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

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

#include "database.h"
#include <float.h>
#include <math.h>

#if BUILD_MODULE_ENABLE_CONTACTOR == 1
/*================== Macros and Definitions ================================*/

```

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

                                   BS_NR_OF_CONTACTORS is defined to be 6 in batterysystem_cfg.h
const CONT_CONFIG_s cont_contactors_config[BS_NR_OF_CONTACTORS] = {
    {CONT_MAIN_PLUS_CONTROL, CLOSE   CONT_MAIN_PLUS_FEEDBACK,
 CONT_FEEDBACK_NORMALLY_OPEN},
    {CONT_PRECHARGE_PLUS_CONTROL,   CONT_PRECHARGE_PLUS_FEEDBACK,
CONT_FEEDBACK_NORMALLY_OPEN},
    {CONT_MAIN_MINUS_CONTROL, CLOSE   CONT_MAIN_MINUS_FEEDBACK,
CONT_FEEDBACK_NORMALLY_OPEN}, {CONT_ENGINE_CONTROL,  CONT_ENGINE_FEEDBACK}, CONT_FEEDBACK_NORMALLY_OPEN},
~~#if BS_SEPARATE_POWERLINES == 1~~
    ~~{CONT_CHARGE_MAIN_PLUS_CONTROL,        CONT_CHARGE_MAIN_PLUS_FEEDBACK,~~
~~CONT_FEEDBACK_NORMALLY_OPEN},~~
    ~~{CONT_CHARGE_PRECHARGE_PLUS_CONTROL,   CONT_CHARGE_PRECHARGE_PLUS_FEEDBACK,~~
 ~~CONT_FEEDBACK_NORMALLY_OPEN},~~
    ~~{CONT_CHARGE_MAIN_MINUS_CONTROL,       CONT_CHARGE_MAIN_MINUS_FEEDBACK,~~
 ~~CONT_FEEDBACK_NORMALLY_OPEN}~~
~~#endif /* BS_SEPARATE_POWERLINES == 1 */~~
~~};~~  {CONT_MAIN_PLUS_CONTROL_OPEN, CONT_MAIN_PLUE_FEEDBACK2, CONT_HAS_NO_FEEDBACK},
        {CONT_MAIN_MINUS_CONTROL_OPEN, CONT_MAIN_MINUS_FEEDBACK2, CONT_HAS_NO_FEEDBACK}
 };
CONT_ELECTRICAL_STATE_s cont_contactor_states[BS_NR_OF_CONTACTORS] = {
    {0,    CONT_SWITCH_OFF},
    {0,    CONT_SWITCH_OFF},          This should have been placed in the contactor.c file.
    {0,    CONT_SWITCH_OFF},
~~#if BS_SEPARATE_POWERLINES == 1~~
    {0,    CONT_SWITCH_OFF},
    {0,    CONT_SWITCH_OFF},
    {0,    CONT_SWITCH_OFF},
~~#endif /* BS_SEPARATE_POWERLINES == 1 */~~
};

const uint8_t cont_contactors_config_length =
sizeof(cont_contactors_config)/sizeof(cont_contactors_config[0]);
const uint8_t cont_contactors_states_length =
sizeof(cont_contactor_states)/sizeof(cont_contactor_states[0]);
/*================== Function Prototypes ==================================*/

        IGNORE all the code changes below!!!
/*================== Function Implementations =============================*/
                         This is only called in contact.c and should be a static function there.
STD_RETURN_TYPE_e CONT_CheckPrecharge(CONT_WHICH_POWERLINE_e caller) {
    DATA_BLOCK_CURRENT_SENSOR_s current_tab = {0};        Why it is called tab? Table?
