Design document

Small OS Design (sos)

TEAM 2

Prepared by:

Amr Gamal El-Abd

Ahmed Atef

Anas Mohmoud

Khaled Mustafa

Table of Contents: -

Subject **Page**

Project introduction:

The SOS (Simple Operating System) plays a vital role within the layered architecture of our embedded system project. Positioned between the hardware-specific layers and the application layer, the SOS abstraction layer provides a unified interface for interacting with the underlying operating system functionalities. It shields the upper layers from the complexities of hardware-dependent operations, enabling developers to write portable and platform-independent code. The SOS abstraction layer promotes modularity, reusability, and maintainability, facilitating the seamless integration of various software components within the system project.

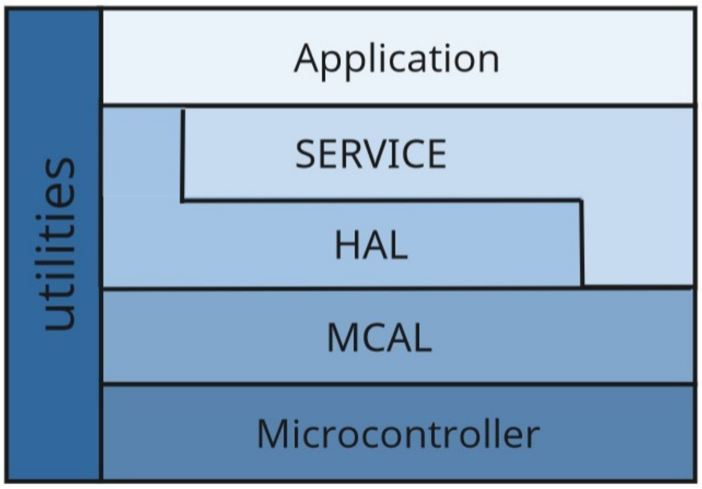
Main Application Flow :

1. Implement an application that calls the SOS module and use 2 tasks

- Task 1: Toggle LED_0 (Every 300 Milli-Seconds)
- Task 2: Toggle LED_1 (Every 500 Milli-Seconds)

2. Make sure that these tasks occur periodically and forever
3. When pressing PBUTTON0, the SOS will stop
4. When Pressing PBUTTON1, the SOS will run.

Layered Architectures:-



Application Layer: This is the topmost layer of the software stack, which contains the actual application logic. It interacts with the lower layers to perform its tasks. It is responsible for implementing the desired functionality of the system.

SERVICE Layer: This layer plays a key role in abstracting and encapsulating the low-level hardware details provided by the MCAL. It provides a set of services, functions, and interfaces that enable the application layer to interact with the underlying hardware

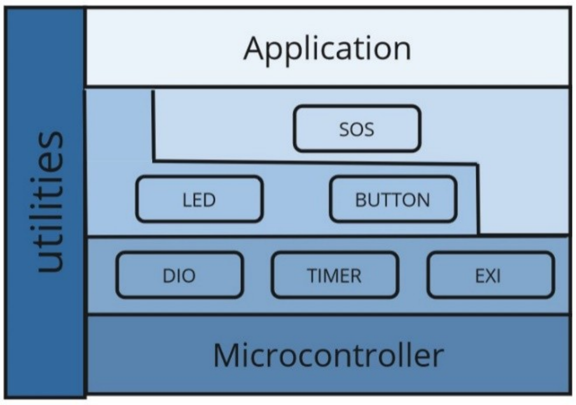
HAL Layer: This layer provides an abstraction for external devices connected to the microcontroller. The HAL layer provides interface to access external devices and hides the implementation details from the application layer.

MCAL Layer (Microcontroller Abstraction Layer): This layer provides an abstraction for the microcontroller hardware. It includes low-level drivers for peripherals. It hides the hardware details and provides a uniform interface to the upper layers.

Utilities Layer: the utilities layer includes memory mapping, standard types, and utils.h. Memory mapping involves defining the memory layout and addresses for different components. Standard types provide a set of predefined data types that ensure consistency and portability across different platforms. The utils.h header file contains utility functions and macros that offer commonly used functionalities, such as bit manipulation.

Microcontroller: This layer represents the physical hardware layer consisting of the microcontroller chip. The microcontroller is responsible for executing the code stored in its memory and controlling the behavior of the system.

Modules description:



SOS Module:

Description: The Simple Operating System (SOS) module is a critical component within the framework of our embedded system project. Designed to provide a streamlined and efficient operating environment, SOS serves as the core software layer responsible for managing system resources, facilitating task scheduling, and enabling seamless communication between various hardware and software components.

LED Module:

Description: The LED driver is responsible for setting up and controlling the LEDs of the microcontroller. It contains different LED states like ON, OFF and toggle.

Button Module:

Description: The Button driver is responsible for setting up and controlling the buttons of the microcontroller. This driver will be used to detect button presses.

DIO Module:

Description: The DIO (Digital Input Output) driver is responsible for setting up the digital pins of the microcontroller to either input or output mode.

Timer Module:

Description: The Timer module is responsible for setting up and controlling the timers of the microcontroller. This driver will be used to create the timing functionalities required in the project.

Interrupt Module:

Description: The Interrupt module is responsible for setting up and controlling the interrupts of the microcontroller. This driver will be used to set the interrupt functionalities.

Drivers' documentation:-

Service layer:

SOS Driver:

Description: The Simple Operating System (SOS) module is a critical component within the framework of our embedded system project. Designed to provide a streamlined and efficient operating environment, SOS serves as the core software layer responsible for managing system resources, facilitating task scheduling, and enabling seamless communication between various hardware and software components.

Functions:

```
enu_sos_Status_t sos_Init() : enu_sos_Status_t;
enu_sos_Status_t sos_CreateTask(u8 u8_Priority, str_sos_configTask_t* str_sos_configTask);
enu_sos_Status_t sos_DeleteTask(u8 u8_Priority) ;
enu_sos_Status_t sos_Run(void) ;
enu_sos_Status_t sos_Deinit(void) ;
enu_sos_Status_t sos_modify_task(u8 u8_Priority, str_sos_configTask_t* str_sos_configTask) ;
enu_sos_Status_t sos_disable(void) ;
```

HAL drivers:

1. LED Driver:

Description: The LED driver is responsible for setting up and controlling the LEDs of the microcontroller. It contains different LED states like ON, OFF and toggle.

Functions:

```
LED_ERROR_TYPE LED_INIT(DIO_PIN_TYPE PIN);
LED_ERROR_TYPE LED_ON(DIO_PIN_TYPE PIN);
LED_ERROR_TYPE LED_OFF(DIO_PIN_TYPE PIN);
```

2. **Button Driver**:

Description: The Button driver is responsible for setting up and controlling the buttons of the microcontroller. This driver will be used to detect button presses.

Functions:

```
BUTTON_ERROR_TYPE Button_INIT(DIO_PIN_TYPE PIN);
BUTTON_ERROR_TYPE Button_read(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE*VOLT);
```

MCAL drivers:

1. **DIO Driver**:

Description: The DIO (Digital Input Output) driver is responsible for setting up the digital pins of the microcontroller to either input or output mode. This driver will be used to control the buttons and LEDs.

Functions:

```
DIO_ERROR_TYPE DIO_INITPIN(DIO_PIN_TYPE PIN,DIO_PINSTATUS_TYPE STATUS);
DIO_ERROR_TYPE DIO_WRITEPIN(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE VOLTAGE);
DIO_ERROR_TYPE DIO_READPIN(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE* VOLT);
void DIO_TogglePin(DIO_PIN_TYPE pin);
```

2. **Interrupt Driver**:

Description: The Interrupt driver is responsible for setting up and controlling the interrupts of the microcontroller. This driver will be used to detect button presses.

