```
Color code for code review:
  * Blue for normal notes
  * Dark green for proposed changes
  * Red for bugs
  * Yellow or cyon for highlights.
```

```c
/**
 *
 * @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.c
 * @author  foxBMS Team
 * @date    23.09.2015 (date of creation)
 * @ingroup DRIVERS
 * @prefix  CONT
 *
 * @brief   Driver for the contactors.
 *
 */

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

#include "database.h"
#include "diag.h"
#include "FreeRTOS.h"
#include "task.h"

#if BUILD_MODULE_ENABLE_CONTACTOR == 1     // Extends to the end of the file.
/*================== Macros and Definitions =================================*/
```

```
63     This should be moved to below the Macros and definitions, say line 97.
64 /**
65  * used to locally copy the current-sensor value from the global database
66  * current table
67  */
68 static DATA_BLOCK_CURRENT_SENSOR_s cont_current_tab = {0};
69
70 /**
71  * Saves the last state and the last substate
72  */
73 #define CONT_SAVELASTSTATES()   cont_state.laststate = cont_state.state; \
74                                 cont_state.lastsubstate = cont_state.substate;
75
76 #define CONT_OPENALLCONTACTORS()   CONT_SwitchAllContactorsOff();
77
78 #define CONT_OPENMINUS()       CONT_SetContactorState(CONT_MAIN_MINUS, OPEN
   CONT_SWITCH_OFF);
79 #define CONT_CLOSEMINUS()       CONT_SetContactorState(CONT_MAIN_MINUS, CLOSE
   CONT_SWITCH_ON);
80
81 #define CONT_OPENPLUS()       CONT_SetContactorState(CONT_MAIN_PLUS, OPEN
   CONT_SWITCH_OFF);
82 #define CONT_CLOSEPLUS()       CONT_SetContactorState(CONT_MAIN_PLUS, CLOSE
   CONT_SWITCH_ON);
83
84 #define CONT_OPENPRECHARGE()       CONT_SetContactorState(CONT_PRECHARGE_PLUS,
   CONT_SWITCH_OFF);
85 #define CONT_CLOSEPRECHARGE()       CONT_SetContactorState(CONT_PRECHARGE_PLUS,
   CONT_SWITCH_ON);
86
87 #if BS_SEPARATE_POWERLINES == 1
88 #define CONT_OPENCHARGEMINUS()       CONT_SetContactorState(CONT_CHARGE_MAIN_MINUS,
   CONT_SWITCH_OFF);
89 #define CONT_CLOSECHARGEMINUS()
   CONT_SetContactorState(CONT_CHARGE_MAIN_MINUS, CONT_SWITCH_ON);
90
91 #define CONT_OPENCHARGEPLUS()       CONT_SetContactorState(CONT_CHARGE_MAIN_PLUS,
   CONT_SWITCH_OFF);
92 #define CONT_CLOSECHARGEPLUS()       CONT_SetContactorState(CONT_CHARGE_MAIN_PLUS,
   CONT_SWITCH_ON);
93
94 #define CONT_OPENCHARGEPRECHARGE()
   CONT_SetContactorState(CONT_CHARGE_PRECHARGE_PLUS, CONT_SWITCH_OFF);
95 #define CONT_CLOSECHARGEPRECHARGE()
   CONT_SetContactorState(CONT_CHARGE_PRECHARGE_PLUS, CONT_SWITCH_ON);
96 #endif /* BS_SEPARATE_POWERLINES == 1 */
97
98 /*================== Constant and Variable Definitions ====================*/
99
100 /**
101  * contains the state of the contactor state machine
102  *
103  */
104 static CONT_STATE_s cont_state = {
105     .timer                  = 0,
106     .statereq               = CONT_STATE_NO_REQUEST,
107     .state                  = CONT_STATEMACH_UNINITIALIZED,
108     .substate               = CONT_ENTRY,
109     .laststate              = CONT_STATEMACH_UNINITIALIZED,
110     .lastsubstate           = 0, Should be CONT_ENTRY
111     .triggerentry           = 0,
112     .ErrRequestCounter      = 0,
113     .initFinished           = E_NOT_OK,
```

The sub is not a good term---it is not clear here. Op, which stands for, operating, may be a better term---contactor operating state. This way, it indicates that the contactors operate in detailed steps.

This is an added page to list the prototypes of all the extern functions.

