```
/**
 *
 * @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.h
 * @author  foxBMS Team
 * @date    21.09.2015 (date of creation)
 * @ingroup ENGINE
 * @prefix  SYS
 *
 * @brief   Sys driver header
 *
 *
 */
```

```c

#ifndef SYS_H_
#define SYS_H_

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

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

/**
 * Symbolic names for busyness of the system
 */
typedef enum {
    SYS_CHECK_OK        = 0,    /*!< system ok      */
    SYS_CHECK_BUSY      = 1,    /*!< system busy    */
    SYS_CHECK_NOT_OK    = 2,    /*!< system not ok  */
} SYS_CHECK_e;



typedef enum {
  SYS_MODE_STARTUP_EVENT    = 0,    /*!< system startup                      */
/* SYS_MODE_EVENT_INIT      = 1,    !< todo                                 */
  SYS_MODE_CYCLIC_EVENT     = 2,    /*!< for cyclic events                   */
  SYS_MODE_TRIGGERED_EVENT  = 3,    /*!< for triggered events                */
  SYS_MODE_ABNORMAL_EVENT   = 4,    /*!< for abnormal (error etc.) events    */
  SYS_MODE_EVENT_RESERVED   = 0xFF, /*!< do not use                          */
} SYS_TRIG_EVENT_e;


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

/**
 * States of the SYS state machine
 */
typedef enum {
    /* Init-Sequence */
    SYS_STATEMACH_UNINITIALIZED                             = 0,    /*!<    */
    SYS_STATEMACH_INITIALIZATION                            = 1,    /*!<    */
    SYS_STATEMACH_INITIALIZED                               = 2,    /*!<    */
    SYS_STATEMACH_INITIALIZE_INTERLOCK                      = 4,    /*!<    */
    SYS_STATEMACH_INITIALIZE_CONTACTORS                     = 5,    /*!<    */
    SYS_STATEMACH_INITIALIZE_BALANCING                      = 6,    /*!<    */
    SYS_STATEMACH_INITIALIZE_BMS                            = 7,    /*!<    */
    SYS_STATEMACH_RUNNING                                   = 8,    /*!<    */
    SYS_STATEMACH_FIRST_MEASUREMENT_CYCLE                   = 9,    /*!<    */
    SYS_STATEMACH_INITIALIZE_MISC                           = 10,   /*!<    */
    SYS_STATEMACH_CHECK_CURRENT_SENSOR_PRESENCE             = 11,   /*!<    */
    SYS_STATEMACH_INITIALIZE_ISOGUARD                       = 12,   /*!<    */
    SYS_STATEMACH_ERROR                                     = 0xF0, /*!< Error-State:       */
} SYS_STATEMACH_e;

```

```

/**
 * Substates of the SYS state machine
 */
typedef enum {
    SYS_ENTRY                           = 0,    /*!< Substate entry state       */
    SYS_CHECK_ERROR_FLAGS               = 1,    /*!< Substate check if any error flag set   */
    SYS_CHECK_STATE_REQUESTS            = 2,    /*!< Substate check if there is a state request   */
    SYS_WAIT_INITIALIZATION_INTERLOCK   = 3,    /*!< Substate to wait for initialization of the interlock state
    machine   */
    SYS_WAIT_INITIALIZATION_CONT        = 4,    /*!< Substate to wait for initialization of the contactor state
    machine   */
    SYS_WAIT_INITIALIZATION_BAL         = 5,    /*!< Substate to wait for initialization of the balancing state
    machine   */
    SYS_WAIT_INITIALIZATION_BMS         = 6,    /*!< Substate to wait for initialization of the bms state machine   */
    SYS_WAIT_FIRST_MEASUREMENT_CYCLE    = 7,    /*!< Substate to wait for first measurement cycle to complete   */
    SYS_WAIT_CURRENT_SENSOR_PRESENCE    = 8,    /*!< Substate to wait for first measurement cycle to complete   */
    SYS_CONT_INIT_ERROR                 = 9,    /*!< Substate error of contactor state machine initialization   */
    SYS_BAL_INIT_ERROR                  = 10,    /*!< Substate error of balancing state machine initialization   */
    SYS_ILCK_INIT_ERROR                 = 11,   /*!< Substate error of contactor state machine initialization   */
    SYS_BMS_INIT_ERROR                  = 12,   /*!< Substate error of bms state machine initialization   */
    SYS_MEAS_INIT_ERROR                 = 13,   /*!< Substate error if first measurement cycle does not complete   */