Functions:

```
EN_int__error_t EXI_Enable (EN_int_t Interrupt);
EN_int__error_t EXI_Disable (EN_int_t Interrupt);
EN_int__error_t EXI_Trigger(EN_int_t Interrupt,EN_trig trigger);
void EXI_SetCallBack(EN_int_t Interrupt,void(*ptrf)(void));
```

3. **Timer Driver**:

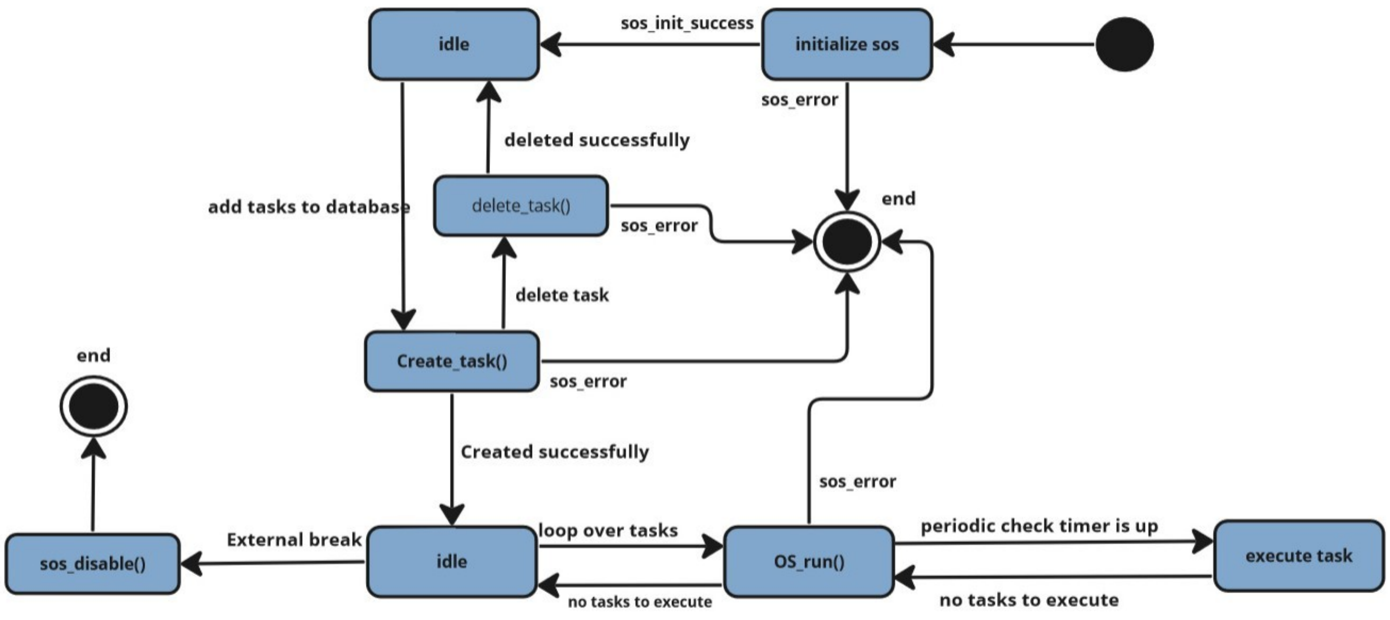
Description: The Timer driver is responsible for setting up and controlling the timers of the microcontroller. This driver will be used to create the timing delays required in the project.

Functions:

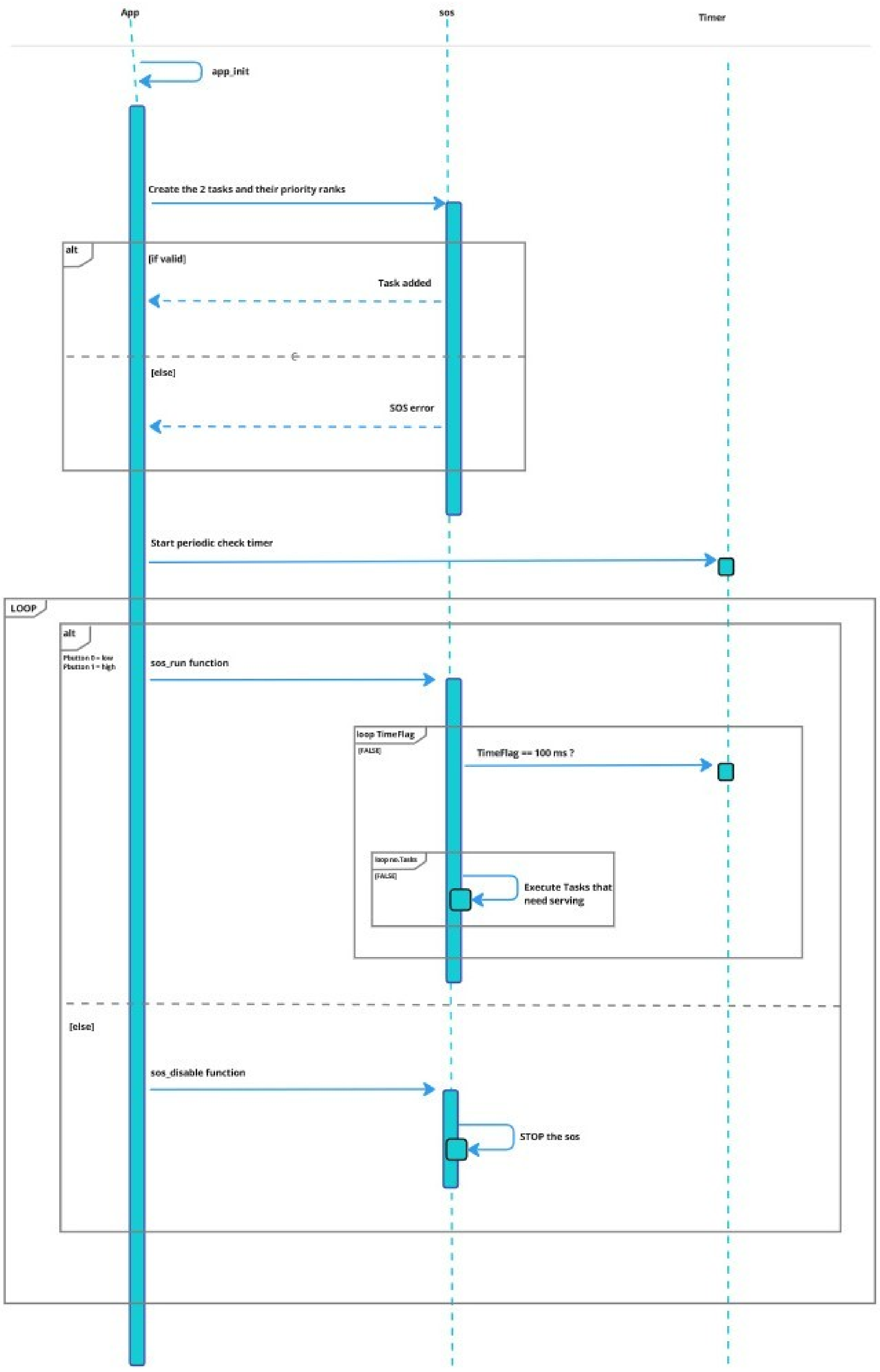
```
Timer_ErrorStatus TIMER_2_init(Timer_Mode mode);
Timer_ErrorStatus TIMER_2_start(Timer_Prescaler prescaler);
void TIMER_2_stop(void);
```

```
Timer_ErrorStatus TIMER_2_setIntialValue(uint8_t value);
Timer_ErrorStatus TIMER_2_OvfNum(double overflow);
void TIMER_2_DELAY_MS(double _delay);
void TIMER_2_INT();
```