Need to check if some of the following is just a static function.

```
extern  138 CONT_ELECTRICAL_STATE_TYPE_s CONT_GetContactorSetValue(CONT_NAMES_e contactor)

extern  147 CONT_ELECTRICAL_STATE_TYPE_s CONT_GetContactorFeedback(CONT_NAMES_e contactor)

extern  185 STD_RETURN_TYPE_e CONT_AcquireContactorFeedbacks(void)

extern  196 STD_RETURN_TYPE_e CONT_SetContactorState(CONT_NAMES_e contactor,
            CONT_ELECTRICAL_STATE_TYPE_s requestedContactorState) {

extern  STD_RETURN_TYPE_e  CONT_SetContractorState_pulse(CONT_NAMES_e contactor);

extern  228 STD_RETURN_TYPE_e CONT_SwitchAllContactorsOff(void)

        286 static CONT_STATE_REQUEST_e CONT_GetStateRequest(void)

extern  297 CONT_STATEMACH_e CONT_GetState(void) {
        298     return (cont_state.state);
        299 }
        300
extern  301 STD_RETURN_TYPE_e CONT_GetInitializationState(void) {
        302     return (cont_state.initFinished);
        303 }
        304
extern  305 CONT_POWER_LINE_e CONT_GetActivePowerLine() {
        306     return (cont_state.activePowerLine);
        307 }

extern  331 CONT_RETURN_TYPE_e CONT_SetStateRequest(CONT_STATE_REQUEST_e statereq) {
```

```
                                    Used to add a large delay when switching to
114     .OscillationCounter     = 0,  the Standby state. See Line 486.
115     .PrechargeTryCounter    = 0,  Need to figure out the meaning of the precharge try counter.
116     .PrechargeTimeOut       = 0,
117     .counter                = 0,
118     .activePowerLine        = CONT_POWER_LINE_NONE,
119 };          The terminals are hot or not.
120
121 static DATA_BLOCK_CONTFEEDBACK_s contfeedback_tab = {
122         .contactor_feedback = 0,
123         .timestamp = 0,
124         .previous_timestamp = 0,
125 };
126
127                     Static
128 /*================== Function Prototypes ==================================*/
129
See 357 130 static CONT_RETURN_TYPE_e CONT_CheckStateRequest(CONT_STATE_REQUEST_e statereq);
See 286 131 static CONT_STATE_REQUEST_e CONT_GetStateRequest(void);
See 319 132 static CONT_STATE_REQUEST_e CONT_TransferStateRequest(void);
See 263 133 static uint8_t CONT_CheckReEntrance(void);
See 851 134 static void CONT_CheckFeedback(void);  Implementations of the above functions
135                                              should be placed here.
136 /*================== Function Implementations ============================*/
137                   (Extren)
138 CONT_ELECTRICAL_STATE_TYPE_s CONT_GetContactorSetValue(CONT_NAMES_e contactor) {
139     CONT_ELECTRICAL_STATE_TYPE_s contactorSetInformation = FALSE;
140     taskENTER_CRITICAL();    Defined in contactor_cfg.c. Comes with the header file.
141     contactorSetInformation = cont_contactor_states[contactor].set;
142     taskEXIT_CRITICAL();                                    set should be expectedValue.
143     return contactorSetInformation;
144 }
145
146
147 CONT_ELECTRICAL_STATE_TYPE_s CONT_GetContactorFeedback(CONT_NAMES_e contactor) {
148     CONT_ELECTRICAL_STATE_TYPE_s measuredContactorState = CONT_SWITCH_UNDEF;
149     if (CONT_HAS_NO_FEEDBACK ==
    cont_contactors_config[contactor].feedback_pin_type) {
150             measuredContactorState = cont_contactor_states[contactor].set;
151     } else {
152         /* the contactor has a feedback pin, but it has to be differenced if the
    feedback pin is normally open or normally closed */
                                                 Misaligned if else if. This should be
153         if (CONT_FEEDBACK_NORMALLY_OPEN ==   changed to switch on
    cont_contactors_config[contactor].feedback_pin_type) {   cont_conttactors_config[contactor].feed
                                                             back_pin_type.
154             IO_PIN_STATE_e pinstate = IO_PIN_RESET;
155             taskENTER_CRITICAL();
156             pinstate = IO_ReadPin(cont_contactors_config[contactor].feedback_pin);
157             taskEXIT_CRITICAL();
158             if (IO_PIN_RESET == pinstate) {
159                 measuredContactorState = CONT_SWITCH_ON;
160             } else if (IO_PIN_SET == pinstate) {
161                 measuredContactorState = CONT_SWITCH_OFF;
162             } else {
163                 measuredContactorState = CONT_SWITCH_UNDEF;
164             }
165         }
166         if (CONT_FEEDBACK_NORMALLY_CLOSED ==
    cont_contactors_config[contactor].feedback_pin_type) {
167             IO_PIN_STATE_e pinstate = IO_PIN_SET;
168             taskENTER_CRITICAL();
169             pinstate = IO_ReadPin(cont_contactors_config[contactor].feedback_pin);
170             taskEXIT_CRITICAL();
171             if (IO_PIN_SET == pinstate) {
172                 measuredContactorState = CONT_SWITCH_ON;
```

```
173             } else if (IO_PIN_RESET == pinstate) {
174                 measuredContactorState = CONT_SWITCH_OFF;
175             } else {
176                 measuredContactorState = CONT_SWITCH_UNDEF;
177             }
178         }
179     }
180     cont_contactor_states[contactor].feedback = measuredContactorState;
181     return measuredContactorState;
182 }
183 
184 
185 STD_RETURN_TYPE_e CONT_AcquireContactorFeedbacks(void) {
186     STD_RETURN_TYPE_e retVal = E_NOT_OK;
187     taskENTER_CRITICAL();                          -2   We only have 4 relays.
188     for (uint8_t i = 0; i < BS_NR_OF_CONTACTORS; i++) {
189         cont_contactor_states[i].feedback = CONT_GetContactorFeedback(i);
190     }
191     retVal = E_OK;
192     taskEXIT_CRITICAL();
193     return retVal;
194 }
195 
196 STD_RETURN_TYPE_e CONT_SetContactorState(CONT_NAMES_e contactor,
CONT_ELECTRICAL_STATE_TYPE_s requestedContactorState) {
197     STD_RETURN_TYPE_e retVal = E_OK;
198 
199     if (requestedContactorState  ==  CONT_SWITCH_ON) {
200         cont_contactor_states[contactor].set = CONT_SWITCH_ON;
201         IO_WritePin(cont_contactors_config[contactor].control_pin, IO_PIN_SET);
202         if (DIAG_HANDLER_RETURN_OK != DIAG_ContHandler(DIAG_EVENT_OK, (uint8_t)
contactor, NULL)) {
                                                  This is ignored.
203             /* TODO: explain why empty if */
204         }
205     } else if (requestedContactorState  ==  CONT_SWITCH_OFF) {
206         DB_ReadBlock(&cont_current_tab, DATA_BLOCK_ID_CURRENT_SENSOR);
207         float currentAtSwitchOff = cont_current_tab.current;
208         if (((BAD_SWITCHOFF_CURRENT_POS < currentAtSwitchOff) && (0 <
currentAtSwitchOff)) ||                                          This is redundant.
209             ((BAD_SWITCHOFF_CURRENT_NEG > currentAtSwitchOff) && (0 >
currentAtSwitchOff))) {
210             if (DIAG_HANDLER_RETURN_OK != DIAG_ContHandler(DIAG_EVENT_NOK,
(uint8_t) contactor, &currentAtSwitchOff)) {
211                 /* currently no error handling, just logging */
212             }
213         } else {
214             if (DIAG_HANDLER_RETURN_OK != DIAG_ContHandler(DIAG_EVENT_OK, (uint8_t)
contactor, NULL)) {
215                 /* TODO: explain why empty if */
216             }
217        }
218         cont_contactor_states[contactor].set = CONT_SWITCH_OFF;
219         IO_WritePin(cont_contactors_config[contactor].control_pin, IO_PIN_RESET);
220     } else {
221         retVal = E_NOT_OK;
222     }                  Need to add CONT_SetContractorState_pulse(CONT_NAMES_e contactor) function.
223                        The steps in the function are as follows:
224     return retVal;     1. IO_WritePin(cont_contactors_config[contactor].control_pin, IO_PIN_SET);
225 }                      2. vTaskDelay(CONT_LATCHING_PULSE_WIDTH_MS);
226                        3. IO_WritePin(cont_contactors_config[contactor].control_pin, IO_PIN_RESET);
227                        4. Update the cont_contactor_states like
228 STD_RETURN_TYPE_e CONT_SwitchAllContactorsOff(void) {
229     STD_RETURN_TYPE_e retVal = E_NOT_OK;
```

```
230     uint8_t offCounter = 0;
231     STD_RETURN_TYPE_e successfullSet = E_NOT_OK;
232
233     for (uint8_t i = 0; i < BS_NR_OF_CONTACTORS; i++) {
234         successfullSet = CONT_SetContactorState(i, CONT_SWITCH_OFF);
235         if (E_OK == successfullSet) {
236             offCounter = offCounter + 1;
237         }
238         successfullSet = E_NOT_OK;
239     }
240
241     if (BS_NR_OF_CONTACTORS -2 == offCounter) {
242         retVal = E_OK;
243     } else {
244         retVal = E_NOT_OK;
245     }
246     return retVal;
247 }
248
249
250
251 /**
252  * @brief   re-entrance check of CONT state machine trigger function
253  *
254  * @details This function is not re-entrant and should only be called time- or
    event-triggered. It
255  *          increments the triggerentry counter from the state variable ltc_state.
256  *          It should never be called by two different processes, so if it is the
    case,
257  *          triggerentry should never be higher than 0 when this function is
    called.
258  *
259  *
260  * @return  0 if no further instance of the function is active, 0xff else
261  *
262  */
263 static uint8_t CONT_CheckReEntrance(void) {
264     uint8_t retval = 0;
265
266     taskENTER_CRITICAL();
267     if (!cont_state.triggerentry) {
268         cont_state.triggerentry++;
269     } else {
270         retval = 0xFF;
271     }
272     taskEXIT_CRITICAL();
273     return retval;
274 }
275
276
277
278
279 /**
280  * @brief   gets the current state request.
281  *
282  * @details This function is used in the functioning of the CONT state machine.
283  *
284  * @return  return the current pending state request
285  */
286 static CONT_STATE_REQUEST_e CONT_GetStateRequest(void) {
287     CONT_STATE_REQUEST_e retval = CONT_STATE_NO_REQUEST;
288
289     taskENTER_CRITICAL();
```

