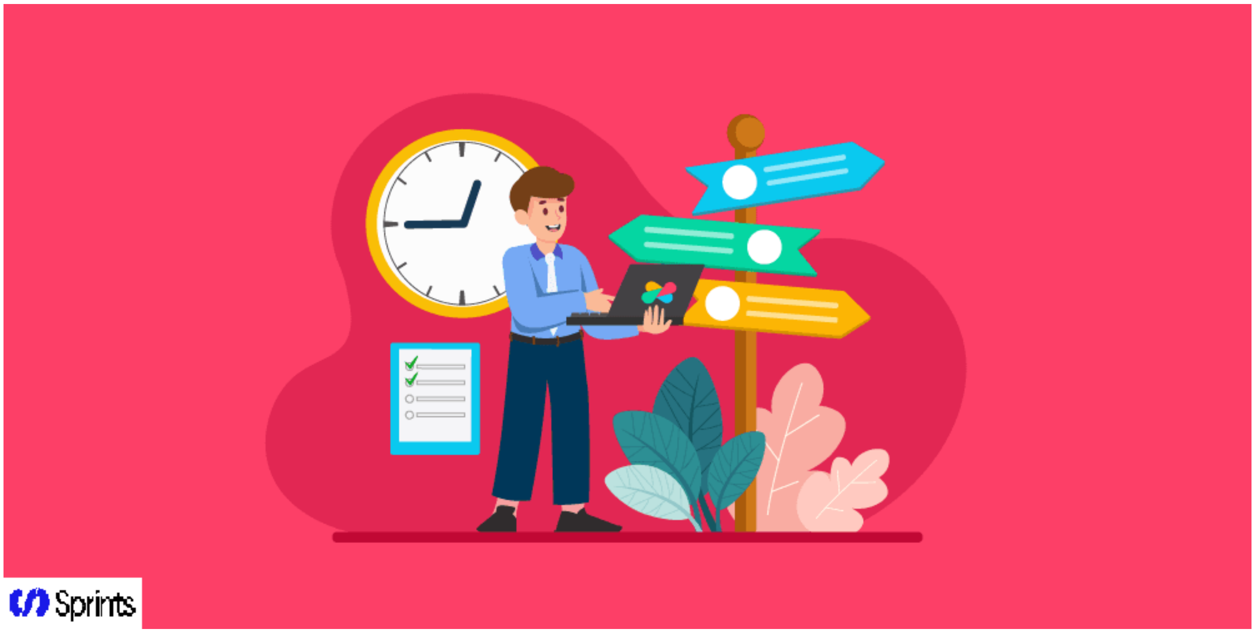


Design a Small OS

By : Mario Saad

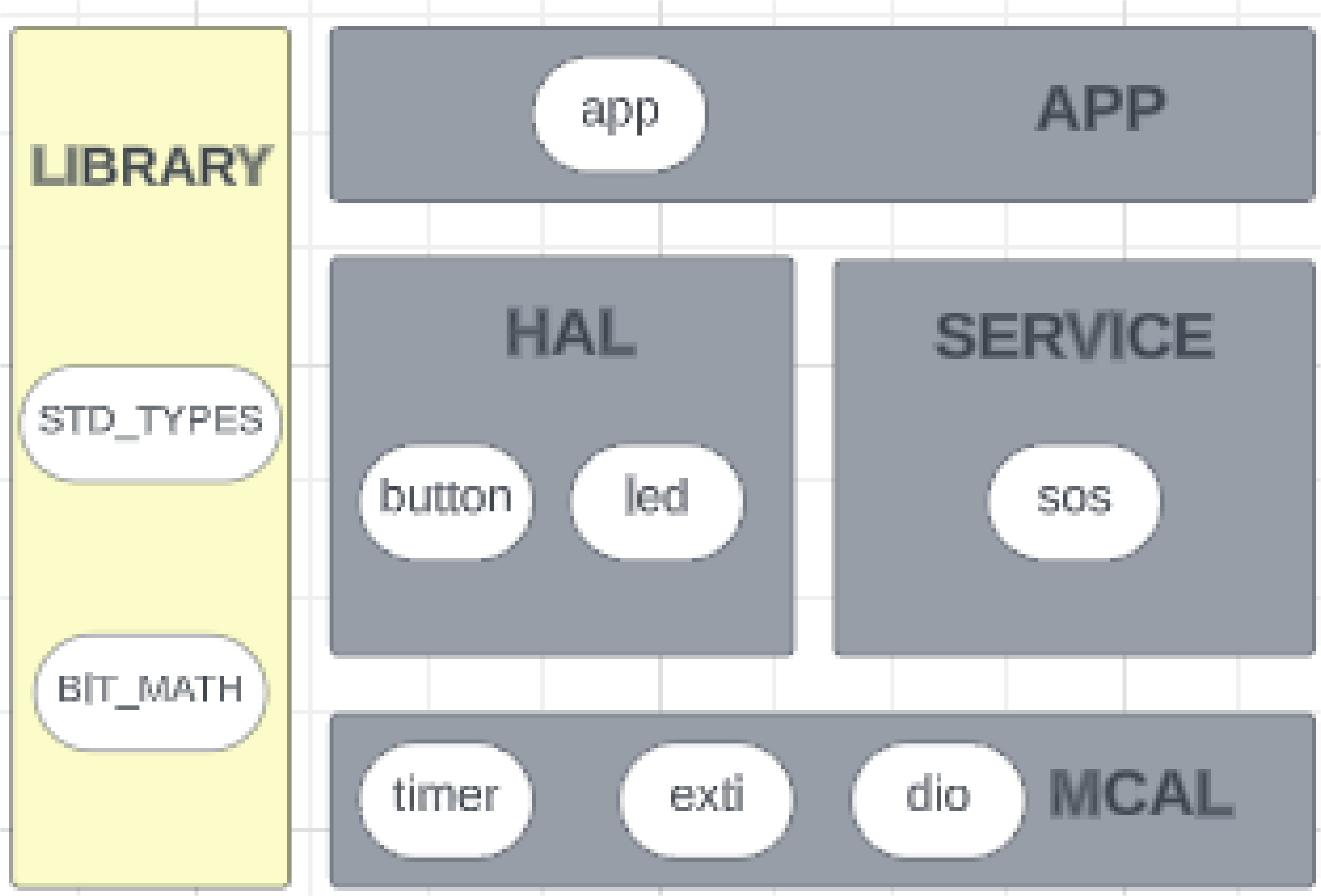
Table of content :

1. Introduction :

This project is a journey into the world of real-time operating systems, focusing on simplicity, efficiency, and flexibility. It is an open door for developers looking to create or adapt an RTOS for their specific applications while maintaining a sharp focus on resource optimization and real-time task management. SOS aims to empower embedded systems with the ability to execute tasks with precision and efficiency, setting the stage for a new generation of responsive and adaptable applications.

2. High-Level Design :

2.1. Layered architecture :



2.2. Modules Descriptions :

2.2.1. dio (Digital Input/Output):

- Description: The DIO component is responsible for controlling General-Purpose Input/Output (GPIO) pins. It provides functions and interfaces to set or read the state of these pins. It plays a critical role in interacting with external devices, sensors, or controlling peripherals.
- Usage: DIO can be used to configure and manipulate GPIO pins based on application requirements.

2.2.2. exti (External Interrupt):

- Description: The EXTI component is responsible for interfacing with external interrupts and events generated by external devices or sensors. It allows the system to respond to specific external events and trigger actions based on those events.
- Usage: EXTI control enables the system to handle external events such as button presses, sensor inputs, or other external triggers, making it a crucial component for system responsiveness and event-driven functionality.

2.2.3. timer:

- Description: The Timer component is essential for managing timing within the system. It controls the execution of tasks at specific intervals, enabling time-triggered functionality.
- Usage: The Timer component is employed to create precise timing for tasks and events, ensuring they occur at the desired intervals.

2.2.4. led :

- Description: The LED component handles the state of LEDs in the system. It provides functions to set LEDs to different states, such as ON or OFF, to convey information or status.
- Usage: LED control is utilized to visually represent system states or provide feedback to users.