    SYS_CURRENT_SENSOR_PRESENCE_ERROR   = 14,   /*!< Substate error if first measurement cycle does not complete   */
} SYS_STATEMACH_SUB_e;


/**
 * State requests for the SYS statemachine
 */
typedef enum {
    SYS_STATE_INIT_REQUEST                = SYS_STATEMACH_INITIALIZATION,           /*!<    */
    SYS_STATE_ERROR_REQUEST               = SYS_STATEMACH_ERROR,                    /*!<    */
    SYS_STATE_NO_REQUEST,                                                           /*!<    */
} SYS_STATE_REQUEST_e;


/**
 * Possible return values when state requests are made to the SYS statemachine
 */
typedef enum {
    SYS_OK                                 = 0,    /*!< CONT --> ok                             */
    SYS_BUSY_OK                            = 1,    /*!< CONT under load --> ok                  */
    SYS_REQUEST_PENDING                    = 2,    /*!< requested to be executed               */
    SYS_ILLEGAL_REQUEST                    = 3,    /*!< Request can not be executed            */
    SYS_ALREADY_INITIALIZED                = 30,   /*!< Initialization of LTC already finished */
    SYS_ILLEGAL_TASK_TYPE                  = 99,   /*!< Illegal                                */
} SYS_RETURN_TYPE_e;



/**
 * This structure contains all the variables relevant for the CONT state machine.
 * The user can get the current state of the CONT state machine with this variable
 */
typedef struct {
    uint16_t timer;                         /*!< time in ms before the state machine processes the next state, e.g.
    in counts of 1ms    */
    SYS_STATE_REQUEST_e statereq;           /*!< current state request made to the state
    machine                                                 */
    SYS_STATEMACH_e state;                  /*!< state of Driver State
    Machine                                                                 */
    SYS_STATEMACH_SUB_e substate;                       /*!< current substate of the state
    machine                                                         */
    SYS_STATEMACH_e laststate;              /*!< previous state of the state
    machine                                                           */
    SYS_STATEMACH_SUB_e lastsubstate;                   /*!< previous substate of the state
    machine                                                         */
    uint32_t ErrRequestCounter;             /*!< counts the number of illegal requests to the SYS state machine */
    uint16_t InitCounter;                   /*!< Timeout to wait for initialization of state machine state machine */
    uint8_t triggerentry;                   /*!< counter for re-entrance protection (function running flag) */
} SYS_STATE_s;


/*================== Function Prototypes ==================================*/
/**
 * @brief   sets the current state request of the state variable sys_state.
 *
 * @details This function is used to make a state request to the state machine, e.g., start
 *          voltage measurement, read result of voltage measurement, re-initialization.
 *          It calls SYS_CheckStateRequest() to check if the request is valid. The state request
 *          is rejected if is not valid. The result of the check is returned immediately, so that
 *          the requester can act in case it made a non-valid state request.
 *
 * @param   statereq state requested to set
 *
 * @return  If the request was successfully set, it returns the SYS_OK, else the current state of
 *          requests (type SYS_STATE_REQUEST_e)
 */
extern SYS_RETURN_TYPE_e SYS_SetStateRequest(SYS_STATE_REQUEST_e statereq);

/**
 * @brief   gets the current state.
 *
 * @details This function is used in the functioning of the SYS state machine.
 *
 * @return  current state, taken from SYS_STATEMACH_e
 */
extern SYS_STATEMACH_e SYS_GetState(void);

/**
 * @brief   trigger function for the SYS driver state machine.
 *
 * @details This function contains the sequence of events in the SYS state machine. It must be
 *          called time-triggered, every 1ms.
```

```c
 */
extern void SYS_Trigger(void);


#endif /* SYS_H_ */

```