THRESHOLD_mA     50  /* mA */


                    (If there is a separate charge power line)
/*================== Charge precharge configuration =====================*/

/**
 * Charge precharge timeout in ms
 */
#define CONT_CHARGE_PRECHARGE_TIMEOUT_MS ((500) * (CONT_TASK_CYCLE_CONTEXT_MS))

/**
 * Delay after closing charge minus in ms
 */
#define CONT_STATEMACH_CHARGE_WAIT_AFTER_CLOSING_MINUS_MS ((50) *
(CONT_TASK_CYCLE_CONTEXT_MS))

/**
 * Delay after closing charge precharge in ms
 */

#define CONT_STATEMACH_CHARGE_WAIT_AFTER_CLOSING_PRECHARGE_MS ((100) *
(CONT_TASK_CYCLE_CONTEXT_MS))

/**
 * Delay after closing charge plus in ms
 */
#define CONT_STATEMACH_CHARGE_WAIT_AFTER_CLOSING_PLUS_MS ((100) *
(CONT_TASK_CYCLE_CONTEXT_MS))

/**
 * @ingroup CONFIG_CONTACTOR
 * \par Type:
 * int
 * \par Default:
 * 1000
 * \par Range:
 * [1000,3000]
 * \par Unit:
 * V
*/
#define CONT_CHARGE_PRECHARGE_VOLTAGE_THRESHOLD_mV   1000  /* mV */

/**
 * @ingroup CONFIG_CONTACTOR
 * \par Type:
 * int
 * \par Default:
 * 10  50
 * \par Range:
 * [50,500]
 * \par Unit:
 * mA
*/
#define CONT_CHARGE_PRECHARGE_CURRENT_THRESHOLD_mA  50  /* mA */

```

```
303
304 /*================== Constant and Variable Definitions ====================*/
305
306 /**
307  * Symbolic names for contactors' possible states
308  */
309 typedef enum {
310     CONT_SWITCH_OFF     = 0,    /*!< Contactor off         --> Contactor is open
        */
311     CONT_SWITCH_ON      = 1,    /*!< Contactor on          --> Contactor is closed
        */
312     CONT_SWITCH_UNDEF   = 2,    /*!< Contactor undefined   --> Contactor state not
known   */
313 } CONT_ELECTRICAL_STATE_TYPE_s;  The s here should be e.   The type should be
                                                              CONT_BASIC_ELECTRICAL_STATE_e
314
315 /**
316  * Symbolic names for the contactors, which are used in
317  * the contactor_config[] array
318  */
319 typedef enum {
320     CONT_MAIN_PLUS_CLOSE        = 0,    /*!< Main contactor in the positive path of
the powerline      */
321     CONT_PRECHARGE_PLUS         = 1,    /*!< Precharge contactor in the positive
path of the powerline */
322     CONT_MAIN_MINUS_CLOSE       = 2,    /*!< Main contactor in the negative path of
the powerline      */
323 #if BS_SEPARATE_POWERLINES == 1                                  for engine start
324     CONT_CHARGE_MAIN_PLUS  CONT_ENGINE = 3,    /*!< Main contactor in the positive charge
path of the powerline      */                  Counterpart of CONT_MAIN_PLUS_CLOSE
325     CONT_CHARGE_PRECHARGE_PLUS  CONT_MAIN_PLUS_OPEN = 4,    /*!< Precharge contactor in the positive
charge path of the powerline */                  Counterpart of CONT_MAIN_MINUS_CLOSE
326     CONT_CHARGE_MAIN_MINUS  CONT_MAIN_MINUS_OPEN = 5,    /*!< Main contactor in the negative charge
path of the powerline      */
327 #endif /* BS_SEPARATE_POWERLINES == 1 */
328 } CONT_NAMES_e;
329
330 /**
331  * Symbolic names defining the electric behavior of the contactor
332  */
333 typedef enum {
334     CONT_FEEDBACK_NORMALLY_OPEN     = 0,    /*!< Feedback line of a contactor is
normally open      */
335     CONT_FEEDBACK_NORMALLY_CLOSED   = 1,    /*!< Feedback line of a contactor is
normally closed    */
336     CONT_HAS_NO_FEEDBACK    = 0xFF  /*!< Feedback line of the contactor is not used
        */
337 } CONT_FEEDBACK_TYPE_e;
338
339 typedef struct {
340     CONT_ELECTRICAL_STATE_TYPE_s set;
341     CONT_ELECTRICAL_STATE_TYPE_s feedback;
342 } CONT_ELECTRICAL_STATE_s;      The type should be CONT_COMPOND_ELECTRICAL_STATE_s
343
344 typedef struct {
345     IO_PORTS_e control_pin;       Defined in io_package_cfg.h
346     IO_PORTS_e feedback_pin;
347     CONT_FEEDBACK_TYPE_e feedback_pin_type;
348 } CONT_CONFIG_s;
349
350
351 typedef enum {
352     CONT_POWERLINE_NORMAL   = 0,
353 #if BS_SEPARATE_POWERLINES == 1
```

```
354     CONT_POWERLINE_CHARGE    = 1
355 #endif /* BS_SEPARATE_POWERLINES == 1 */
356 } CONT_WHICH_POWERLINE_e;
357                                     Defined in the C version.
358 extern const CONT_CONFIG_s cont_contactors_config[BS_NR_OF_CONTACTORS];
359 extern CONT_ELECTRICAL_STATE_s cont_contactor_states[BS_NR_OF_CONTACTORS];
360
361 /*================== Function Prototypes ==================================*/
362          These functions are moved to contactor.c
363 /**
364  * @brief   Checks if the current limitations are violated
365  *
366  * @return  E_OK if the current limitations are NOT violated, else E_NOT_OK (type:
    STD_RETURN_TYPE_e)
367  */
368 extern STD_RETURN_TYPE_e CONT_CheckPrecharge(CONT_WHICH_POWERLINE_e caller);
369
370
371 /**
372  * Function to check fuse state. Fuse state can only be checked if at least
373  * plus contactors are closed. Furthermore fuse needs to be placed in plus
374  * path and monitored by Isabellenhuette HV measurement
375  *
376  * @return  Returns E_OK if fuse is intact, and E_NOT_OK if fuse is tripped.
377  *
378  */
379 extern STD_RETURN_TYPE_e CONT_CheckFuse(CONT_WHICH_POWERLINE_e caller);
380
381 /*================== Function Implementations ==============================*/
382
383
384 #endif /* CONTACTOR_CFG_H_ */
385
```