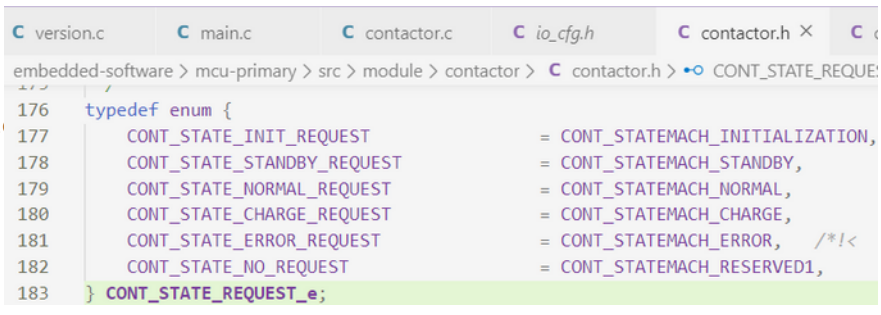
Keep this offCounter

Need to list all the four relays one by one since they are not the same type.

```
bool reEntry = false;
reEntry = true;
return reEntry;
```

This should just be called isReEntry. The return type is boolean. This way, it is much more cleaner.

```
290     retval = cont_state.statereq;
291     taskEXIT_CRITICAL();
292
293     return retval;
294 }
295
296
297 CONT_STATEMACH_e CONT_GetState(void) {
298     return (cont_state.state);
299 }
300
301 STD_RETURN_TYPE_e CONT_GetInitializationState(void) {
302     return (cont_state.initFinished);
303 }
304
305 CONT_POWER_LINE_e CONT_GetActivePowerLine() {
306     return (cont_state.activePowerLine);
307 }
308
309
310 /**
311  * @brief   transfers the current state request to the state machine.
312  *
313  * @details This function takes the current state request from cont_state and
    transfers it to the
314  *          state machine. It resets the value from cont_state to
    CONT_STATE_NO_REQUEST
315  *
316  * @return  current state request, taken from CONT_STATE_REQUEST_e
317  *
318  */
319 static CONT_STATE_REQUEST_e CONT_TransferStateRequest(void) {
320     CONT_STATE_REQUEST_e retval = CONT_STATE_NO_REQUEST;
321
322     taskENTER_CRITICAL();
323     retval    = cont_state.statereq;
324     cont_state.statereq = CONT_STATE_NO_REQUEST;
325     taskEXIT_CRITICAL();
326
327     return (retval);
328 }
329
330
331 CONT_RETURN_TYPE_e CONT_SetStateRequest(CONT_STATE_REQUEST_e statereq) {
332     CONT_RETURN_TYPE_e retVal = CONT_STATE_NO_REQUEST; CONT_ILLEGAL_REQUEST
333                                 This is not a member of CONT_RETURN_TYPE_e. This is a BUG.
334     taskENTER_CRITICAL();
335     retVal = CONT_CheckStateRequest(statereq);      A better name of this function can be
336                                                     CONT_isStateRequestOk.
337     if (retVal == CONT_OK) {
338             cont_state.statereq    = statereq;      If retVal is not CONT_OK, the statereq
339     }                                               is lost/ignored?
340     taskEXIT_CRITICAL();
341
342     return retVal;
343 }
344
345
346
347 /**
348  * @bri
349  *
```



```
 * @brief    checks the state request that is just made.
 * @details This function checks the validity of the state requests. The results of the checked is
 *          returned immediately.
 *
 * @param    statereq    state request to be checked
 *          the evaluation
 * @return  result of the state request that was made, taken from (type: CONT_RETURN_TYPE_e)
 */
static CONT_RETURN_TYPE_e CONT_CheckStateRequest(CONT_STATE_REQUEST_e statereq) {
    if (statereq == CONT_STATE_ERROR_REQUEST) {
        return CONT_OK;
    }

    if (cont_state.statereq == CONT_STATE_NO_REQUEST) {
        /* init only allowed from the uninitialized state */
        if (statereq == CONT_STATE_INIT_REQUEST) {          This is requested form sys.c.
            if (cont_state.state == CONT_STATEMACH_UNINITIALIZED) {
                return CONT_OK;
            } else {
                return CONT_ALREADY_INITIALIZED;
            }
        } else      If we don't use switch, we                    A "switch" on variable statereq
                    need to use else if here.                     is more appropriate.
        if ((statereq == CONT_STATE_STANDBY_REQUEST) || (statereq == CONT_STATE_NORMAL_REQUEST) || (statereq == CONT_STATE_CHARGE_REQUEST)) {
            return CONT_OK;  BUG
        } else if (statereq == CONT_STATE_NORMAL_REQUEST) {
            if (cont_state.state == CONT_STATEMACH_CHARGE_PRECHARGE || cont_state.state == CONT_STATEMACH_CHARGE) {
                return CONT_REQUEST_IMPOSSIBLE;          BUG
            } else {
                return CONT_OK;
            }
        } else if (statereq == CONT_STATE_CHARGE_REQUEST) {
            if (cont_state.state == CONT_STATEMACH_PRECHARGE || cont_state.state == CONT_STATEMACH_NORMAL) {
                return CONT_REQUEST_IMPOSSIBLE;
            } else {
                return CONT_OK;
            }
        } else {  Add the Engine request here. See code snippet (2) of the next page.
            return CONT_ILLEGAL_REQUEST;
        }
    } else {
        return CONT_REQUEST_PENDING;
    }           Then, where is this saved? Lost?
}

void CONT_Trigger(void) {
    STD_RETURN_TYPE_e retVal = E_OK;
    CONT_STATE_REQUEST_e statereq = CONT_STATE_NO_REQUEST;

              isReEntry
    if (CONT_CheckReEntrance()) {        triggerentry++
        return;
    }

    DIAG_SysMonNotify(DIAG_SYSMON_CONT_ID, 0);  /* task is running, state = ok */

    if (cont_state.state != CONT_STATEMACH_UNINITIALIZED) {
        CONT_CheckFeedback();
    }
```

There should be two more cases---for Engine start. Hence, a switch is more appropriate. See code snippet (1a) or (1b) of the next page for the implementation of the highlighted.