2.2.5. button:

- Description: The Button component is responsible for interfacing with physical buttons or switches. It detects button presses and releases, allowing the system to respond to user input.
- Usage: Button control enables the system to start or stop specific functions, such as running or halting the operating system.

2.2.6. sos (Small Operating System):

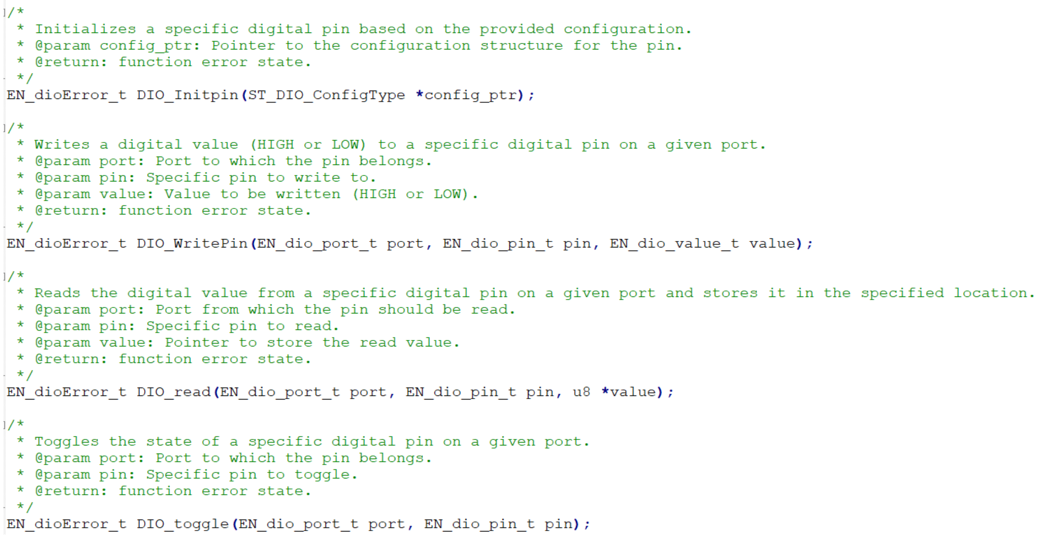
- Description: The SOS component is the heart of the system, functioning as a compact real-time operating system. It manages the execution of application processes, provides task scheduling, and ensures that tasks are executed in a priority-based, preemptive manner.
- Usage: SOS is the core of the system, orchestrating the execution of tasks and ensuring the efficient operation of the application.

2.2.7. app (Application):

- Description: The App component houses the main logic of the system. It defines how different components interact and orchestrates the flow of the application. It utilizes services provided by other components to achieve the system's overall functionality.
- Usage: The App component is where the unique logic of the application is implemented, making use of the capabilities provided by DIO, Timer, LED, Button, and SOS to achieve the system's goals.