    STD_RETURN_TYPE_e retVal = E_NOT_OK;

    DB_ReadBlock(&current_tab, DATA_BLOCK_ID_CURRENT_SENSOR);
    float cont_prechargeVoltDiff_mV = 0.0;
    int32_t current_mA = 0;

    /* Only current not current direction is checked */
    if (current_tab.current > 0) {
        current_mA = current_tab.current;        current_mA = abs(current_tab.current);
    } else {                                     // Just need to define an abs function.
        current_mA = -current_tab.current;
    }

    if (caller == CONT_POWERLINE_NORMAL) {      // This is for discharge
        ~~cont_prechargeVoltDiff_mV = 0.0;~~
        /* Voltage difference between V2 and V3 of Isabellenhuette current sensor
*/
        ~~if (current_tab.voltage[1] > current_tab.voltage[2]) {~~
            cont_prechargeVoltDiff_mV = abs(current_tab.voltage[1] -
current_tab.voltage[2];)
        ~~} else {~~

                              This depends on the configuration of the
                              current/voltage sensors.
            ~~cont_prechargeVoltDiff_mV = current_tab.voltage[2] - current_tab.voltage[1];~~
        ~~}~~

        if ((cont_prechargeVoltDiff_mV < CONT_PRECHARGE_VOLTAGE_THRESHOLD_mV) && (current_mA < CONT_PRECHARGE_CURRENT_THRESHOLD_mA)) {
            retVal = E_OK;
        } else {
            retVal = E_NOT_OK;
        }
    } else if (caller == CONT_POWERLINE_CHARGE) {     // This is for charge
        ~~cont_prechargeVoltDiff_mV = 0.0~~;
        /* Voltage difference between V1 and V3 of Isabellenhuette current sensor */
        ~~if (current_tab.voltage[0] > current_tab.voltage[2]) {~~
            cont_prechargeVoltDiff_mV = current_tab.voltage[0] - current_tab.voltage[2];
                                                      abs(...)
        ~~} else {~~
            cont_prechargeVoltDiff_mV = ~~current_tab.voltage[2] - current_tab.voltage[0];~~
        }

        if ((cont_prechargeVoltDiff_mV < CONT_CHARGE_PRECHARGE_VOLTAGE_THRESHOLD_mV) && (current_mA < CONT_CHARGE_PRECHARGE_CURRENT_THRESHOLD_mA)) {
            retVal = E_OK;
        } else {
            retVal = E_NOT_OK;
        }
    }
    return retVal;
}
                         This is only called in contact.c and should be a static function there.

STD_RETURN_TYPE_e CONT_CheckFuse(CONT_WHICH_POWERLINE_e caller) {
#if (BS_CHECK_FUSE_PLACED_IN_NORMAL_PATH == TRUE) || (BS_CHECK_FUSE_PLACED_IN_CHARGE_PATH == TRUE)
    STD_RETURN_TYPE_e fuseState = E_NOT_OK;
    DATA_BLOCK_CURRENT_SENSOR_s curSensTab;
    DATA_BLOCK_CONTFEEDBACK_s contFeedbackTab;
    uint32_t voltDiff_mV = 0;
    STD_RETURN_TYPE_e checkFuseState = E_NOT_OK;

    DB_ReadBlock(&curSensTab, DATA_BLOCK_ID_CURRENT_SENSOR);
    DB_ReadBlock(&contFeedbackTab, DATA_BLOCK_ID_CONTFEEDBACK);

    if (caller == CONT_POWERLINE_NORMAL) {
        /* Fuse state can only be checked if plus and minus contactors are closed. */
        if ((((contFeedbackTab.contactor_feedback & 0x01) == 0x01) ||
                ((contFeedbackTab.contactor_feedback & 0x02) == 0x02)) &&
                ((contFeedbackTab.contactor_feedback & 0x04) == 0x04)) {
                    /* main plus OR main precharge AND minus are closed */
                checkFuseState = E_OK;
        }  else {
            /* Fuse state can't be checked if no plus contactors are closed */