This case needs to be processed similar to the above.

version.c | main.c | contactor.c | io_cfg.h | contactor.h

embedded-software > mcu-primary > src > module > contactor > contactor.h > CONT_STATE_R

```
typedef enum {
    CONT_OK                 = 0,    /*!< CONT --> ok
    CONT_BUSY_OK            = 1,    /*!< CONT under loc
    CONT_REQUEST_PENDING    = 2,    /*!< requested to b
    CONT_REQUEST_IMPOSSIBLE = 3,    /*!< requested not
    CONT_ILLEGAL_REQUEST    = 4,    /*!< Request can no
    CONT_INIT_ERROR         = 5,    /*!< Error state: S
    CONT_OK_FROM_ERROR      = 6,    /*!< Return from er
    CONT_ALREADY_INITIALIZED = 30,  /*!< Initializatior
    CONT_ILLEGAL_TASK_TYPE  = 99,   /*!< Illegal
} CONT_RETURN_TYPE_e;
```

(1a)

```
switch (cont_state.state) {
case CONT_STATEMACH_CHARGE_PRECHARGE:
case CONT_STATEMACH_CHARGE:
case CONT_STATEMACH_ENGINE_PRECHARGE:
case CONT_STATEMACH_ENGINE:
    return CONT_REQUEST_IMPOSSIBLE;
default:
    return CONT_OK;
}
```

Note that (1a) and (1b) are not exactly the same, but they perform a similar function. (1b) is cleaner than (1a).

(1b)

```
if (cont_state.state == CONT_STATEMACH_STANDBY) {
    return CONT_OK;
} else {
    return CONT_REQUEST_IMPOSSIBLE;
}
```

It can only go from STANDBY to NORMAL

(2)

```
else if (statereq == CONT_STATE_ENGINE_REQUEST) {
    if (cont_state.state == CONT_STATEMACH_STANDBY) {
        return CONT_OK;
    } else {
        return CONT_REQUEST_IMPOSSIBLE;
    }
}
```

Or, the entire function can be written as, using a different logic in terms of cont_state.state:

```
static CONT_RETURN_TYPE_e CONT_CheckStateRequest(CONT_STATE_REQUEST_e statereq) {
    if (statereq == CONT_STATE_ERROR_REQUEST) {
        return CONT_OK;
    } else if (statereq == CONT_STATE_NO_REQUEST) {
        return CONT_ILLEGAL_REQUEST;
    }

    if cont_state.statereq == CONT_STATE_NO_REQUEST) {
        if (cont_state.state == CONT_STATEMECH_STANDBY) {
            if (statereq == CONT_STATE_STANDBY_REQUEST) {
                return CONT_ILLEGAL_REQUEST;
            } else {
                return CONT_OK;
            }
        }
    } else {
        return CONT_REQUEST_PENDING;
    }
}
```

```
408     if (cont_state.OscillationCounter > 0) {
409         cont_state.OscillationCounter--;            See Line 586.
410     }
411
412     if (cont_state.PrechargeTimeOut > 0) {
413         if (cont_state.PrechargeTimeOut > CONT_TASK_CYCLE_CONTEXT_MS) {
414             cont_state.PrechargeTimeOut -= CONT_TASK_CYCLE_CONTEXT_MS;
415         } else {
416             cont_state.PrechargeTimeOut = 0;
417         }
418     }
419
420     if (cont_state.timer) {
421         if (cont_state.timer > CONT_TASK_CYCLE_CONTEXT_MS) {
422             cont_state.timer -= CONT_TASK_CYCLE_CONTEXT_MS;
423         } else {
424             cont_state.timer = 0;
425         }
426         if (cont_state.timer) {                  triggerentry--
427   Bad logic     cont_state.triggerentry--;
428             return;     /* handle state machine only if timer has elapsed */
429         }
430     }
431
432
433     switch (cont_state.state) {
434
/*************************UNINITIALIZED************
435         case CONT_STATEMACH_UNINITIALIZED:
436             /* waiting for Initialization Request */
437             statereq = CONT_TransferStateRequest();
438             if (statereq == CONT_STATE_INIT_REQUEST) {
439                 CONT_SAVELASTSTATES();
440                 cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
441                 cont_state.state = CONT_STATEMACH_INITIALIZATION;
442                 cont_state.substate = CONT_ENTRY;
443             } else if (statereq == CONT_STATE_NO_REQUEST) {
444                 /* no actual request pending */
445             } else {
446                 cont_state.ErrRequestCounter++;   /* illegal request pending */
447             }
448             break;
449
450
451
/*************************INITIALIZATION***
452         case CONT_STATEMACH_INITIALIZATION:
453
454             CONT_SAVELASTSTATES();
455             CONT_OPENALLCONTACTORS();
456
457             cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
458             cont_state.state = CONT_STATEMACH_INITIALIZED;
459             cont_state.substate = CONT_ENTRY;
460
461             break;
462
463
/*************************INITIALIZED*************************************/
464         case CONT_STATEMACH_INITIALIZED:
465             CONT_SAVELASTSTATES();
466             cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
467             cont_state.state = CONT_STATEMACH_IDLE;
```

```
if (timer > Context_ms) {
    timer -= Contex_ms;
    triggerentry--;
    return;
} else {
    timer = 0;
}
```

From Line 398 and the code here, we know that triggertenty in only 0 or 1. Hence, it may be better to use a boolean var to make it clear.

version.c | main.c | contactor.c | io_cfg.h | contactor.h ×

embedded-software > mcu-primary > src > module > contactor > C contactor.h > CONT_STATE_R

```
140 typedef enum {
141     /* Init-Sequence */
142     CONT_STATEMACH_UNINITIALIZED          = 0,   /*!<    */
143     CONT_STATEMACH_INITIALIZATION         = 1,   /*!<    */
144     CONT_STATEMACH_INITIALIZED            = 2,   /*!<    */
145     CONT_STATEMACH_IDLE                   = 3,   /*!<    */
146     CONT_STATEMACH_STANDBY                = 4,   /*!<    */
147     CONT_STATEMACH_PRECHARGE              = 5,   /*!<    */
148     CONT_STATEMACH_NORMAL                 = 6,   /*!<    */
149     CONT_STATEMACH_CHARGE_PRECHARGE       = 7,   /*!<    */
150     CONT_STATEMACH_CHARGE                 = 8,   /*!<    */
151     CONT_STATEMACH_UNDEFINED              = 20,  /*!< undefined sta
152     CONT_STATEMACH_RESERVED1              = 0x80, /*!< reserved stat
153     CONT_STATEMACH_ERROR                  = 0xF0, /*!< Error-State:
154 } CONT_STATEMACH_e;
```

version.c | main.c | contactor.c | io_cfg.h | contactor.h ×

embedded-software > mcu-primary > src > module > contactor > C contactor.h > CONT_STATE_REQUE

```
176 typedef enum {
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,
181     CONT_STATE_ERROR_REQUEST              = CONT_STATEMACH_ERROR,   /*!<
182     CONT_STATE_NO_REQUEST                 = CONT_STATEMACH_RESERVED1,
183 } CONT_STATE_REQUEST_e;
```

See code snippet (1) of the next page

(1)

```
switch (cont_state.state) {

/*************************UNINITIALIZED*********************************/
case CONT_STATEMACH_ININITIALIZED:
    // Read the feedback of main plus (mp) and main minus (mm)
    CONT_ELECTRICAL_STATE_TYPE_s main_plus_feedback =
            CONT_GetContactorFeedback(CONT_MAIN_PLUS_CLOSE);
    CONT_ELECTRICAL_STATE_TYPE_s main_minus_feedback =
            CONT_GetContactorFeedback(CONT_MAIN_MINUS_CLOSE);

    // Transition to three states:
    // * Standby if mp is open
    // * Charge if mp is closed and mm is open
    // * Normal if both are closed
    if (main_plus_feedback == CONT_SWITCH_OFF) {
        cont_state.state = CONT_STATEMACH_STANDBY;
        cont_state.substate = CONT_ENTRY;
        break;
    } else {
        if (main_minus_feedback == CONT_SWITCH_OFF) {
            cont_state.state = CONT_STATEMACH_CHARGE;
            // need to see if the a new substate is needed
            cont_state.substate = CONT_STANDBY;

        } else {
            cont_state.state = CONT_STATEMACH_NORMAL;
            cont_state.substate = CONT_STANDBY;
        }
    }

    break;
```