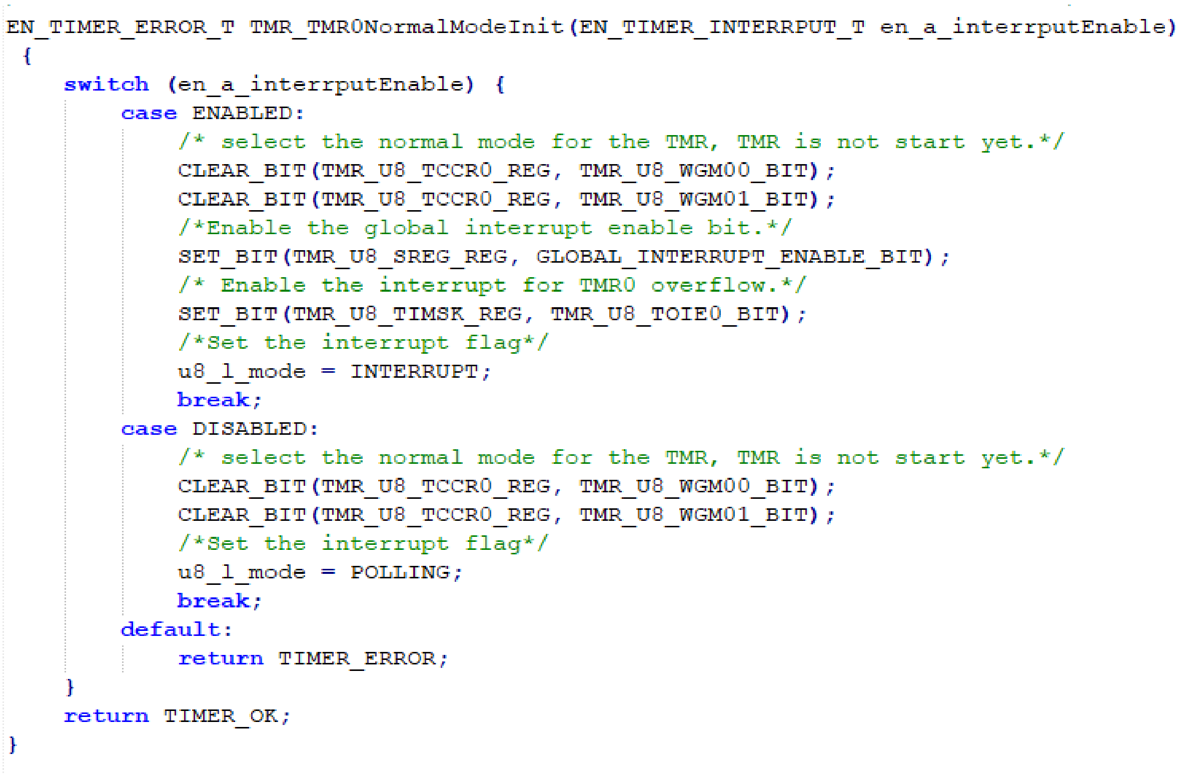
2.3. Drivers' documentation :

2.3.1. dio :

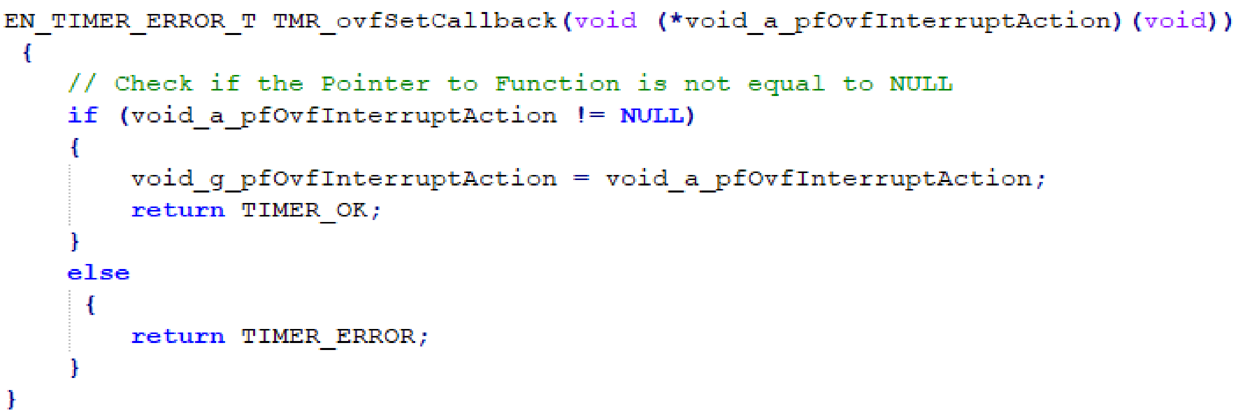


```
/*
 * Initializes a specific digital pin based on the provided configuration.
 * @param config_ptr: Pointer to the configuration structure for the pin.
 * @return: function error state.
 */
EN_dioError_t DIO_Initpin(ST_DIO_ConfigType *config_ptr);

/*
 * Writes a digital value (HIGH or LOW) to a specific digital pin on a given port.
 * @param port: Port to which the pin belongs.
 * @param pin: Specific pin to write to.
 * @param value: Value to be written (HIGH or LOW).
 * @return: function error state.
 */
EN_dioError_t DIO_WritePin(EN_dio_port_t port, EN_dio_pin_t pin, EN_dio_value_t value);

/*
 * Reads the digital value from a specific digital pin on a given port and stores it in the specified location.
 * @param port: Port from which the pin should be read.
 * @param pin: Specific pin to read.
 * @param value: Pointer to store the read value.
 * @return: function error state.
 */
EN_dioError_t DIO_read(EN_dio_port_t port, EN_dio_pin_t pin, u8 *value);

/*
 * Toggles the state of a specific digital pin on a given port.
 * @param port: Port to which the pin belongs.
 * @param pin: Specific pin to toggle.
 * @return: function error state.
 */
EN_dioError_t DIO_toggle(EN_dio_port_t port, EN_dio_pin_t pin);
```