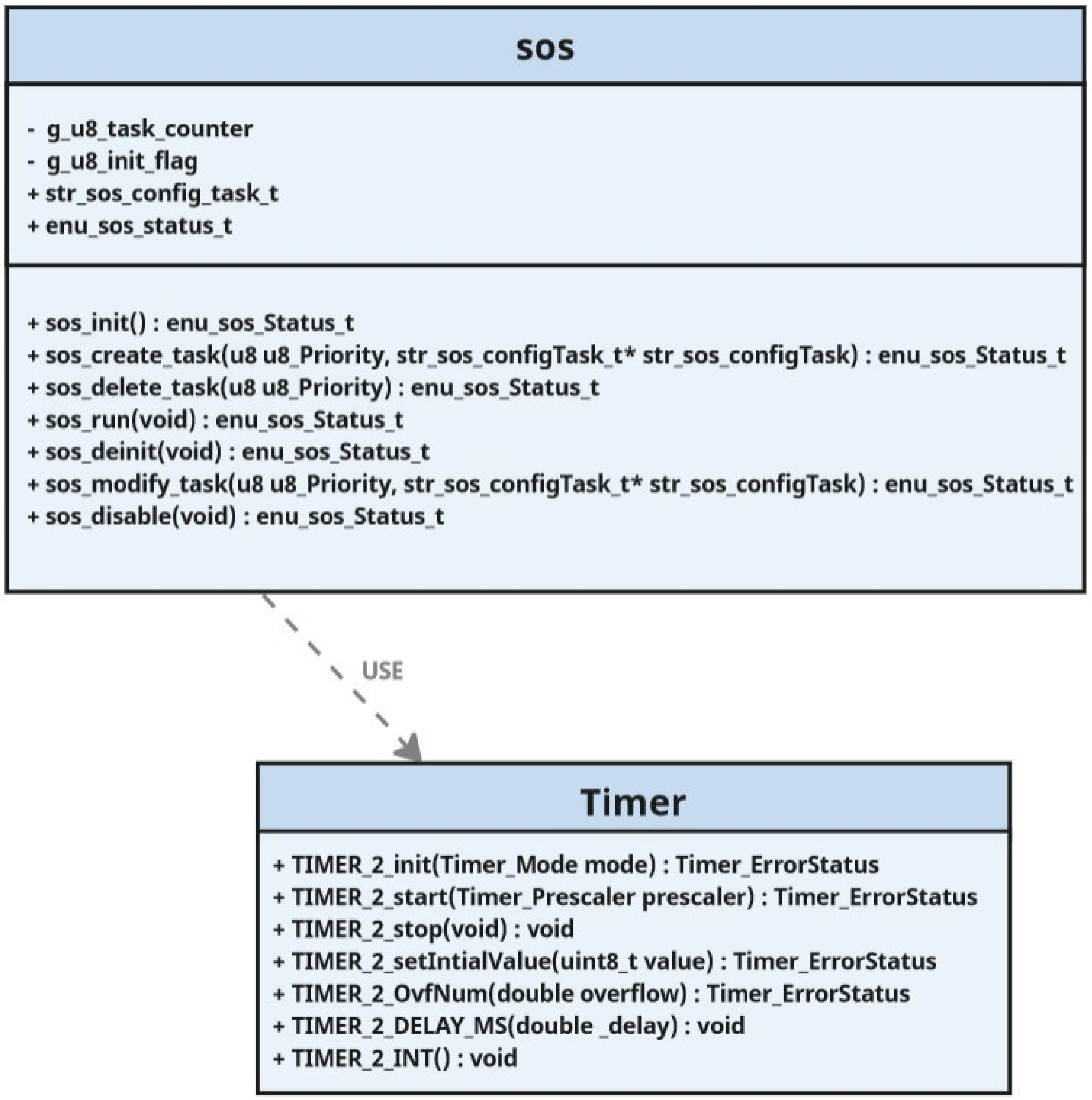
State machine for SOS module:



<u>Sequence Diagram for APP:</u>



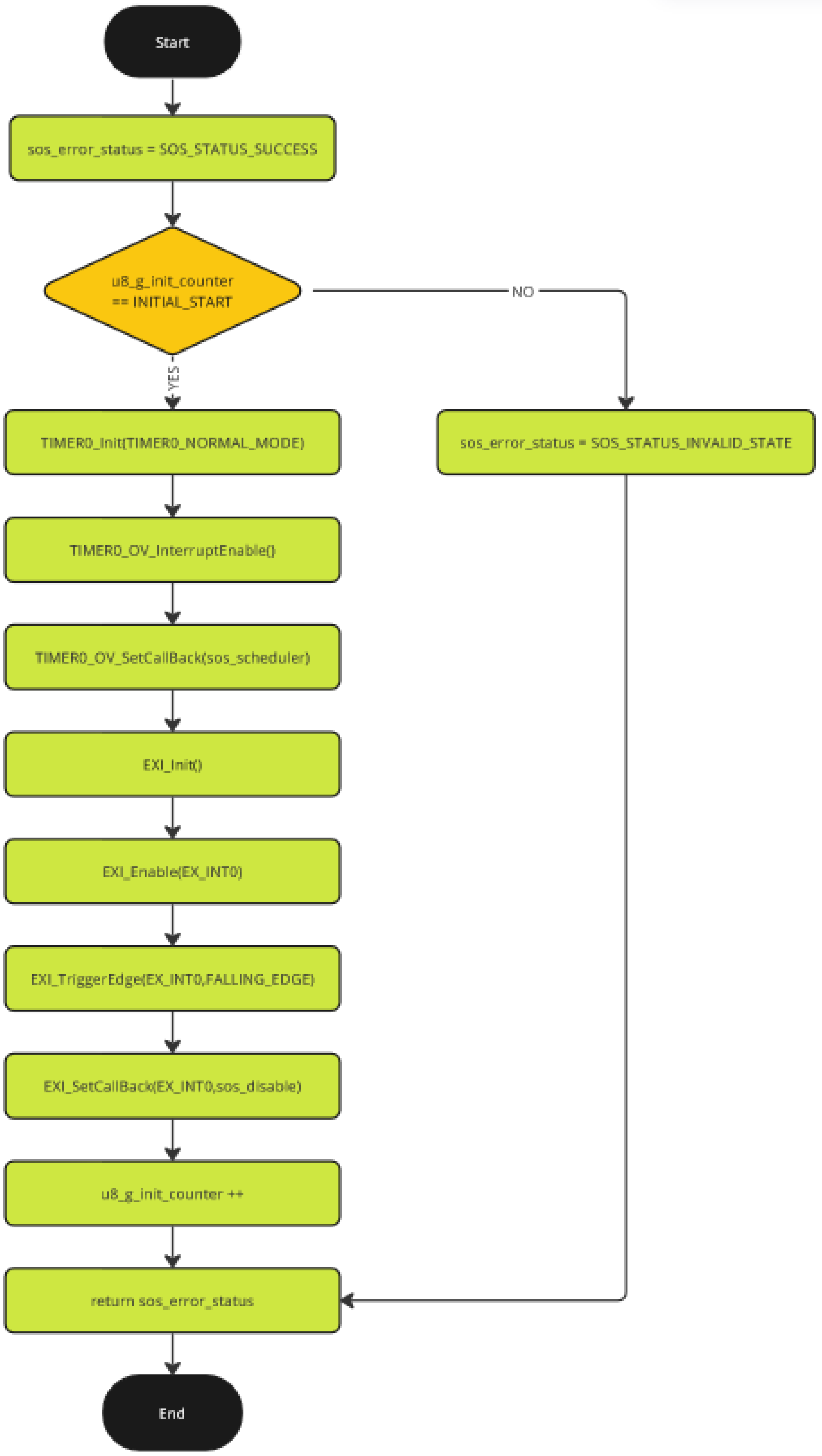
Class diagram for SOS module:


sos
- g_u8_task_counter
- g_u8_init_flag
+ str_sos_config_task_t
+ enu_sos_status_t
+ sos_init() : enu_sos_Status_t
+ sos_create_task(u8 u8_Priority, str_sos_configTask_t* str_sos_configTask) : enu_sos_Status_t
+ sos_delete_task(u8 u8_Priority) : enu_sos_Status_t
+ sos_run(void) : enu_sos_Status_t
+ sos_deinit(void) : enu_sos_Status_t
+ sos_modify_task(u8 u8_Priority, str_sos_configTask_t* str_sos_configTask) : enu_sos_Status_t
+ sos_disable(void) : enu_sos_Status_t
USE
Timer
+ TIMER_2_init(Timer_Mode mode) : Timer_ErrorStatus
+ TIMER_2_start(Timer_Prescaler prescaler) : Timer_ErrorStatus
+ TIMER_2_stop(void) : void
+ TIMER_2_setIntialValue(uint8_t value) : Timer_ErrorStatus
+ TIMER_2_OvfNum(double overflow) : Timer_ErrorStatus
+ TIMER_2_DELAY_MS(double _delay) : void
+ TIMER_2_INT() : void

