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

/**
 * @file    sys.c
 * @author  foxBMS Team
 * @date    21.09.2015 (date of creation)
 * @ingroup ENGINE
 * @prefix  SYS
 *
 * @brief   Sys driver implementation
 */


```

```c
/*================== Includes =============================================*/
#include "sys.h"

#include "bal.h"
#include "bms.h"
#include "cansignal.h"
#include "contactor.h"
#include "diag.h"
#include "interlock.h"
#include "isoguard.h"
#include "meas.h"
#include "rtc.h"
#include "sox.h"
#include "FreeRTOS.h"
#include "task.h"

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

/**
 * Saves the last state and the last substate
 */
#define SYS_SAVELASTSTATES()    sys_state.laststate = sys_state.state; \
                                sys_state.lastsubstate = sys_state.substate

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

/**
 * contains the state of the contactor state machine
 *
 */
static SYS_STATE_s sys_state = {
    .timer                  = 0,
    .statereq               = SYS_STATE_NO_REQUEST,
    .state                  = SYS_STATEMACH_UNINITIALIZED,
    .substate               = SYS_ENTRY,
    .laststate              = SYS_STATEMACH_UNINITIALIZED,
    .lastsubstate           = 0,
    .triggerentry           = 0,
    .ErrRequestCounter      = 0,
};

/*================== Function Prototypes ==================================*/

static SYS_RETURN_TYPE_e SYS_CheckStateRequest(SYS_STATE_REQUEST_e statereq);
static SYS_STATE_REQUEST_e SYS_GetStateRequest(void);
static SYS_STATE_REQUEST_e SYS_TransferStateRequest(void);
static uint8_t SYS_CheckReEntrance(void);

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

/**
 * @brief   re-entrance check of SYS state machine trigger function
 *
 * This function is not re-entrant and should only be called time- or event-triggered.
 * It increments the triggerentry counter from the state variable ltc_state.
 * It should never be called by two different processes, so if it is the case, triggerentry
 * should never be higher than 0 when this function is called.
 *
 *
 * @return  retval  0 if no further instance of the function is active, 0xff else
 *
 */
static uint8_t SYS_CheckReEntrance(void) {
    uint8_t retval = 0;

    taskENTER_CRITICAL();
    if (!sys_state.triggerentry) {
        sys_state.triggerentry++;
    } else {
        retval = 0xFF;  /* multiple calls of function */
    }
    taskEXIT_CRITICAL();

    return retval;
}




/**
 * @brief   gets the current state request.
 *
 * This function is used in the functioning of the SYS state machine.
 *
 * @return  retval  current state request, taken from SYS_STATE_REQUEST_e
 */
static SYS_STATE_REQUEST_e SYS_GetStateRequest(void) {
    SYS_STATE_REQUEST_e retval = SYS_STATE_NO_REQUEST;

    taskENTER_CRITICAL();
    retval    = sys_state.statereq;
    taskEXIT_CRITICAL();

    return (retval);
}


SYS_STATEMACH_e SYS_GetState(void) {
    return (sys_state.state);
}


/**
 * @brief   transfers the current state request to the state machine.
 *
 * This function takes the current state request from #sys_state and transfers it to the state machine.
 * It resets the value from #sys_state to #SYS_STATE_NO_REQUEST
 *
 * @return  retVal          current state request, taken from #SYS_STATE_REQUEST_e
 *
 */
static SYS_STATE_REQUEST_e SYS_TransferStateRequest(void) {
    SYS_STATE_REQUEST_e retval = SYS_STATE_NO_REQUEST;

    taskENTER_CRITICAL();
    retval    = sys_state.statereq;
    sys_state.statereq = SYS_STATE_NO_REQUEST;
    taskEXIT_CRITICAL();

    return (retval);
}



SYS_RETURN_TYPE_e SYS_SetStateRequest(SYS_STATE_REQUEST_e statereq) {
    SYS_RETURN_TYPE_e retVal = SYS_ILLEGAL_REQUEST;

