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

```c

#ifndef BMS_H_
#define BMS_H_

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

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

/**
 * Symbolic names for battery system state
 */
typedef enum {
    BMS_CHARGING,      /*!< battery is charged */
    BMS_DISCHARGING,   /*!< battery is discharged */
    BMS_RELAXATION,    /*!< battery relaxation ongoing */
    BMS_AT_REST,       /*!< battery is resting */
} BMS_CURRENT_FLOW_STATE_e;


/**
 * Symbolic names for busyness of the syscontrol
 */
typedef enum {
    BMS_CHECK_OK        = 0,    /*!< syscontrol ok      */
    BMS_CHECK_BUSY      = 1,    /*!< syscontrol busy    */
    BMS_CHECK_NOT_OK    = 2,    /*!< syscontrol not ok  */
} BMS_CHECK_e;


typedef enum {
  BMS_MODE_STARTUP_EVENT    = 0,    /*!< syscontrol startup                 */
/*  BMS_MODE_EVENT_INIT      = 1,*/ /*!< todo                               */
  BMS_MODE_CYCLIC_EVENT     = 2,    /*!< for cyclic events                  */
  BMS_MODE_TRIGGERED_EVENT  = 3,    /*!< for triggered events               */
  BMS_MODE_ABNORMAL_EVENT   = 4,    /*!< for abnormal (error etc.) events   */
  BMS_MODE_EVENT_RESERVED   = 0xFF, /*!< do not use                         */
} BMS_TRIG_EVENT_e;


/**
 * States of the SYS state machine
 */
typedef enum {
    /* Init-Sequence */
    BMS_STATEMACH_UNINITIALIZED             = 0,    /*!<    */
    BMS_STATEMACH_INITIALIZATION            = 1,    /*!<    */
    BMS_STATEMACH_INITIALIZED               = 2,    /*!<    */
    BMS_STATEMACH_IDLE                      = 3,    /*!<    */
    BMS_STATEMACH_STANDBY                   = 4,    /*!<    */
    BMS_STATEMACH_PRECHARGE                 = 5,    /*!<    */
    BMS_STATEMACH_NORMAL                    = 6,    /*!<    */
    BMS_STATEMACH_CHARGE_PRECHARGE          = 7,    /*!<    */
    BMS_STATEMACH_CHARGE                    = 8,    /*!<    */
```
Add Engine related states
```c
    BMS_STATEMACH_UNDEFINED                 = 20,   /*!< undefined state                                    */
    BMS_STATEMACH_RESERVED1                 = 0x80, /*!< reserved state                                     */
    BMS_STATEMACH_ERROR                     = 0xF0, /*!< Error-State:  */
} BMS_STATEMACH_e;


/**
 * Substates of the SYS state machine
 */
typedef enum {
    BMS_ENTRY                                     = 0,    /*!< Substate entry state       */
    BMS_CHECK_ERROR_FLAGS_INTERLOCK               = 1,    /*!< Substate check measurements after interlock closed       */
    BMS_INTERLOCK_CHECKED                         = 2,    /*!< Substate interlocked checked       */
    BMS_CHECK_STATE_REQUESTS                      = 3,    /*!< Substate check if there is a state request   */
    BMS_CHECK_BALANCING_REQUESTS                  = 4,    /*!< Substate check if there is a balancing request   */
    BMS_CHECK_ERROR_FLAGS                         = 5,    /*!< Substate check if any error flag set   */
    BMS_CHECK_CONTACTOR_NORMAL_STATE              = 6,    /*!< Substate in precharge, check if there contactors reached normal   */
    BMS_CHECK_CONTACTOR_CHARGE_STATE              = 7,    /*!< Substate in precharge, check if there contactors reached normal   */
```
Add Engine related state
```c
    BMS_OPEN_INTERLOCK                            = 8,    /*!< Substate in error to open interlock after contactors have been opened   */
    BMS_CHECK_INTERLOCK_CLOSE_AFTER_ERROR         = 9,    /*!< Substate in error to close interlock after all error flags were reset   */
} BMS_STATEMACH_SUB_e;


/**
 * State requests for the SYS statemachine
 */
typedef enum {
    BMS_STATE_INIT_REQUEST                  = BMS_STATEMACH_INITIALIZATION,           /*!<    */
    BMS_STATE_ERROR_REQUEST                 = BMS_STATEMACH_ERROR,    /*!<    */
    BMS_STATE_NO_REQUEST                    = BMS_STATEMACH_RESERVED1,                /*!<    */
} BMS_STATE_REQUEST_e;


/**
 * Possible return values when state requests are made to the SYS statemachine
 */
typedef enum {
    BMS_OK                                  = 0,    /*!< CONT --> ok                             */
    BMS_BUSY_OK                             = 1,    /*!< CONT under load --> ok                  */
    BMS_REQUEST_PENDING                     = 2,    /*!< requested to be executed               */
    BMS_ILLEGAL_REQUEST                     = 3,    /*!< Request can not be executed            */
    BMS_ALREADY_INITIALIZED                 = 30,   /*!< Initialization of LTC already finished */
    BMS_ILLEGAL_TASK_TYPE                   = 99,   /*!< Illegal                                */
} BMS_RETURN_TYPE_e;
```

```c


/**
 * 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    */
    BMS_STATE_REQUEST_e statereq;               /*!< current state request made to the state
    machine                                        */
    BMS_STATEMACH_e state;                      /*!< state of Driver State
    Machine                                                          */
    BMS_STATEMACH_SUB_e substate;               /*!< current substate of the state
    machine                                                  */
    BMS_CURRENT_FLOW_STATE_e currentFlowState;  /*!< state of battery
    system                                                                */
    BMS_STATEMACH_e laststate;                  /*!< previous state of the state
    machine                                                    */
    BMS_STATEMACH_SUB_e lastsubstate;           /*!< previous substate of the state
    machine                                                 */
    uint32_t ErrRequestCounter;                 /*!< counts the number of illegal requests to the LTC state
    machine                         */
    STD_RETURN_TYPE_e initFinished;             /*!< #E_OK if the initialization has passed, #E_NOT_OK
    otherwise                            */
    uint8_t triggerentry;                       /*!< counter for re-entrance protection (function running
    flag)                             */
    uint32_t restTimer_ms;                     /*!< timer until battery system is at
    rest                                                  */
    uint8_t counter;                            /*!< general purpose
    counter                                                               */
} BMS_STATE_s;


/*================== Function Prototypes ==================================*/
/**
 * @brief   sets the current state request of the state variable bms_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 BMS_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 request to set
 *
 * @return  current state request, taken from BMS_STATE_REQUEST_e
 */
extern BMS_RETURN_TYPE_e BMS_SetStateRequest(BMS_STATE_REQUEST_e statereq);

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

/**
 * @brief   Gets the initialization state.
 *
 * This function is used for getting the BMS initialization state.
 *
 * @return  #E_OK if initialized, otherwise #E_NOT_OK
 */
STD_RETURN_TYPE_e BMS_GetInitializationState(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.
 */
extern void BMS_Trigger(void);


/**
 * @brief   Returns current battery system state (charging/discharging,
 *          resting or in relaxation phase)
 *
 * @return  BS_CHARGING, BS_DISCHARGING, BS_RELAXATION or BS_AT_REST
 */
extern BMS_CURRENT_FLOW_STATE_e BMS_GetBatterySystemState(void);

#endif /* BMS_H_ */
```