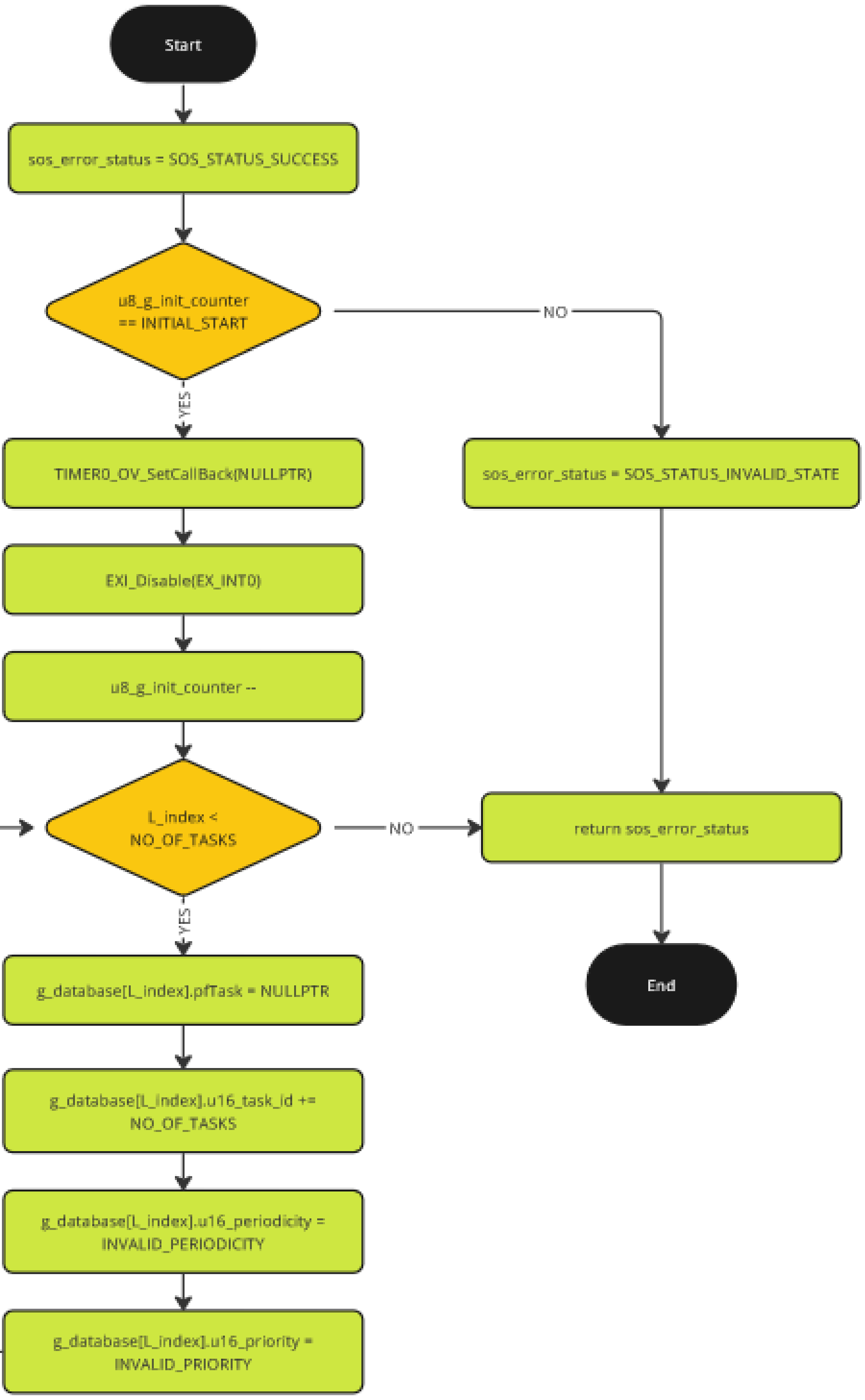
Modules' Flowcharts:

SOS API

1. enu_sos_error_status_t sos_init (void)

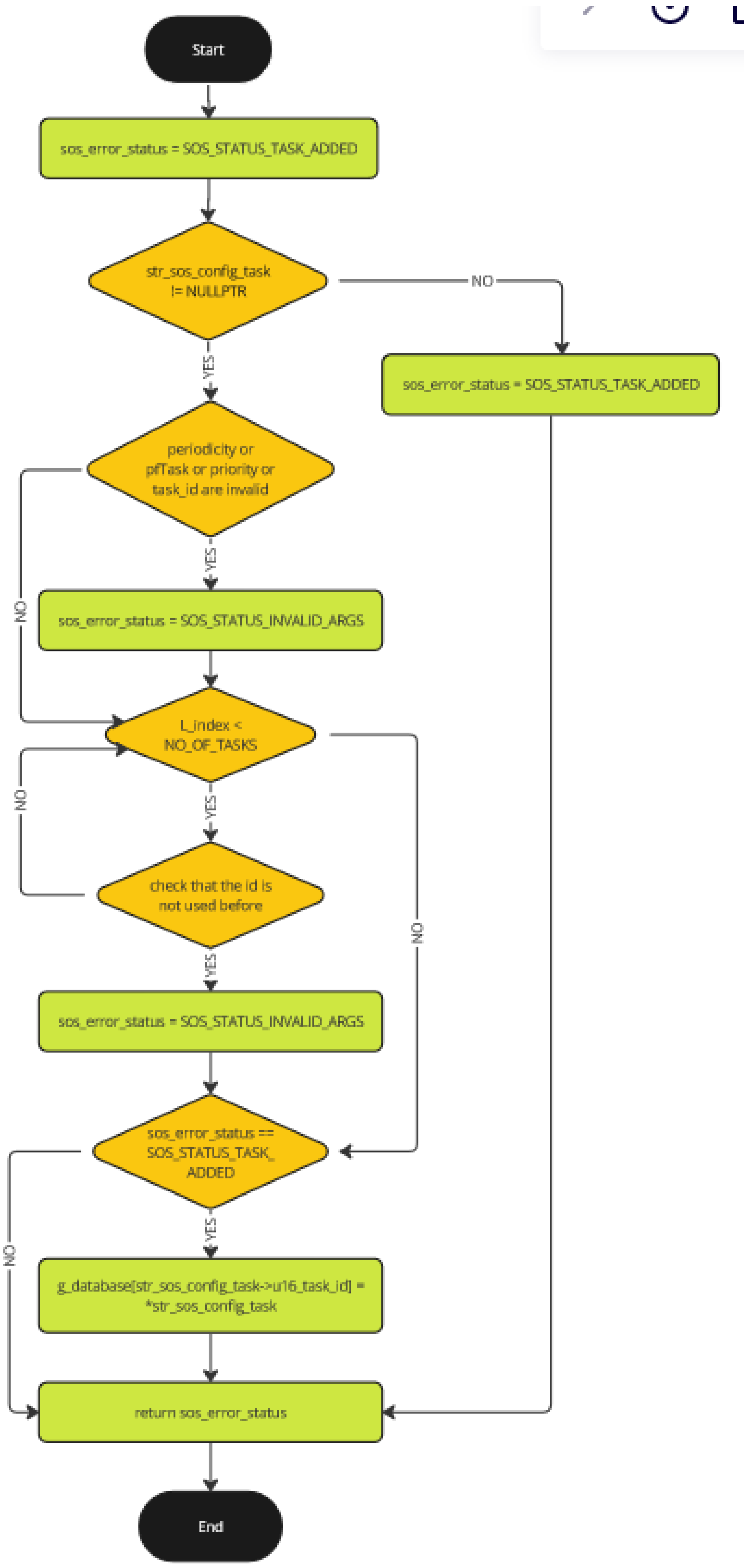

Start
sos_error_status = SOS_STATUS_SUCCESS
u8_g_init_counter == INITIAL_START
NO
YES
TIMER0_Init(TIMER0_NORMAL_MODE)
sos_error_status = SOS_STATUS_INVALID_STATE
TIMER0_OV_InterruptEnable()
TIMER0_OV_SetCallBack(sos_scheduler)
EXI_Init()
EXI_Enable(EX_INT0)
EXI_TriggerEdge(EX_INT0,FALLING_EDGE)
EXI_SetCallBack(EX_INT0,sos_disable)
u8_g_init_counter ++
return sos_error_status
End

2. enu_sos_error_status_t sos_deinit (void)