    taskENTER_CRITICAL();
    retVal = SYS_CheckStateRequest(statereq);

    if (retVal == SYS_OK) {
            sys_state.statereq  = statereq;
        }
    taskEXIT_CRITICAL();

    return (retVal);
}



/**
 * @brief   checks the state requests that are made.
 *
 * This function checks the validity of the state requests.
 * The results of the checked is returned immediately.
 *
 * @param   statereq    state request to be checked
 *
 * @return              result of the state request that was made, taken from SYS_RETURN_TYPE_e
 */
static SYS_RETURN_TYPE_e SYS_CheckStateRequest(SYS_STATE_REQUEST_e statereq) {
    SYS_RETURN_TYPE_e retval = SYS_ILLEGAL_REQUEST;
    if (statereq == SYS_STATE_ERROR_REQUEST) {
        retval = SYS_OK;
    } else {
        if (sys_state.statereq == SYS_STATE_NO_REQUEST) {
            /* init only allowed from the uninitialized state */
            if (statereq == SYS_STATE_INIT_REQUEST) {
                if (sys_state.state == SYS_STATEMACH_UNINITIALIZED) {
                    retval = SYS_OK;
                } else {
                    retval = SYS_ALREADY_INITIALIZED;
                }
            } else {
                retval = SYS_ILLEGAL_REQUEST;
            }
        } else {
            retval = SYS_REQUEST_PENDING;
        }
    }
    return retval;
}


void SYS_Trigger(void) {
    /* STD_RETURN_TYPE_e retVal=E_OK; */
    SYS_STATE_REQUEST_e statereq = SYS_STATE_NO_REQUEST;
    ILCK_STATEMACH_e ilckstate = ILCK_STATEMACH_UNDEFINED;
    STD_RETURN_TYPE_e contstate = E_NOT_OK;
    STD_RETURN_TYPE_e balInitState = E_NOT_OK;
    STD_RETURN_TYPE_e bmsstate = E_NOT_OK;


    DIAG_SysMonNotify(DIAG_SYSMON_SYS_ID, 0);  /*  task is running, state = ok */
    /* Check re-entrance of function */
    if (SYS_CheckReEntrance()) {
        return;
    }

    if (sys_state.timer) {
        if (--sys_state.timer) {
            sys_state.triggerentry--;
            return;  /* handle state machine only if timer has elapsed */
        }
    }

    /****Happens every time the state machine is triggered**************/


    switch (sys_state.state) {
        /****************************UNINITIALIZED***********************************/
        case SYS_STATEMACH_UNINITIALIZED:
            /* waiting for Initialization Request */
            statereq = SYS_TransferStateRequest();
            if (statereq == SYS_STATE_INIT_REQUEST) {
                SYS_SAVELASTSTATES();
                sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                sys_state.state = SYS_STATEMACH_INITIALIZATION;
```

```c
                    sys_state.substate = SYS_ENTRY;
                } else if (statereq == SYS_STATE_NO_REQUEST) {
                    /* no actual request pending */
                } else {
                    sys_state.ErrRequestCounter++;   /* illegal request pending */
                }
                break;
            /****************************INITIALIZATION***********************************/
            case SYS_STATEMACH_INITIALIZATION:

                SYS_SAVELASTSTATES();
                /* Initializations done here */

                /* Send CAN boot message directly on CAN */
                SYS_SendBootMessage(1);

                /* Check if undervoltage MSL violation was detected before reset */
                if (RTC_DEEP_DISCHARGE_DETECTED == 1) {
                    /* Error detected */
                    DIAG_Handler(DIAG_CH_DEEP_DISCHARGE_DETECTED, DIAG_EVENT_NOK, 0);
                } else {
                    DIAG_Handler(DIAG_CH_DEEP_DISCHARGE_DETECTED, DIAG_EVENT_OK, 0);
                }

                sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                sys_state.state = SYS_STATEMACH_INITIALIZED;
                sys_state.substate = SYS_ENTRY;
                break;