(2)

```
 /*************************STANDBY*************************************/
 case CONT_STATEMACH_STANDBY:
     CONT_SAVELASTSTATES();

     if (cont_state.substate == CONT_ENTRY) {
         cont_state.OscillationCounter = CONT_OSCILLATION_LIMIT;
         CONT_OPEALLCONTACTORS();
         cont_state.activePowerLine = CONT_POWER_LINE_NONE;
         cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
         cont_state.substate = CONT_STANDBY;
     } else if (cont_state.substate == CONT_STANDBY) {
         statereq = CONT_TransferStateRequest();
         switch (statereq) {
         case CONT_STATE_NORMAL_REQUEST:
             CONT_SAVELASTSTATES();
             cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
             cont_state.state = CONT_STATEMACH_PRECHARGE;
             cont_state.substate = CONT_ENTRY;
             break;
         case CONT_STATE_CHARGE_REQUEST:
             CONT_SAVELASTSTATES();
             cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
             cont_state.state = CONT_STATEMACH_CHARGE_PRECHARGE;
             cont_state.substate = CONT_ENTRY;
             break;
         case CONT_STATE_ENGINE_REQUEST:
             CONT_SAVELASTSTATES();
             cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
             cont_state.state = CONT_STATEMACH_ENGINE_PRECHARGE;
             cont_state.substate = CONT_ENTRY;
             break;
         case CONT_STATE_ERROR_REQUEST:
             CONT_SAVELASTSTATES();
             cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
             cont_state.state = CONT_STATEMACH_ERROR;
             cont_state.substate = CONT_ENTRY;
             break;
         case CONT_STATE_NO_REQUEST:
         case CONT_STATE_STANDBY_REQUEST:
             break;
         default:
             cont_state.ErrRequestCounter++;  /* illegal request pending */
         }
     }
     break;
```

```
468             cont_state.substate = CONT_ENTRY;
469             break;
470
471         /*****************************IDLE*************************************/
472         case CONT_STATEMACH_IDLE:
473             CONT_SAVELASTSTATES();
474             cont_state.initFinished = E_OK;
475             cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
476             cont_state.state = CONT_STATEMACH_STANDBY;
477             cont_state.substate = CONT_ENTRY;
478             break;
479
480         /*****************************STANDBY**************
481         case CONT_STATEMACH_STANDBY:
482             CONT_SAVELASTSTATES();
483
484             /* first precharge process */
485             if (cont_state.substate == CONT_ENTRY) {
486                 cont_state.OscillationCounter = CONT_OSCILLATION_LIMIT;
487                 CONT_OPENPRECHARGE();
488                 #if BS_SEPARATE_POWERLINES == 1
489                     CONT_OPENCHARGEPRECHARGE();
490                 #endif
491                 cont_state.activePowerLine = CONT_POWER_LINE_NONE;
492                 cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
493                 cont_state.substate = CONT_OPEN_FIRST_CONTACTOR;
494                 break;
495             } else if (cont_state.substate == CONT_OPEN_FIRST_CONTACTOR) {
496                 if (BS_CheckCurrent_Direction() == BS_CURRENT_DISCHARGE) {
497                     CONT_OPENPLUS();
498                     #if BS_SEPARATE_POWERLINES == 1
499                         CONT_OPENCHARGEPLUS();
500                     #endif
501                     cont_state.timer = CONT_DELAY_BETWEEN_OPENING_CONTACTORS_MS;
502                     cont_state.substate = CONT_OPEN_SECOND_CONTACTOR_MINUS;
503                 } else {
504                     CONT_OPENMINUS();
505                     #if BS_SEPARATE_POWERLINES == 1
506                         CONT_OPENCHARGEMINUS();
507                     #endif
508                     cont_state.timer = CONT_DELAY_BETWEEN_OPENING_CONTACTORS_MS;
509                     cont_state.substate = CONT_OPEN_SECOND_CONTACTOR_PLUS;
510                 }
511                 break;
512             } else if (cont_state.substate == CONT_OPEN_SECOND_CONTACTOR_MINUS) {
513                 CONT_OPENMINUS();
514                 #if BS_SEPARATE_POWERLINES == 1
515                     CONT_OPENCHARGEMINUS();
516                 #endif
517                 cont_state.timer = CONT_DELAY_AFTER_OPENING_SECOND_CONTACTORS_MS;
518                 cont_state.substate = CONT_STANDBY;
519                 break;
520             } else if (cont_state.substate == CONT_OPEN_SECOND_CONTACTOR_PLUS) {
521                 CONT_OPENPLUS();
522                 #if BS_SEPARATE_POWERLINES == 1
523                     CONT_OPENCHARGEPLUS();
524                 #endif
525                 cont_state.timer = CONT_DELAY_AFTER_OPENING_SECOND_CONTACTORS_MS;
526                 cont_state.substate = CONT_STANDBY;
527                 break;
528             } else if (cont_state.substate == CONT_STANDBY) {
529                 /* when process done, look for requests */
530                 statereq = CONT_TransferStateRequest();
```

Defined in contactor.h

```
typedef enum {
    CONT_ENTRY                          = 0,
    CONT_OPEN_FIRST_CONTACTOR           = 1,
    CONT_OPEN_SECOND_CONTACTOR_MINUS    = 2,
    CONT_OPEN_SECOND_CONTACTOR_PLUS     = 3,
    CONT_STANDBY                        = 4,
    CONT_PRECHARGE_CLOSE_MINUS          = 5,
    CONT_PRECHARGE_CLOSE_PRECHARGE      = 6,
    CONT_PRECHARGE_CLOSE_PLUS           = 7,
    CONT_PRECHARGE_CHECK_VOLTAGES       = 8,
    CONT_PRECHARGE_OPEN_PRECHARGE       = 9,
    CONT_ERROR                          = 10,
} CONT_STATEMACH_SUB_e;
```

See code snippet (2) of the previous page

This is a perfect case to use Switch on cont_state.substate.

We can just used a fixed sequence of opening.