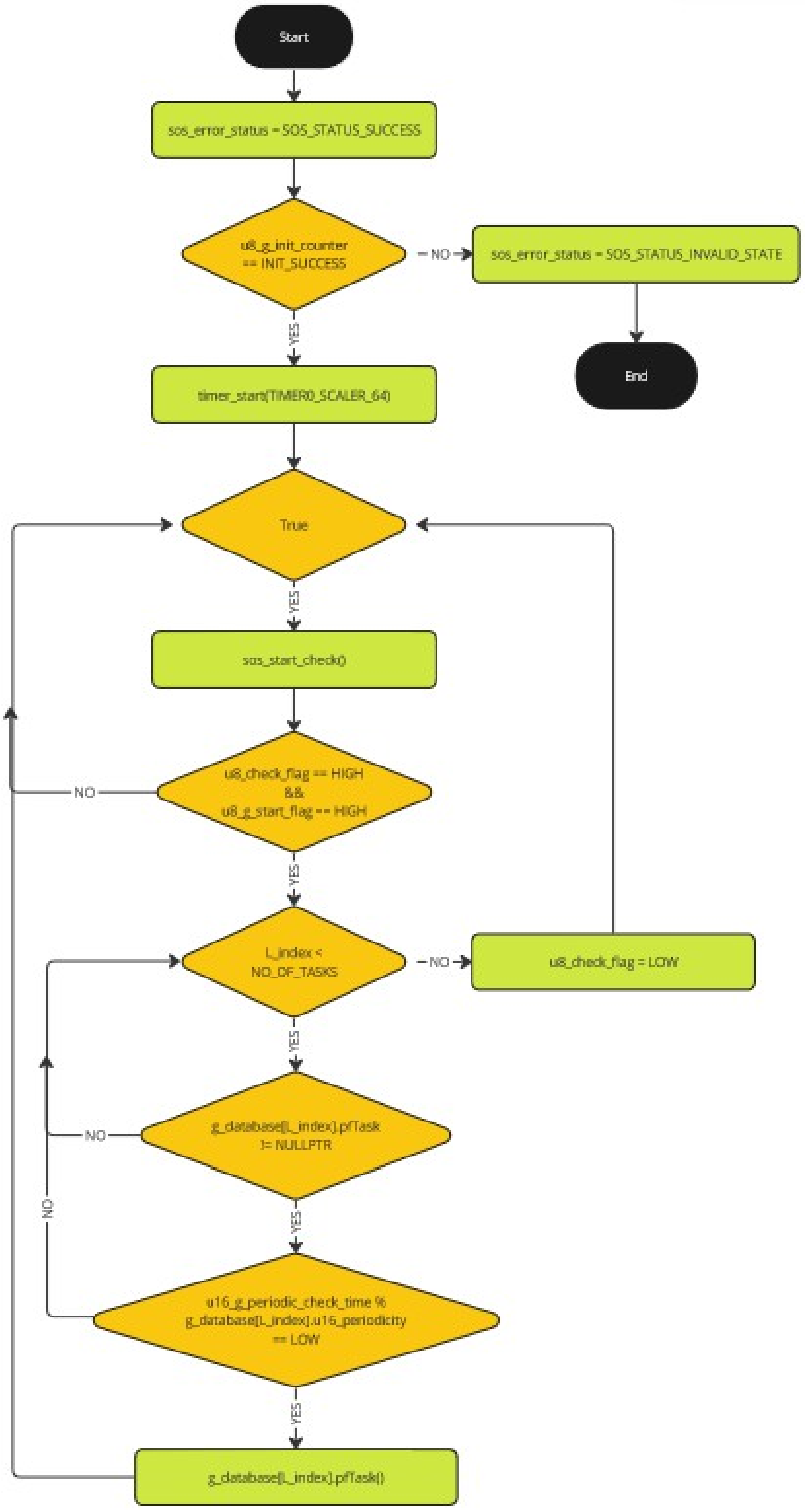
Start
sos_error_status = SOS_STATUS_SUCCESS
u8_g_init_counter == INITIAL_START
NO
YES
TIMER0_OV_SetCallBack(NULLPTR)
sos_error_status = SOS_STATUS_INVALID_STATE
EXI_Disable(EX_INT0)
u8_g_init_counter --
L_index < NO_OF_TASKS
NO
return sos_error_status
YES
End
g_database[L_index].pfTask = NULLPTR
g_database[L_index].u16_task_id += NO_OF_TASKS
g_database[L_index].u16_periodicity = INVALID_PERIODICITY
g_database[L_index].u16_priority = INVALID_PRIORITY

3. enu_sos_error_status_t sos_create_task (str_sos_config_task_t* str_sos_config_task)

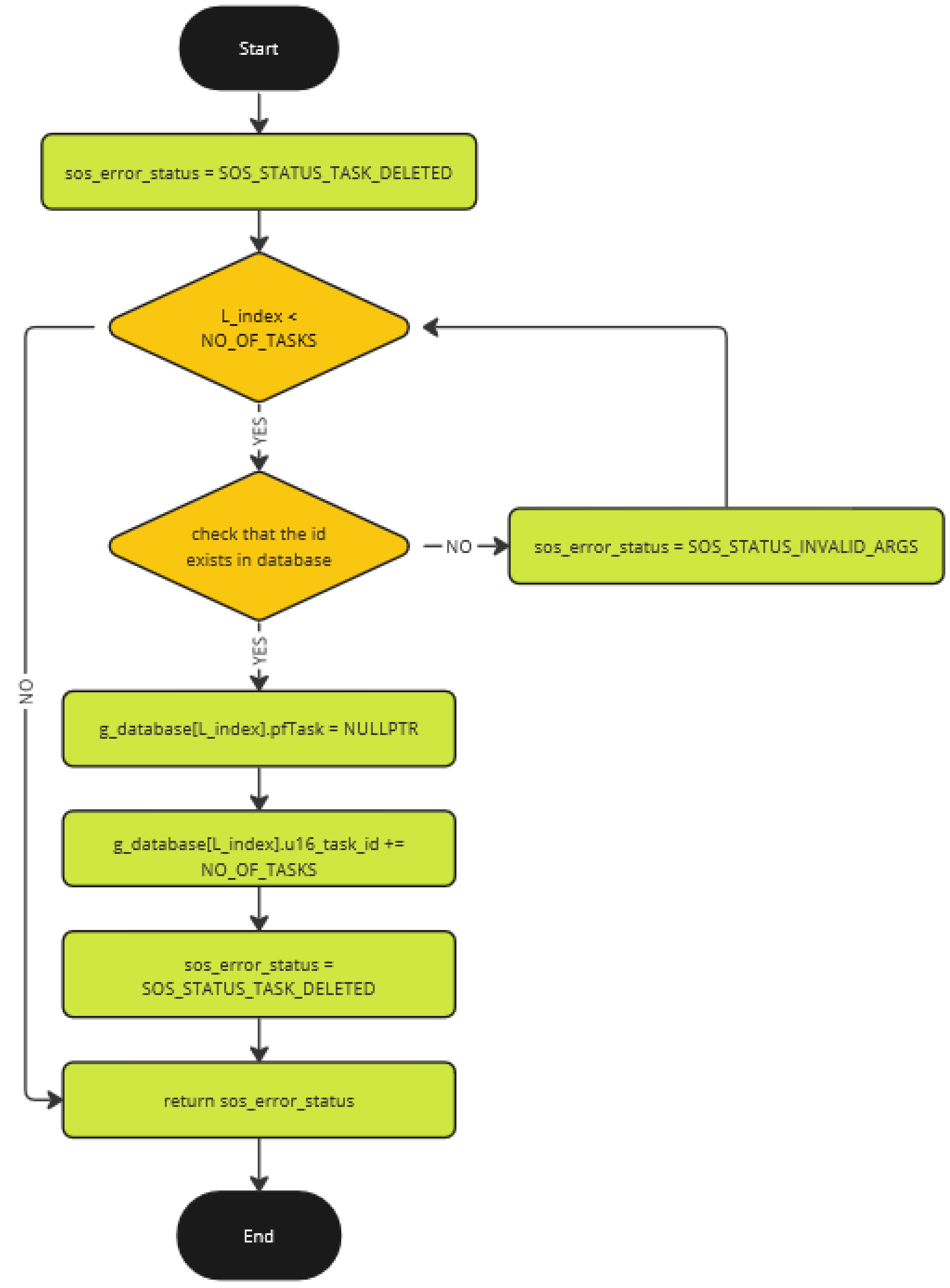


Start
sos_error_status = SOS_STATUS_TASK_ADDED
str_sos_config_task != NULLPTR
NO
sos_error_status = SOS_STATUS_TASK_ADDED
YES
periodicity or pfTask or priority or task_id are invalid
YES
NO
sos_error_status = SOS_STATUS_INVALID_ARGS
l_index < NO_OF_TASKS
NO
YES
check that the id is not used before
NO
YES
sos_error_status = SOS_STATUS_INVALID_ARGS
sos_error_status == SOS_STATUS_TASK_ADDED
NO
YES
g_database[str_sos_config_task->u16_task_id] = *str_sos_config_task
return sos_error_status
End