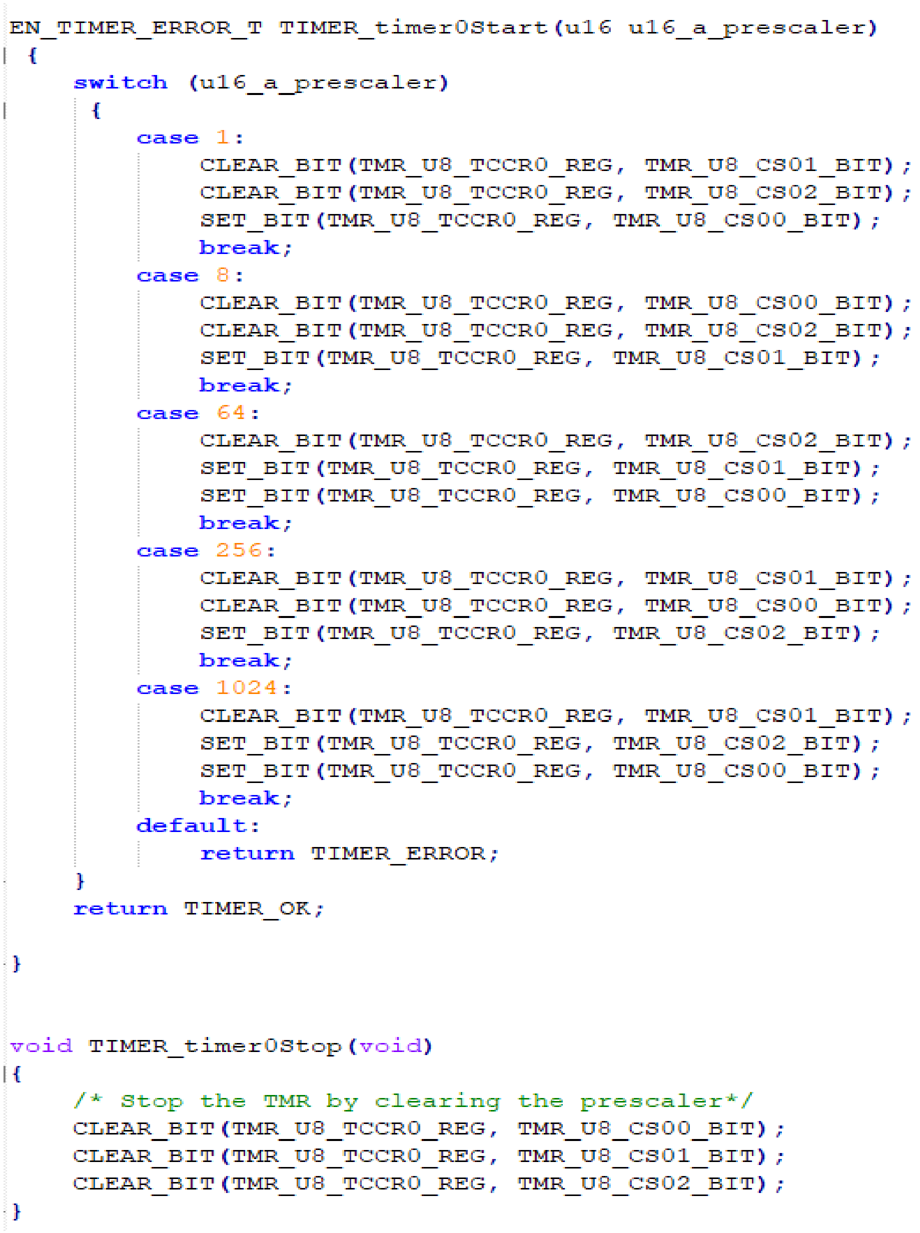
2.3.2. timer :



```
EN_TIMER_ERROR_T TMR_TMR0NormalModeInit(EN_TIMER_INTERRPUT_T en_a_interrputEnable)
 {
    switch (en_a_interrputEnable) {
        case ENABLED:
            /* select the normal mode for the TMR, TMR is not start yet.*/
            CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_WGM00_BIT);
            CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_WGM01_BIT);
            /*Enable the global interrupt enable bit.*/
            SET_BIT(TMR_U8_SREG_REG, GLOBAL_INTERRUPT_ENABLE_BIT);
            /* Enable the interrupt for TMR0 overflow.*/
            SET_BIT(TMR_U8_TIMSK_REG, TMR_U8_TOIE0_BIT);
            /*Set the interrupt flag*/
            u8_l_mode = INTERRUPT;
            break;
        case DISABLED:
            /* select the normal mode for the TMR, TMR is not start yet.*/
            CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_WGM00_BIT);
            CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_WGM01_BIT);
            /*Set the interrupt flag*/
            u8_l_mode = POLLING;
            break;
        default:
            return TIMER_ERROR;
    }
    return TIMER_OK;
}
```