Need to have a new state diagram for the sub-states. Or, no need to have a sub-state machine here---just open them quickly in one shot.

```
531                 if (statereq == CONT_STATE_STANDBY_REQUEST) {
532                     /* we stay already in requested state, nothing to do */
533                 } else if (statereq == CONT_STATE_NORMAL_REQUEST) {
534                     CONT_SAVELASTSTATES();
535                     cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
536                     cont_state.state = CONT_STATEMACH_PRECHARGE;
537                     cont_state.substate = CONT_ENTRY;
538                 }
539 #if BS_SEPARATE_POWERLINES == 1
540                 else if (statereq == CONT_STATE_CHARGE_REQUEST) { /*
    NOLINT(readability/braces) */
541                     CONT_SAVELASTSTATES();
542                     cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
543                     cont_state.state = CONT_STATEMACH_CHARGE_PRECHARGE;
544                     cont_state.substate = CONT_ENTRY;
545                 }
546 #endif /* BS_SEPARATE_POWERLINES == 1 */
547                 else if (statereq == CONT_STATE_ERROR_REQUEST) { /*
    NOLINT(readability/braces) */                    Add the ENGINE_PRECHARGE state.
548                     CONT_SAVELASTSTATES();
549                     cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
550                     cont_state.state = CONT_STATEMACH_ERROR;
551                     cont_state.substate = CONT_ENTRY;
552                 } else if (statereq == CONT_STATE_NO_REQUEST) {
553                     /* no actual request pending */
554                 } else {
555                     cont_state.ErrRequestCounter++;  /* illegal request pending */
556                 }
557 
558                 /* check fuse state */
559                 CONT_CheckFuse(CONT_POWERLINE_NORMAL);     Defined in contactor_cfg.c.
560                 break;                                     Should have been defined in
561             }                                              this file.
562             break;
563 
564 
     /****************************PRECHARGE***********************************/
565         case CONT_STATEMACH_PRECHARGE:
566             CONT_SAVELASTSTATES();
567             /*  check state requests */
568             statereq = CONT_TransferStateRequest();
569             if (statereq == CONT_STATE_ERROR_REQUEST) {
570                 CONT_SAVELASTSTATES();
571                 cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
572                 cont_state.state = CONT_STATEMACH_ERROR;
573                 cont_state.substate = CONT_ENTRY;
574                 break;
575             }
576             if (statereq == CONT_STATE_STANDBY_REQUEST) {
577                 CONT_SAVELASTSTATES();
578                 cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
579                 cont_state.state = CONT_STATEMACH_STANDBY;
580                 cont_state.substate = CONT_ENTRY;
581                 break;
582             }
583 
584             /* precharge process, can be interrupted anytime by the requests above
    */
585             if (cont_state.substate == CONT_ENTRY) {
586                 if (cont_state.OscillationCounter > 0) {
587                     cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
588                     break;
589                 } else {
```

```
590                     cont_state.PrechargeTryCounter = 0;
591                     cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
592                     cont_state.substate = CONT_PRECHARGE_CLOSE_MINUS;
593                     break;
594                 }
595             } else if (cont_state.substate == CONT_PRECHARGE_CLOSE_MINUS) {
596                 cont_state.PrechargeTryCounter++;
597                 cont_state.PrechargeTimeOut = CONT_PRECHARGE_TIMEOUT_MS;
598                 CONT_CLOSEMINUS();
599                 cont_state.timer = CONT_STATEMACH_WAIT_AFTER_CLOSING_MINUS_MS;
600                 cont_state.substate = CONT_PRECHARGE_CLOSE_PRECHARGE;
601                 break;
602             } else if (cont_state.substate == CONT_PRECHARGE_CLOSE_PRECHARGE) {
603                 CONT_CLOSEPRECHARGE();
604                 cont_state.timer = CONT_STATEMACH_WAIT_AFTER_CLOSING_PRECHARGE_MS;
605                 cont_state.substate = CONT_PRECHARGE_CHECK_VOLTAGES;
606                 break;                                          TIME
607             } else if (cont_state.substate == CONT_PRECHARGE_CHECK_VOLTAGES) {
608                 retVal = CONT_CheckPrecharge(CONT_POWERLINE_NORMAL);
609                 if (retVal == E_OK) {
610                     CONT_CLOSEPLUS();
611                     cont_state.timer = CONT_STATEMACH_WAIT_AFTER_CLOSING_PLUS_MS;
612                     cont_state.substate = CONT_PRECHARGE_OPEN_PRECHARGE;
613                     break;
614                 } else if (cont_state.PrechargeTimeOut > 0) {
615                     break;
616                 } else {
617                     if (cont_state.PrechargeTryCounter < CONT_PRECHARGE_TRIES) {
618                         CONT_OPENALLCONTACTORS();   // report error
619                         cont_state.timer =
    CONT_STATEMACH_TIMEAFTERPRECHARGEFAIL_MS;
620                         cont_state.substate = CONT_PRECHARGE_CLOSE_MINUS;
621                         break;
622                     } else {
623                         cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
624                         cont_state.state = CONT_STATEMACH_ERROR;
625                         cont_state.substate = CONT_ENTRY;
626                         break;
627                     }
628                 }
629             } else if (cont_state.substate == CONT_PRECHARGE_OPEN_PRECHARGE) {
630                 CONT_OPENPRECHARGE();
631                 cont_state.timer = CONT_STATEMACH_WAIT_AFTER_OPENING_PRECHARGE_MS;
632                 cont_state.state = CONT_STATEMACH_NORMAL;
633                 cont_state.substate = CONT_ENTRY;
634                 cont_state.activePowerLine = CONT_POWER_LINE_0;
635                 break;
636             }
637
638             break;
639
640         /*****************************NORMAL***********************************/
641         case CONT_STATEMACH_NORMAL:
642             CONT_SAVELASTSTATES();
643             statereq = CONT_TransferStateRequest();
644             if (statereq == CONT_STATE_ERROR_REQUEST) {
645                 CONT_SAVELASTSTATES();
646                 cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
647                 cont_state.state = CONT_STATEMACH_ERROR;
648                 cont_state.substate = CONT_ENTRY;
649                 break;
650             }
651             if (statereq == CONT_STATE_STANDBY_REQUEST) {
```