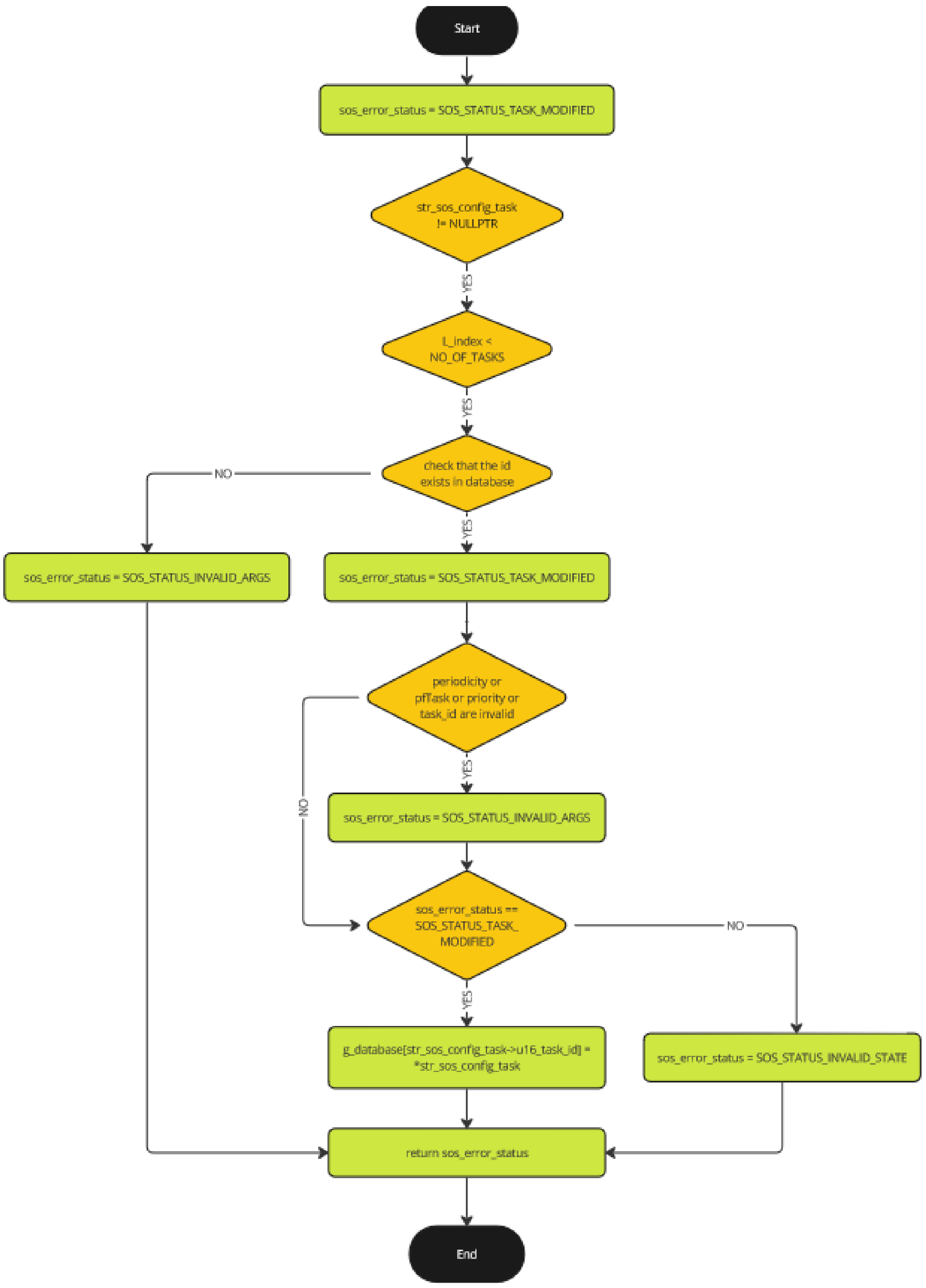
4. enu_sos_error_status_t sos_run (void)


Start
sos_error_status = SOS_STATUS_SUCCESS
u8_g_init_counter == INIT_SUCCESS
NO
sos_error_status = SOS_STATUS_INVALID_STATE
End
YES
timer_start(TIMER0_SCALER_64)
True
YES
sos_start_check()
u8_check_flag == HIGH && u8_g_start_flag == HIGH
NO
YES
l_index < NO_OF_TASKS
NO
u8_check_flag = LOW
YES
g_database[l_index].pfTask != NULLPTR
NO
YES
u16_g_periodic_check_time % g_database[l_index].u16_periodicity == LOW
NO
YES
g_database[l_index].pfTask()

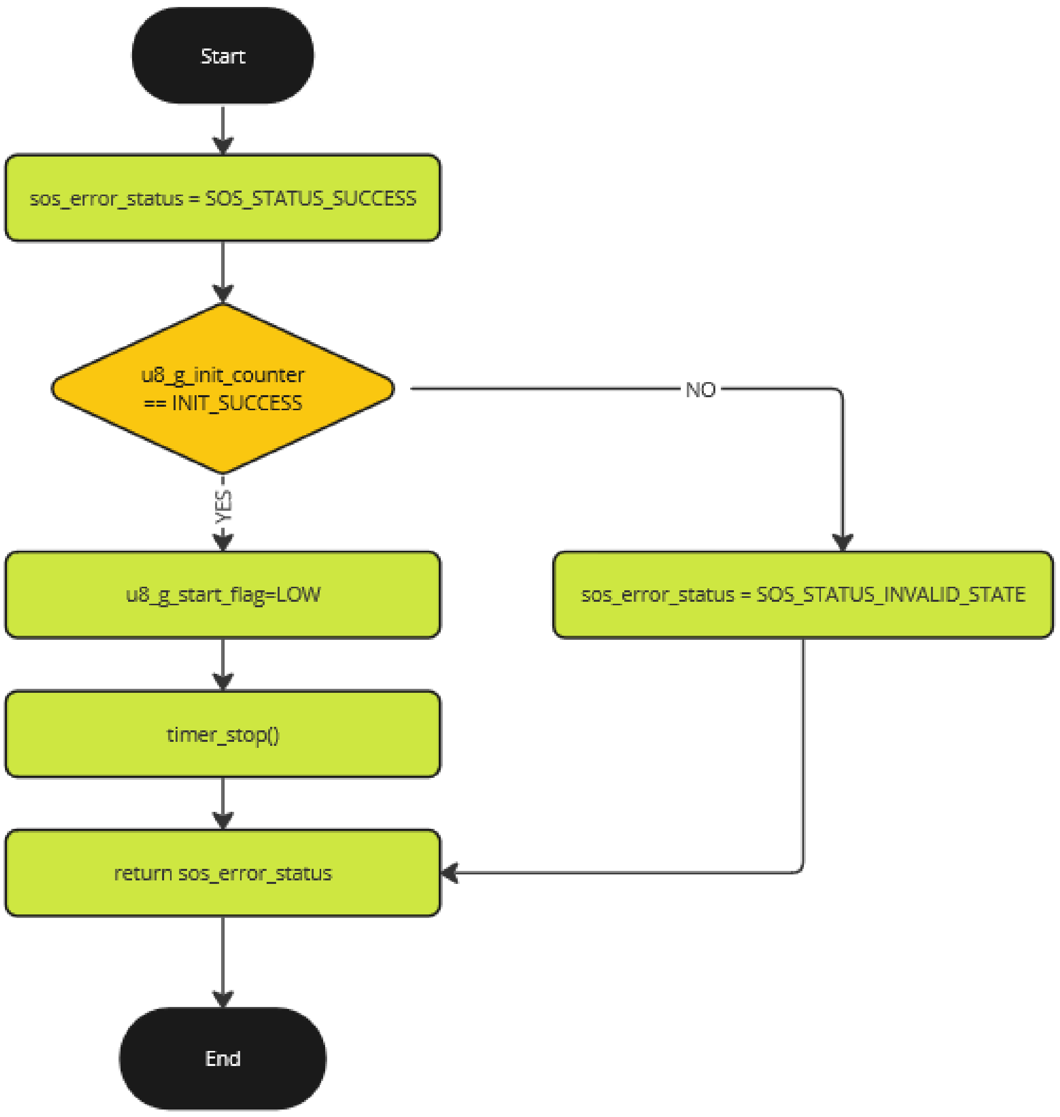
5. enu_sos_error_status_t sos_delete_task (u16 u16_task_id)


Start
sos_error_status = SOS_STATUS_TASK_DELETED
L_index < NO_OF_TASKS
YES
NO
check that the id exists in database
NO
sos_error_status = SOS_STATUS_INVALID_ARGS
YES
g_database[L_index].pfTask = NULLPTR
g_database[L_index].u16_task_id += NO_OF_TASKS
sos_error_status = SOS_STATUS_TASK_DELETED
return sos_error_status
End