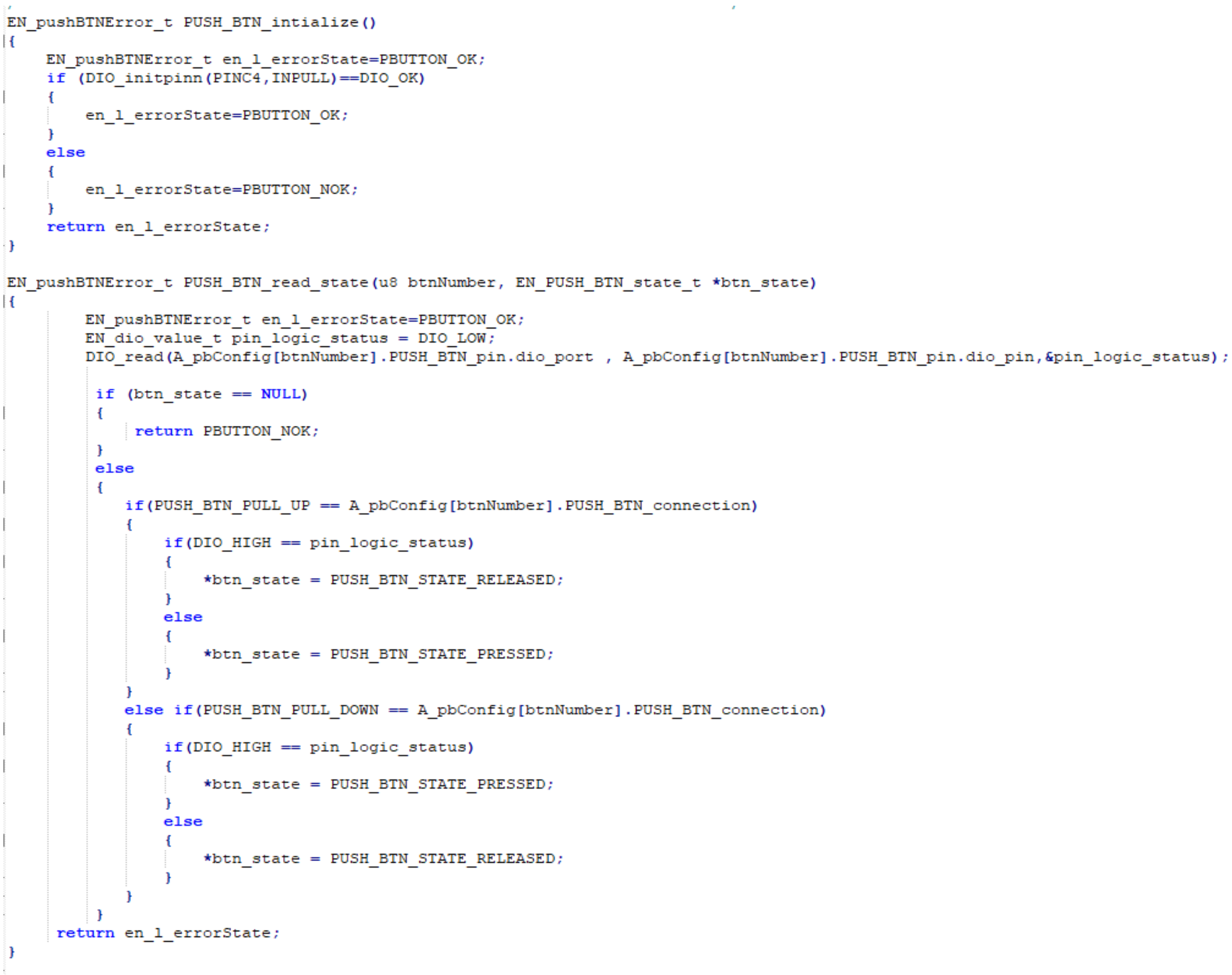


```
EN_TIMER_ERROR_T TMR_ovfSetCallback(void (*void_a_pfOvfInterruptAction)(void))
 {
    // Check if the Pointer to Function is not equal to NULL
    if (void_a_pfOvfInterruptAction != NULL)
    {
        void_g_pfOvfInterruptAction = void_a_pfOvfInterruptAction;
        return TIMER_OK;
    }
    else
     {
        return TIMER_ERROR;
    }
}
```