Code between Lines 602 and 631 can be put in a function, called CONT_ClosePlusWithPrecharge. See the next page.

```
static void CONT_ClosePlusWithPrecharge(void) {
    if (cont_state.substate == CONT_PRECHARGE_CLOSE_PRECHARGE) {
        CONT_CLOSEPRECHARGE();
        //cont_state.timer = CONT_STATEMACH_WAIT_AFTER_CLOSING_PRECHARGE_MS;
        cont_state.timer = CONT_PRECHARGE_TIME_MS;
        cont_state.substate = CONT_PRECHARGE_CHECK_TIME;
    } else if (cont_state.substate == CONT_PRECHARGE_CHECK_TIME) {
        CONT_CLOSEPLUS();
        cont_state.timer = CONT_STATEMACH_WAIT_AFTER_CLOSING_PLUS_MS;
        cont_state.substate = CONT_PRECHARGE_OPEN_PRECHARGE;
    } else if (cont_state.substate == CONT_PRECHARGE_OPEN_PRECHARGE) {
        CONT_OPENPRECHARGE();
        cont_state.timer = CONT_STATEMACH_WAIT_AFTER_OPENING_PRECHARGE_MS;
        cont_state.substate = CONT_ENTRY;
    }
}
```

```
652                 CONT_SAVELASTSTATES();
653                 cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
654                 cont_state.state = CONT_STATEMACH_STANDBY;
655                 cont_state.substate = CONT_ENTRY;
656                 break;
657             }
658 
659             /* check fuse state */
660             CONT_CheckFuse(CONT_POWERLINE_NORMAL);
661             break;
662 
663 #if BS_SEPARATE_POWERLINES == 1
664 
 /****************************CHARGE_PRECHARGE**********************************
/
665         case CONT_STATEMACH_CHARGE_PRECHARGE:
666             CONT_SAVELASTSTATES();
667 
668             /* check state requests */
669             statereq = CONT_TransferStateRequest();
670             if (statereq == CONT_STATE_ERROR_REQUEST) {
671                 CONT_SAVELASTSTATES();
672                 cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
673                 cont_state.state = CONT_STATEMACH_ERROR;
674                 cont_state.substate = CONT_ENTRY;
675                 break;
676             }
677             if (statereq == CONT_STATE_STANDBY_REQUEST) {
678                 CONT_SAVELASTSTATES();
679                 cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
680                 cont_state.state = CONT_STATEMACH_STANDBY;
681                 cont_state.substate = CONT_ENTRY;
682                 break;
683             }
684 
685             /* precharge process, can be interrupted anytime by the requests above
*/
686             if (cont_state.substate == CONT_ENTRY) {
687                 if (cont_state.OscillationCounter > 0) {
688                     cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
689                     break;
690                 } else {
691                     cont_state.PrechargeTryCounter = 0;
692                     cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
693                     cont_state.substate = CONT_PRECHARGE_CLOSE_MINUS;
694                     break;                                          PRECHARGE
695                 }
696             } else if (cont_state.substate == CONT_PRECHARGE_CLOSE_MINUS) {
697                 cont_state.PrechargeTryCounter++;
698                 cont_state.PrechargeTimeOut = CONT_CHARGE_PRECHARGE_TIMEOUT_MS;
699                 CONT_CLOSECHARGEMINUS();
700                 cont_state.timer =
CONT_STATEMACH_CHARGE_WAIT_AFTER_CLOSING_MINUS_MS;
701                 cont_state.substate = CONT_PRECHARGE_CLOSE_PRECHARGE;
702                 break;
703             } else if (cont_state.substate == CONT_PRECHARGE_CLOSE_PRECHARGE) {
704                 CONT_CLOSECHARGEPRECHARGE();
705                 cont_state.timer =
CONT_STATEMACH_CHARGE_WAIT_AFTER_CLOSING_PRECHARGE_MS;
706                 cont_state.substate = CONT_PRECHARGE_CHECK_VOLTAGES;
707                 break;
708             } else if (cont_state.substate == CONT_PRECHARGE_CHECK_VOLTAGES) {
709                 retVal = CONT_CheckPrecharge(CONT_POWERLINE_CHARGE);

                CONT_ClosePlusWithPrecharge();
```

```
710                 if (retVal == E_OK) {
711                     CONT_CLOSECHARGEPLUS();
712                     cont_state.timer =
CONT_STATEMACH_CHARGE_WAIT_AFTER_CLOSING_PLUS_MS;
713                     cont_state.substate = CONT_PRECHARGE_OPEN_PRECHARGE;
714                     break;
715                 } else if (cont_state.PrechargeTimeOut > 0) {
716                     break;
717                 } else {
718                     if (cont_state.PrechargeTryCounter < CONT_PRECHARGE_TRIES) {
719                         CONT_OPENALLCONTACTORS();
720                         cont_state.timer =
CONT_STATEMACH_TIMEAFTERPRECHARGEFAIL_MS;
721                         cont_state.substate = CONT_PRECHARGE_CLOSE_MINUS;
722                         break;
723                     } else {
724                         cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
725                         cont_state.state = CONT_STATEMACH_ERROR;
726                         cont_state.substate = CONT_ENTRY;
727                         break;
728                     }
729                 }
730             } else if (cont_state.substate == CONT_PRECHARGE_OPEN_PRECHARGE) {
731                 CONT_OPENCHARGEPRECHARGE();
732                 cont_state.timer = CONT_STATEMACH_WAIT_AFTER_OPENING_PRECHARGE_MS;
733                 cont_state.state = CONT_STATEMACH_CHARGE;
734                 cont_state.substate = CONT_ENTRY;
735                 cont_state.activePowerLine = CONT_POWER_LINE_1;
736                 break;
737             }
738
739             break;
740
741         /*****************************CHARGE************************************/
742         case CONT_STATEMACH_CHARGE:
743             CONT_SAVELASTSTATES();
744
745             statereq = CONT_TransferStateRequest();
746             if (statereq == CONT_STATE_ERROR_REQUEST) {
747                 CONT_SAVELASTSTATES();
748                 cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
749                 cont_state.state = CONT_STATEMACH_ERROR;
750                 cont_state.substate = CONT_ENTRY;
751                 break;
752             }
753             if (statereq == CONT_STATE_STANDBY_REQUEST) {
754                 CONT_SAVELASTSTATES();
755                 cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
756                 cont_state.state = CONT_STATEMACH_STANDBY;
757                 cont_state.substate = CONT_ENTRY;
758                 break;
759             }
760
761 #endif /* BS_SEPARATE_POWERLINES == 1 */
762             break; Add the ENGINE handling here!!! The code should be the same as the Normal
763                    case, with CONT_CLOSEMINUS() being replaced by CONT_CLOSEENGINE().
764         /*****************************ERROR*************************************/
765         case CONT_STATEMACH_ERROR:
766             CONT_SAVELASTSTATES();
767
768             /* first error process */
769             if (cont_state.substate == CONT_ENTRY) {
770                 cont_state.OscillationCounter = CONT_OSCILLATION_LIMIT;
```

```
771                 CONT_OPENPRECHARGE();
772                 #if BS_SEPARATE_POWERLINES == 1
773                     CONT_OPENCHARGEPRECHARGE();
774                 #endif
775                 cont_state.timer = CONT_DELAY_BETWEEN_OPENING_CONTACTORS_MS;
776                 cont_state.substate = CONT_OPEN_FIRST_CONTACTOR;
777                 break;
778 
779             } else if (cont_state.substate == CONT_OPEN_FIRST_CONTACTOR) {
780                 if (BS_CheckCurrent_Direction() == BS_CURRENT_DISCHARGE) {