6. enu_sos_error_status_t sos_modify_task (str_sos_config_task_t* str_sos_config_task)

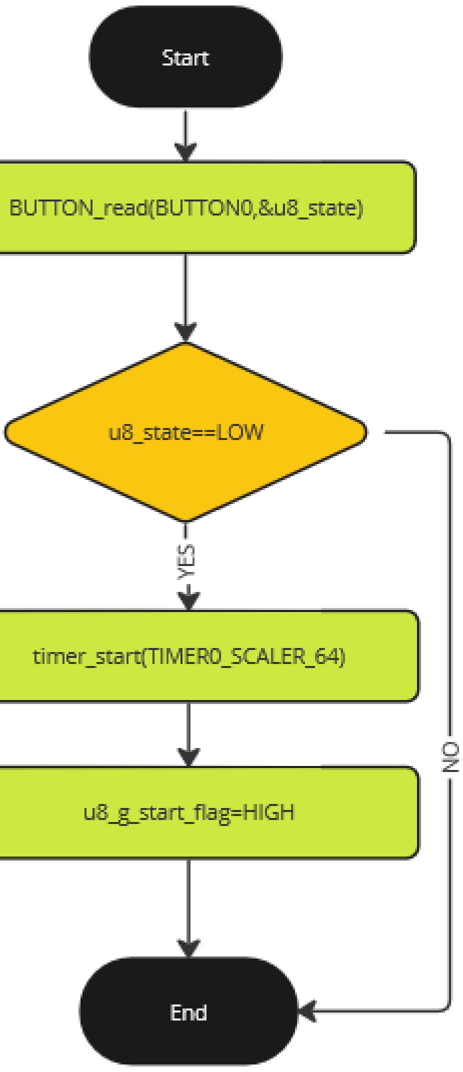

Start
sos_error_status = SOS_STATUS_TASK_MODIFIED
str_sos_config_task != NULLPTR
YES
l_index < NO_OF_TASKS
YES
check that the id exists in database
NO
YES
sos_error_status = SOS_STATUS_INVALID_ARGS
sos_error_status = SOS_STATUS_TASK_MODIFIED
periodicity or pfTask or priority or task_id are invalid
NO
YES
sos_error_status = SOS_STATUS_INVALID_ARGS
sos_error_status == SOS_STATUS_TASK_MODIFIED
NO
YES
g_database[str_sos_config_task->u16_task_id] = *str_sos_config_task
sos_error_status = SOS_STATUS_INVALID_STATE
return sos_error_status
End

7. enu_sos_error_status_t sos_disable(void)

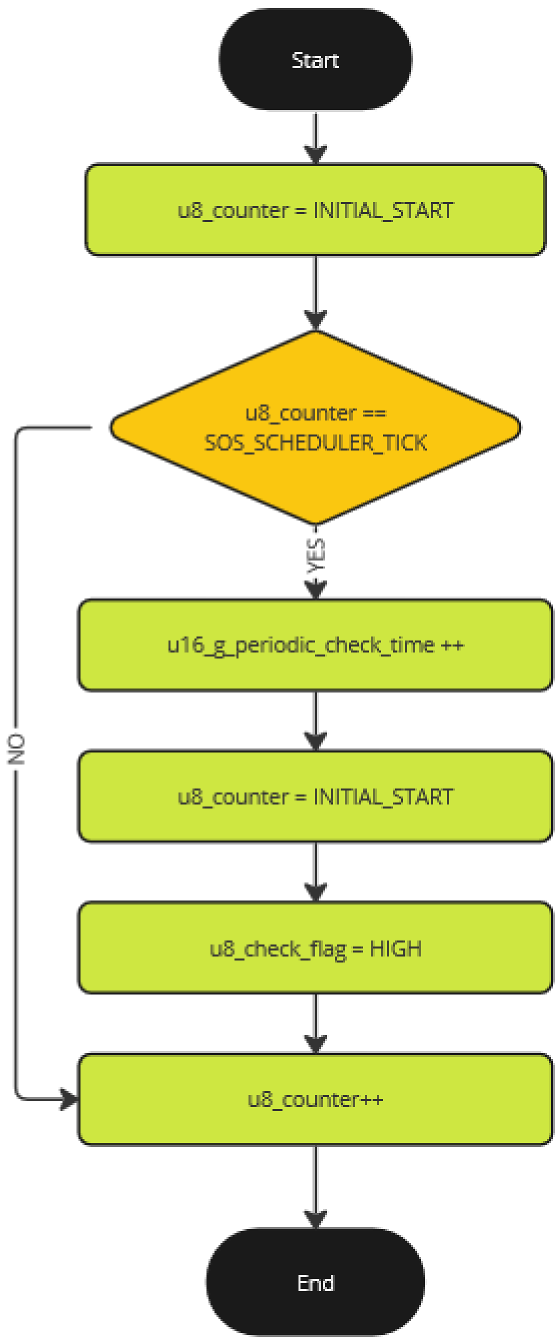


Start
sos_error_status = SOS_STATUS_SUCCESS
u8_g_init_counter == INIT_SUCCESS
NO
YES
u8_g_start_flag=LOW
sos_error_status = SOS_STATUS_INVALID_STATE
timer_stop()
return sos_error_status
End

8. void sos_start_check (void)

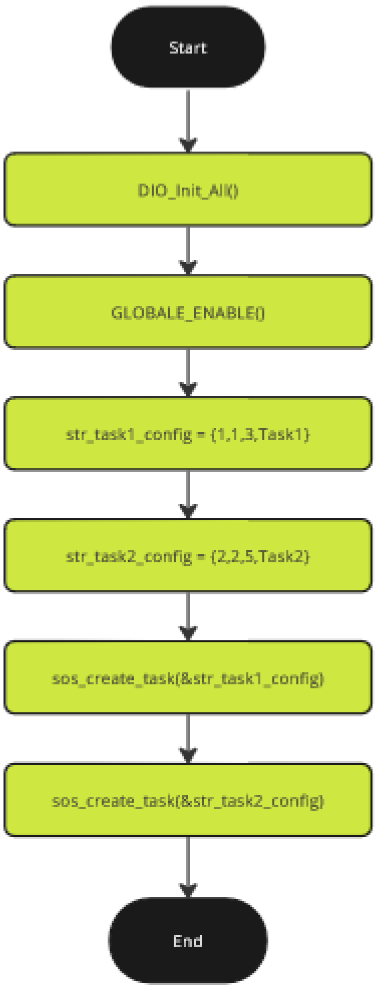


9. void sos_scheduler (void)

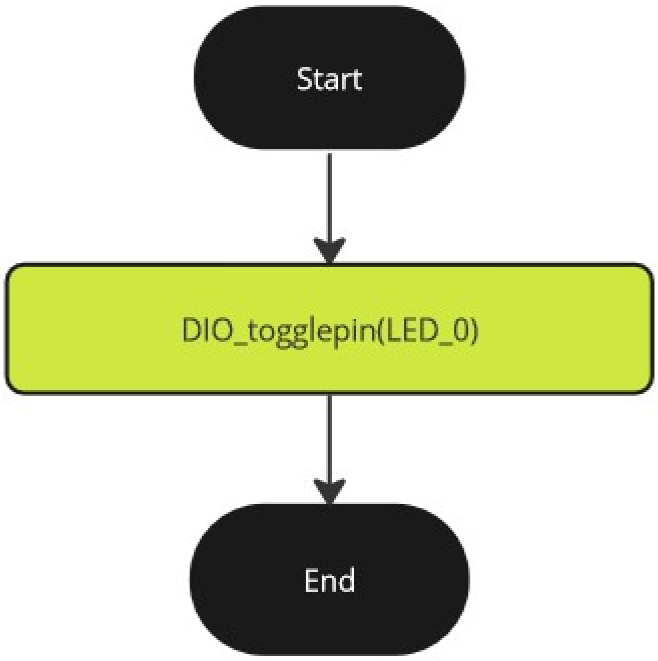

Start
u8_counter = INITIAL_START
u8_counter == SOS_SCHEDULER_TICK
YES
NO
u16_g_periodic_check_time ++
u8_counter = INITIAL_START
u8_check_flag = HIGH
u8_counter++
End

Application APIs

1. void app_init ();