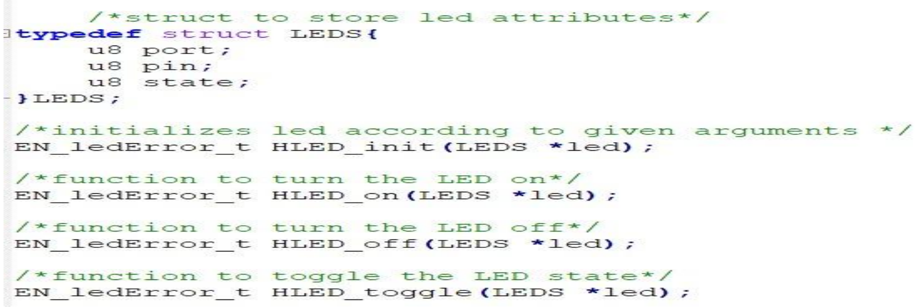
```
EN_TIMER_ERROR_T TIMER_timer0Start(u16 u16_a_prescaler)
 {
     switch (u16_a_prescaler)
      {
         case 1:
             CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS01_BIT);
             CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS02_BIT);
             SET_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS00_BIT);
             break;
         case 8:
             CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS00_BIT);
             CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS02_BIT);
             SET_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS01_BIT);
             break;
         case 64:
             CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS02_BIT);
             SET_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS01_BIT);
             SET_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS00_BIT);
             break;
         case 256:
             CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS01_BIT);
             CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS00_BIT);
             SET_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS02_BIT);
             break;
         case 1024:
             CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS01_BIT);
             SET_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS02_BIT);
             SET_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS00_BIT);
             break;
         default:
             return TIMER_ERROR;
      }
     return TIMER_OK;

}


void TIMER_timer0Stop(void)
{
     /* Stop the TMR by clearing the prescaler*/
     CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS00_BIT);
     CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS01_BIT);
     CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS02_BIT);
}
```

2.3.3. button :