781                     CONT_OPENPLUS();
782                     #if BS_SEPARATE_POWERLINES == 1
783                         CONT_OPENCHARGEPLUS();
784                     #endif
785                     cont_state.timer = CONT_DELAY_BETWEEN_OPENING_CONTACTORS_MS;
786                     cont_state.substate = CONT_OPEN_SECOND_CONTACTOR_MINUS;
787                 } else {
788                     CONT_OPENMINUS();
789                     #if BS_SEPARATE_POWERLINES == 1
790                         CONT_OPENCHARGEMINUS();
791                     #endif
792                     cont_state.timer = CONT_DELAY_BETWEEN_OPENING_CONTACTORS_MS;
793                     cont_state.substate = CONT_OPEN_SECOND_CONTACTOR_PLUS;
794                 }
795                 /* mark no powerline as connected */
796                 cont_state.activePowerLine = CONT_POWER_LINE_NONE;
797                 break;
798 
799             } else if (cont_state.substate == CONT_OPEN_SECOND_CONTACTOR_MINUS) {
800                 CONT_OPENMINUS();
801                 #if BS_SEPARATE_POWERLINES == 1
802                     CONT_OPENCHARGEMINUS();
803                 #endif
804                 cont_state.timer = CONT_DELAY_AFTER_OPENING_SECOND_CONTACTORS_MS;
805                 cont_state.substate = CONT_ERROR;
806                 break;
807 
808             } else if (cont_state.substate == CONT_OPEN_SECOND_CONTACTOR_PLUS) {
809                 CONT_OPENPLUS();
810                 #if BS_SEPARATE_POWERLINES == 1
811                     CONT_OPENCHARGEPLUS();
812                 #endif
813                 cont_state.timer = CONT_DELAY_AFTER_OPENING_SECOND_CONTACTORS_MS;
814                 cont_state.substate = CONT_ERROR;
815                 break;
816 
817             } else if (cont_state.substate == CONT_ERROR) {
818                 /* Check if fuse is tripped */
819                 CONT_CheckFuse(CONT_POWERLINE_NORMAL);
820                 /* when process done, look for requests */
821                 statereq = CONT_TransferStateRequest();
822                 if (statereq == CONT_STATE_ERROR_REQUEST) {
823                     /* we stay already in requested state, nothing to do */
824                 } else if (statereq == CONT_STATE_STANDBY_REQUEST) {
825                     CONT_SAVELASTSTATES();
826                     cont_state.timer = CONT_STATEMACH_SHORTTIME_MS;
827                     cont_state.state = CONT_STATEMACH_STANDBY;
828                     cont_state.substate = CONT_ENTRY;
829                 } else if (statereq == CONT_STATE_NO_REQUEST) {
830                     /* no actual request pending */
831                 } else {
832                     cont_state.ErrRequestCounter++;   /* illegal request pending */
833                 }
```

```
834                 break;
835             }
836             break;
837
838         default:
839             break;
840     }  /* end switch (cont_state.state) */
841
842     cont_state.triggerentry--;
843     cont_state.counter++;
844 }
845
846 /**
847  * @brief   checks the feedback of the contactors
848  *
849  * @details makes a DIAG entry for each contactor when the feedback does not match
    the set value
850  */
851 void CONT_CheckFeedback(void) {
852     CONT_ELECTRICAL_STATE_TYPE_s feedback;
853     uint16_t contactor_feedback_state = 0;
854
855     for (uint8_t i = 0; i < BS_NR_OF_CONTACTORS; i++) {
856         feedback = CONT_GetContactorFeedback(i);
857
858         switch (i) {          -> contactor_feedback_state |= feedback << i;
859             case CONT_MAIN_PLUS:
860                 contactor_feedback_state |= feedback << CONT_MAIN_PLUS;
861                 break;
862             case CONT_MAIN_MINUS:
863                 contactor_feedback_state |= feedback << CONT_MAIN_MINUS;
864                 break;
865             case CONT_PRECHARGE_PLUS:
866                 contactor_feedback_state |= feedback << CONT_PRECHARGE_PLUS;
867                 break;
868 #if BS_SEPARATE_POWERLINES == 1
869             case CONT_CHARGE_MAIN_PLUS:
870                 contactor_feedback_state |= feedback << CONT_CHARGE_MAIN_PLUS;
871                 break;
872             case CONT_CHARGE_MAIN_MINUS:
873                 contactor_feedback_state |= feedback << CONT_CHARGE_MAIN_MINUS;
874                 break;
875             case CONT_CHARGE_PRECHARGE_PLUS:
876                 contactor_feedback_state |= feedback << CONT_CHARGE_PRECHARGE_PLUS;
877                 break;
878 #endif /* BS_SEPARATE_POWERLINES == 1 */
879             default:
880                 break;
881         }
882
883         contfeedback_tab.contactor_feedback &= (~0x3F);
884         contfeedback_tab.contactor_feedback |= contactor_feedback_state;
885
886         if (feedback != CONT_GetContactorSetValue(i)) {
887             switch (i) {
888                 case CONT_MAIN_PLUS:
889                     DIAG_Handler(DIAG_CH_CONTACTOR_MAIN_PLUS_FEEDBACK,
    DIAG_EVENT_NOK, 0);
890                     break;
891                 case CONT_MAIN_MINUS:
892                     DIAG_Handler(DIAG_CH_CONTACTOR_MAIN_MINUS_FEEDBACK,
    DIAG_EVENT_NOK, 0);
893                     break;
```

```
typedef enum {
    CONT_SWITCH_OFF     = 0,    /*!< Contactor off
    CONT_SWITCH_ON      = 1,    /*!< Contactor on
    CONT_SWITCH_UNDEF   = 2,    /*!< Contactor undefined
} CONT_ELECTRICAL_STATE_TYPE_s;
```

```
                case CONT_PRECHARGE_PLUS:
                    DIAG_Handler(DIAG_CH_CONTACTOR_PRECHARGE_FEEDBACK,
DIAG_EVENT_NOK, 0);
                    break;
#if BS_SEPARATE_POWERLINES == 1
                case CONT_CHARGE_MAIN_PLUS:
                    DIAG_Handler(DIAG_CH_CONTACTOR_CHARGE_MAIN_PLUS_FEEDBACK,
DIAG_EVENT_NOK, 0);
                    break;
                case CONT_CHARGE_MAIN_MINUS:
                    DIAG_Handler(DIAG_CH_CONTACTOR_CHARGE_MAIN_MINUS_FEEDBACK,
DIAG_EVENT_NOK, 0);
                    break;
                case CONT_CHARGE_PRECHARGE_PLUS:
                    DIAG_Handler(DIAG_CH_CONTACTOR_CHARGE_PRECHARGE_FEEDBACK,
DIAG_EVENT_NOK, 0);
                    break;
#endif /* BS_SEPARATE_POWERLINES == 1 */
                default:
                    break;
            }
        } else {
            switch (i) {
                case CONT_MAIN_PLUS:
                    DIAG_Handler(DIAG_CH_CONTACTOR_MAIN_PLUS_FEEDBACK,
DIAG_EVENT_OK, 0);
                    break;
                case CONT_MAIN_MINUS:
                    DIAG_Handler(DIAG_CH_CONTACTOR_MAIN_MINUS_FEEDBACK,
DIAG_EVENT_OK, 0);
                    break;
                case CONT_PRECHARGE_PLUS:
                    DIAG_Handler(DIAG_CH_CONTACTOR_PRECHARGE_FEEDBACK,
DIAG_EVENT_OK, 0);
                    break;
#if BS_SEPARATE_POWERLINES == 1
                case CONT_CHARGE_MAIN_PLUS:
                    DIAG_Handler(DIAG_CH_CONTACTOR_CHARGE_MAIN_PLUS_FEEDBACK,
DIAG_EVENT_OK, 0);
                    break;
                case CONT_CHARGE_MAIN_MINUS:
                    DIAG_Handler(DIAG_CH_CONTACTOR_CHARGE_MAIN_MINUS_FEEDBACK,
DIAG_EVENT_OK, 0);
                    break;
                case CONT_CHARGE_PRECHARGE_PLUS:
                    DIAG_Handler(DIAG_CH_CONTACTOR_CHARGE_PRECHARGE_FEEDBACK,
DIAG_EVENT_OK, 0);
                    break;
#endif /* BS_SEPARATE_POWERLINES == 1 */
                default:
                    break;
            }
        }
    }

    DB_WriteBlock(&contfeedback_tab, DATA_BLOCK_ID_CONTFEEDBACK);
}

#endif /* BUILD_MODULE_ENABLE_CONTACTOR */

```