            /****************************INITIALIZED*************************************/
            case SYS_STATEMACH_INITIALIZED:
                SYS_SAVELASTSTATES();
                sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
    #if BUILD_MODULE_ENABLE_ILCK == 1
                sys_state.state = SYS_STATEMACH_INITIALIZE_INTERLOCK;
    #elif BUILD_MODULE_ENABLE_CONTACTOR == 1
                sys_state.state = SYS_STATEMACH_INITIALIZE_CONTACTORS;
    #else
                sys_state.state = SYS_STATEMACH_INITIALIZE_BALANCING;
    #endif
                sys_state.substate = SYS_ENTRY;
                break;

    #if BUILD_MODULE_ENABLE_ILCK == 1
            /****************************INITIALIZE INTERLOCK*************************************/
            case SYS_STATEMACH_INITIALIZE_INTERLOCK:
                SYS_SAVELASTSTATES();

                if (sys_state.substate == SYS_ENTRY) {
                    ILCK_SetStateRequest(ILCK_STATE_INIT_REQUEST);
                    sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                    sys_state.substate = SYS_WAIT_INITIALIZATION_INTERLOCK;
                sys_state.InitCounter = 0;
                break;
            } else if (sys_state.substate == SYS_WAIT_INITIALIZATION_INTERLOCK) {
                ilckstate = ILCK_GetState();
                if (ilckstate == ILCK_STATEMACH_WAIT_FIRST_REQUEST) {
                    ILCK_SetStateRequest(ILCK_STATE_OPEN_REQUEST);
                    sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
#if BUILD_MODULE_ENABLE_CONTACTOR == 1
                    sys_state.state = SYS_STATEMACH_INITIALIZE_CONTACTORS;
#else
                    sys_state.state = SYS_STATEMACH_INITIALIZE_BALANCING;
#endif
                    sys_state.substate = SYS_ENTRY;
                    break;
                } else {
                    if (sys_state.InitCounter > (100/SYS_TASK_CYCLE_CONTEXT_MS)) {
                        sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                        sys_state.state = SYS_STATEMACH_ERROR;
                        sys_state.substate = SYS_ILCK_INIT_ERROR;
                        break;
                    }
                    sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                    sys_state.InitCounter++;
                    break;
                }
            }

            break;
#endif

#if BUILD_MODULE_ENABLE_CONTACTOR == 1
        /****************************INITIALIZE CONTACTORS*************************************/
        case SYS_STATEMACH_INITIALIZE_CONTACTORS:
            SYS_SAVELASTSTATES();

            if (sys_state.substate == SYS_ENTRY) {
                CONT_SetStateRequest(CONT_STATE_INIT_REQUEST);

                sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                sys_state.substate = SYS_WAIT_INITIALIZATION_CONT;
                sys_state.InitCounter = 0;
                break;
            } else if (sys_state.substate == SYS_WAIT_INITIALIZATION_CONT) {
                contstate = CONT_GetInitializationState();
                if (contstate == E_OK) {
                    sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                    sys_state.state = SYS_STATEMACH_INITIALIZE_BALANCING;
                    sys_state.substate = SYS_ENTRY;
                    break;
                } else {
                    if (sys_state.InitCounter > (100/SYS_TASK_CYCLE_CONTEXT_MS)) {
                        sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
```