            checkFuseState = E_NOT_OK;
        }
        /* Check voltage difference between battery voltage and voltage after fuse */
        if (checkFuseState == E_OK) {
            if (curSensTab.voltage[0] > curSensTab.voltage[1]) {
                voltDiff_mV = curSensTab.voltage[0] - curSensTab.voltage[1];
            } else {
                voltDiff_mV = curSensTab.voltage[1] - curSensTab.voltage[0];
            }

            /* If voltage difference is larger than max. allowed voltage drop over fuse*/
            if (voltDiff_mV > BS_MAX_VOLTAGE_DROP_OVER_FUSE_mV) {
                fuseState = E_NOT_OK;
            } else {
                fuseState = E_OK;
            }
        } else {
            /* Can't draw any conclusions about fuse state -> do not return E_NOT_OK */
            fuseState = E_OK;
        }
    } else if (caller == CONT_POWERLINE_CHARGE) {
        /* Fuse state can only be checked if plus and minus contactors are closed. */
        if ((((contFeedbackTab.contactor_feedback & 0x08) == 0x08) ||
                ((contFeedbackTab.contactor_feedback & 0x10) == 0x10)) &&
                ((contFeedbackTab.contactor_feedback & 0x20) == 0x20)) {
            /* charge plus OR charge precharge AND minus are closed */
                checkFuseState = E_OK;
        } else {
            /* Fuse state can't be checked if no plus contactors are closed */
            checkFuseState = E_NOT_OK;
        }
        /* Check voltage difference between battery voltage and voltage after fuse */
        if (checkFuseState == E_OK) {
            if (curSensTab.voltage[0] > curSensTab.voltage[1]) {
                voltDiff_mV = curSensTab.voltage[0] - curSensTab.voltage[2];
            } else {
                voltDiff_mV = curSensTab.voltage[2] - curSensTab.voltage[0];
            }

            /* If voltage difference is larger than max. allowed voltage drop over fuse*/
            if (voltDiff_mV > BS_MAX_VOLTAGE_DROP_OVER_FUSE_mV) {
                fuseState = E_NOT_OK;
            } else {
                fuseState = E_OK;
            }
        } else {
            /* Can't draw any conclusions about fuse state -> do not return E_NOT_OK */
            fuseState = E_OK;
        }
    }
#if BS_CHECK_FUSE_PLACED_IN_NORMAL_PATH == TRUE
    if (fuseState == E_OK) {
        /* Fuse state ok -> check precharging */
        DIAG_Handler(DIAG_CH_FUSE_STATE_NORMAL, DIAG_EVENT_OK, 0);
    } else {
        /* Fuse tripped -> switch to error state */
        DIAG_Handler(DIAG_CH_FUSE_STATE_NORMAL, DIAG_EVENT_NOK, 0);
    }
#endif  /* BS_CHECK_FUSE_PLACED_IN_NORMAL_PATH == TRUE */
#if BS_CHECK_FUSE_PLACED_IN_CHARGE_PATH == TRUE
    if (fuseState == E_OK) {
        /* Fuse state ok -> check precharging */
        DIAG_Handler(DIAG_CH_FUSE_STATE_CHARGE, DIAG_EVENT_OK, 0);
    } else {
        /* Fuse tripped -> switch to error state */
        DIAG_Handler(DIAG_CH_FUSE_STATE_CHARGE, DIAG_EVENT_NOK, 0);
    }
#endif  /* BS_CHECK_FUSE_PLACED_IN_CHARGE_PATH == TRUE */
    return fuseState;
#else /* BS_CHECK_FUSE_PLACED_IN_NORMAL_PATH == FALSE && BS_CHECK_FUSE_PLACED_IN_CHARGE_PATH == FALSE */
    return E_OK;
#endif /* BS_CHECK_FUSE_PLACED_IN_NORMAL_PATH || BS_CHECK_FUSE_PLACED_IN_CHARGE_PATH */
}
#endif /* BUILD_MODULE_ENABLE_CONTACTOR */

```