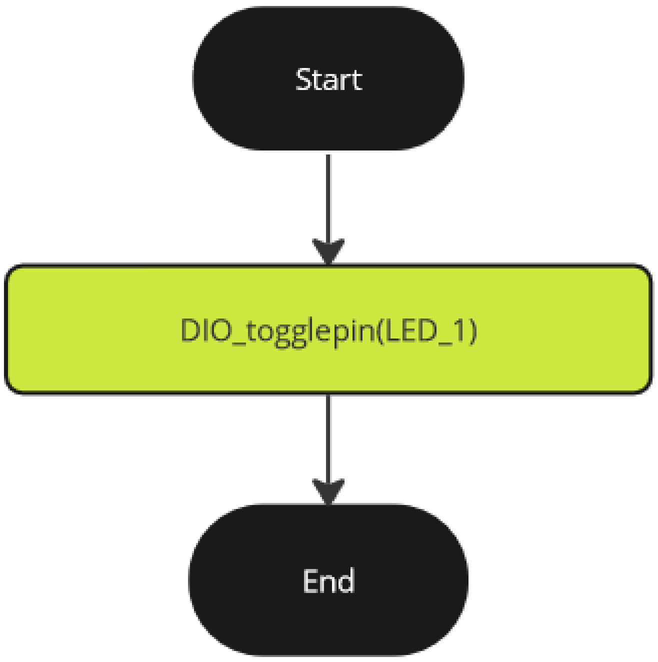

Start
DIO_Init_All()
GLOBALE_ENABLE()
str_task1_config = {1,1,3,Task1}
str_task2_config = {2,2,5,Task2}
sos_create_task(&str_task1_config)
sos_create_task(&str_task2_config)
End

2. void Task1(void);



3. void

Task2(void)



Pre-compiled configurations:

1. SOS:

```
#define NO_OF_TASKS                    10
#define SOS_SCHEDULER_TICK             100
#define INVALID_PRIORITY               0
#define INVALID_PERIODICITY            0
#define INITIAL_START                  0
#define INIT_SUCCESS                   1
#define BUTTON0                        PIND0
#define LED_0                          PINB0
#define LED_1                          PINB1
```

2. Timer

```
const DIO_PinStatus_type  PinsStatusArray[TOTAL_PINS]={
        INFREE,         /* Port A Pin 0 ADC0*/
        INFREE,         /* Port A Pin 1 ADC1*/
        INFREE,         /* Port A Pin 2 */
        INFREE,         /* Port A Pin 3 */
        INFREE,         /* Port A Pin 4 */
        INFREE,         /* Port A Pin 5 */
        INFREE,         /* Port A Pin 6 */
        INFREE,         /* Port A Pin 7 ADC7*/
        OUTPUT,         /* Port B Pin 0   / */
        OUTPUT,         /* Port B Pin 1   /*/
        INFREE,          /* Port B Pin 2 / INT2*/
        INFREE,          /* Port B Pin 3   /OC0*/
        INFREE,          /* Port B Pin 4 /ss*/
        INFREE,          /* Port B Pin 5 //mosi*/
        INFREE,         /* Port B Pin 6 /miso*/
        INFREE,          /* Port B Pin 7 clk*/
        INFREE,          /* Port C Pin 0 */
        INFREE,          /* Port C Pin 1 */
        INFREE,          /* Port C Pin 2 */
        INFREE,          /* Port C Pin 3 */
        INFREE,          /* Port C Pin 4 */
        INFREE,          /* Port C Pin 5 */
        INFREE,          /* Port C Pin 6 */
        INFREE,          /* Port C Pin 7 */
        INPULL,          /* Port D Pin 0 */
        INFREE,          /* Port D Pin 1 */
        INPULL,         /* Port D Pin 2 /INT0*/
        INFREE,         /* Port D Pin 3 / INT1 */
        INFREE,          /* Port D Pin 4  OC1B*/
        INFREE,          /* Port D Pin 5 OC1A*/
        INFREE,          /* Port D Pin 6 /   ICP*/
        INFREE           /* Port D Pin 7 */
```

Linking configurations

1. Push Button

```
typedef enum en_ButtonError_t
{           BUTTON_OK,
            BUTTON_ERROR,
            WRONG_BUTTON_PIN
}en_buttonError_t;
#define  button0 PIND0
#define  button1 PIND1
```

2. DIO

```
typedef enum{
            PA=0,
            PB,
            PC,
            PD
}DIO_Port_type;

typedef enum{
            OUTPUT,
            INFREE,
            INPULL
}DIO_PinStatus_type;

typedef enum{
            LOW=0,
            HIGH,
}DIO_PinVoltage_type;

typedef enum dioError{
            DIO_OK,
            WRONG_PORT_NUMBER,
            WRONG_PIN_NUMBER,
            WRONG_VALUE,
            WRONG_DIRECTION
}en_dioError_t;

typedef enum{
            PINA0=0,
            PINA1=1,
            PINA2,
            PINA3,
            PINA4,
            PINA5,
	PINA6,
	PINA7,
	PINB0,
	PINB1,
	PINB2,
	PINB3,
	PINB4,
	PINB5,
	PINB6,
	PINB7,
	PINC0,
	PINC1,
	PINC2,
	PINC3,
	PINC4,
	PINC5,
	PINC6,
	PINC7,
	PIND0,
	PIND1,
	PIND2,
	PIND3,
	PIND4,
	PIND5,
	PIND6,
	PIND7,
	TOTAL_PINS
}DIO_Pin_type;
```

3. External Interrupt

```
typedef enum{
	LOW_LEVEL=0,
	ANY_LOGIC_CHANGE,
	FALLING_EDGE,
	RISING_EDGE,
}TriggerEdge_type;

typedef enum{
	EX_INT0=0,
	EX_INT1,
	EX_INT2
}ExInterruptSource_type;

typedef enum extintError{
	EXTINT_OK,
	WRONG_INT_NUMBER
}en_extintError_t;
```

4. Timer

```
#define TIMR0_MAX_VALUE                 256
#define TIMR1_MAX_VALUE                 256
#define TIMR2_MAX_VALUE                 256
extern   u8 car_mode ;
extern   s32 mode_ovf ;
extern   u8 g_speed_flag;

/********   TIMER0 [TCCR0] BITS   *******/
#define    CS00        0          // TIMER0 Prescaller Clock Select BIT 0
#define    CS01        1          // TIMER0 Prescaller Clock Select BIT 1
#define    CS02        2          // TIMER0 Prescaller Clock Select BIT 2
#define    WGM01       3           // Waveform Generation Mode (Normal,PWM-Phase
Correct,CTC0,Fast PWM).
#define    COM00       4          // Compare Match Output Mode BIT 0 (OC0) behavior
#define    COM01       5          // Compare Match Output Mode BIT 1 (OC0) behavior
#define    WGM00       6           // Waveform Generation Mode (Normal,PWM-Phase
Correct,CTC0,Fast PWM).
#define    FOC0        7          // Force Output Compare

/* TIFR */
#define OCF2    7
#define TOV2    6
#define ICF1    5
#define OCF1A   4
#define OCF1B   3
#define TOV1    2
#define OCF0    1
#define TOV0    0

/* TIMSK */
#define OCIE2   7
#define TOIE2   6
#define TICIE1  5
#define OCIE1A  4
#define OCIE1B  3
#define TOIE1   2
#define OCIE0   1
#define TOIE0   0
```

```c
typedef enum {
		INVALID_PRESCALER,
		INVALID_MODE,
		INVALID_OVF,
		INVALID_VALUE,
		TIMER_OK
} EN_timerError_t;

typedef enum{
		TIMER0_STOP=0,
		TIMER0_SCALER_1,
		TIMER0_SCALER_8,
		TIMER0_SCALER_64,
		TIMER0_SCALER_256,
		TIMER0_SCALER_1024,
		EXTERNALl_FALLING,
		EXTERNAL_RISING
}Timer0Scaler_type;

typedef enum{
		SCALER_1=1,
		SCALER_8=8,
		SCALER_64=64,
		SCALER_256=256,
		SCALER_1024=1024
}Scaler_type;

typedef enum
{
		TIMER0_NORMAL_MODE=0,
		TIMER0_PHASECORRECT_MODE,
		TIMER0_CTC_MODE,
		TIMER0_FASTPWM_MODE
}Timer0Mode_type;

typedef enum
{
		OC0_DISCONNECTED=0,
		OC0_TOGGLE,
		OC0_NON_INVERTING,
		OC0_INVERTING
}OC0Mode_type;

typedef enum{
		RISING,
		FALLING
}ICU_Edge_type;
```