```c
                            sys_state.state = SYS_STATEMACH_ERROR;
                            sys_state.substate = SYS_CONT_INIT_ERROR;
                            break;
                        }
                        sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                        sys_state.InitCounter++;
                        break;
                    }
                }

                break;
#endif
            /****************************INITIALIZE BALANCING*************************************/
            case SYS_STATEMACH_INITIALIZE_BALANCING:
                SYS_SAVELASTSTATES();
                if (sys_state.substate == SYS_ENTRY) {
                    BAL_SetStateRequest(BAL_STATE_INIT_REQUEST);
                    sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                    sys_state.substate = SYS_WAIT_INITIALIZATION_BAL;
                    sys_state.InitCounter = 0;
                    break;
                } else if (sys_state.substate == SYS_WAIT_INITIALIZATION_BAL) {
                    balInitState = BAL_GetInitializationState();
                    if (BALANCING_DEFAULT_INACTIVE == TRUE) {
                        BAL_SetStateRequest(BAL_STATE_GLOBAL_DISABLE_REQUEST);
                    } else {
                        BAL_SetStateRequest(BAL_STATE_GLOBAL_ENABLE_REQUEST);
                    }
                    if (balInitState == E_OK) {
                        sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                        sys_state.state = SYS_STATEMACH_INITIALIZE_ISOGUARD;
                        sys_state.substate = SYS_ENTRY;
                        break;
                    } else {
                        if (sys_state.InitCounter > (100/SYS_TASK_CYCLE_CONTEXT_MS)) {
                            sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                            sys_state.state = SYS_STATEMACH_ERROR;
                            sys_state.substate = SYS_BAL_INIT_ERROR;
                            break;
                        }
                        sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                        sys_state.InitCounter++;
                        break;
                    }
                }

                break;

            /**************************** Initialize Isoguard *************************/
            case SYS_STATEMACH_INITIALIZE_ISOGUARD:

#if BUILD_MODULE_ENABLE_ISOGUARD == 1
```

```c
                ISO_Init();
#endif
                sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                sys_state.state = SYS_STATEMACH_FIRST_MEASUREMENT_CYCLE;
                sys_state.substate = SYS_ENTRY;
                break;

            /****************************START FIRST MEAS CYCLE**************************/
            case SYS_STATEMACH_FIRST_MEASUREMENT_CYCLE:
                SYS_SAVELASTSTATES();
                if (sys_state.substate == SYS_ENTRY) {
                    MEAS_StartMeasurement();
                    sys_state.InitCounter = 0;
                    sys_state.substate = SYS_WAIT_FIRST_MEASUREMENT_CYCLE;
                } else if (sys_state.substate == SYS_WAIT_FIRST_MEASUREMENT_CYCLE) {
                    if (MEAS_IsFirstMeasurementCycleFinished() == TRUE) {
                        MEAS_Request_OpenWireCheck();
                        sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                        if (CURRENT_SENSOR_PRESENT == TRUE)
                            sys_state.state = SYS_STATEMACH_CHECK_CURRENT_SENSOR_PRESENCE;
                        else
                            sys_state.state = SYS_STATEMACH_INITIALIZE_MISC;
                        sys_state.substate = SYS_ENTRY;
                        break;
                    } else {
                        if (sys_state.InitCounter > (100/SYS_TASK_CYCLE_CONTEXT_MS)) {
                            sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                            sys_state.state = SYS_STATEMACH_ERROR;
                            sys_state.substate = SYS_MEAS_INIT_ERROR;
                            break;
                        } else {
                            sys_state.timer = SYS_STATEMACH_MEDIUMTIME_MS;
                            sys_state.InitCounter++;
                            break;
                        }
                    }
                }
                break;

            /****************************CHECK CURRENT SENSOR PRESENCE*************************************/
            case SYS_STATEMACH_CHECK_CURRENT_SENSOR_PRESENCE:
                SYS_SAVELASTSTATES();

                if (sys_state.substate == SYS_ENTRY) {
                    sys_state.InitCounter = 0;
                    CANS_Enable_Periodic(TRUE);
#if CURRENT_SENSOR_ISABELLENHUETTE_TRIGGERED
                    /* If triggered mode is used, CAN trigger message needs to
                     * be transmitted and current sensor response has to be
                     * received afterwards. This may take some time, therefore
                     * delay has to be increased.
                     */
                    sys_state.timer = SYS_STATEMACH_LONGTIME_MS;
#else /* CURRENT_SENSOR_ISABELLENHUETTE_TRIGGERED */
                    sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
#endif /* CURRENT_SENSOR_ISABELLENHUETTE_TRIGGERED */
                    sys_state.substate = SYS_WAIT_CURRENT_SENSOR_PRESENCE;
                } else if (sys_state.substate == SYS_WAIT_CURRENT_SENSOR_PRESENCE) {
                    if (CANS_IsCurrentSensorPresent() == TRUE) {
                        SOF_Init();
                        if (CANS_IsCurrentSensorCCPresent() == TRUE) {
                            SOC_Init(TRUE);
                        } else {
                            SOC_Init(FALSE);
                        }
                        sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                        sys_state.state = SYS_STATEMACH_INITIALIZE_MISC;
                        sys_state.substate = SYS_ENTRY;
                        break;
                    } else {
                        if (sys_state.InitCounter > (100/SYS_TASK_CYCLE_CONTEXT_MS)) {
                            sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                            sys_state.state = SYS_STATEMACH_ERROR;
                            sys_state.substate = SYS_CURRENT_SENSOR_PRESENCE_ERROR;
                            break;
                        } else {
                            sys_state.timer = SYS_STATEMACH_MEDIUMTIME_MS;
                            sys_state.InitCounter++;
                            break;
                        }
                    }
                }
                break;

            /****************************INITIALIZED_MISC*************************************/
            case SYS_STATEMACH_INITIALIZE_MISC:
                SYS_SAVELASTSTATES();

                if (CURRENT_SENSOR_PRESENT == FALSE) {
                    CANS_Enable_Periodic(TRUE);
                    SOC_Init(FALSE);
                }

                sys_state.timer = SYS_STATEMACH_MEDIUMTIME_MS;
                sys_state.state = SYS_STATEMACH_INITIALIZE_BMS;
                sys_state.substate = SYS_ENTRY;
                break;

            /****************************INITIALIZE BMS*************************************/
            case SYS_STATEMACH_INITIALIZE_BMS:
                SYS_SAVELASTSTATES();

                if (sys_state.substate == SYS_ENTRY) {
                    BMS_SetStateRequest(BMS_STATE_INIT_REQUEST);
```

```c
                        sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                        sys_state.substate = SYS_WAIT_INITIALIZATION_BMS;
                        sys_state.InitCounter = 0;
                        break;
                    } else if (sys_state.substate == SYS_WAIT_INITIALIZATION_BMS) {
                        bmsstate = BMS_GetInitializationState();
                        if (bmsstate == E_OK) {
                            sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                            sys_state.state = SYS_STATEMACH_RUNNING;
                            sys_state.substate = SYS_ENTRY;
                            break;
                        } else {
                            if (sys_state.InitCounter > (100/SYS_TASK_CYCLE_CONTEXT_MS)) {
                                sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                                sys_state.state = SYS_STATEMACH_ERROR;
                                sys_state.substate = SYS_BMS_INIT_ERROR;
                                break;
                            }
                            sys_state.timer = SYS_STATEMACH_SHORTTIME_MS;
                            sys_state.InitCounter++;
                            break;
                        }
                    }
                    break;

            /****************************RUNNNIG*************************************/
            case SYS_STATEMACH_RUNNING:
                SYS_SAVELASTSTATES();
                sys_state.timer = SYS_STATEMACH_LONGTIME_MS;
                break;

            /****************************ERROR*************************************/
            case SYS_STATEMACH_ERROR:
                SYS_SAVELASTSTATES();
                CANS_Enable_Periodic(TRUE);
                sys_state.timer = SYS_STATEMACH_LONGTIME_MS;
                break;
            /***************************DEFAULT CASE*************************************/
            default:
                /* This default case should never be entered.
                 * If we actually enter this case, it means that an
                 * unrecoverable error has occurred. Therefore the program
                 * will trap.
                 */
                configASSERT(0);
                break;
        }  /* end switch (sys_state.state) */
        sys_state.triggerentry--;
    }
```