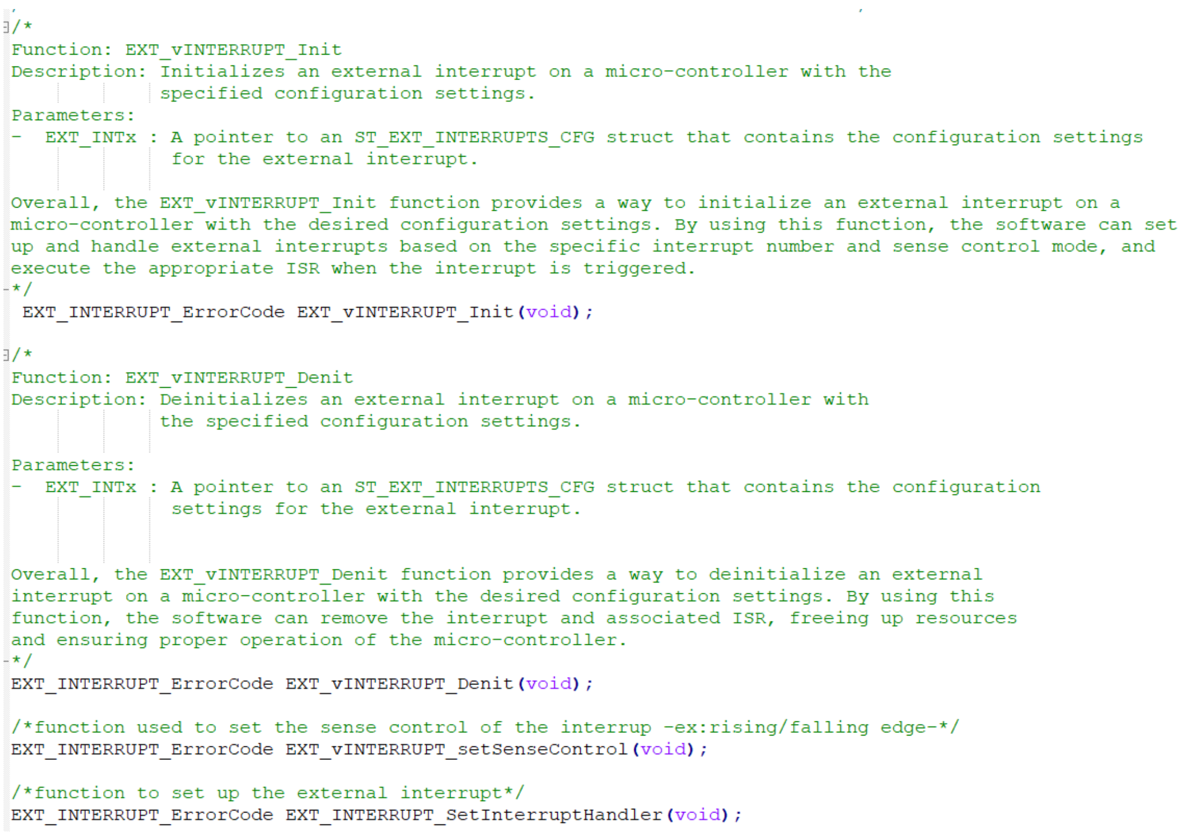
```
EN_pushBTNError_t PUSH_BTN_intialize()
{
    EN_pushBTNError_t en_l_errorState=PBUTTON_OK;
    if (DIO_initpinn(PINC4,INPULL)==DIO_OK)
    {
        en_l_errorState=PBUTTON_OK;
    }
    else
    {
        en_l_errorState=PBUTTON_NOK;
    }
    return en_l_errorState;
}

EN_pushBTNError_t PUSH_BTN_read_state(u8 btnNumber, EN_PUSH_BTN_state_t *btn_state)
{
        EN_pushBTNError_t en_l_errorState=PBUTTON_OK;
        EN_dio_value_t pin_logic_status = DIO_LOW;
        DIO_read(A_pbConfig[btnNumber].PUSH_BTN_pin.dio_port , A_pbConfig[btnNumber].PUSH_BTN_pin.dio_pin,&pin_logic_status);

         if (btn_state == NULL)
         {
             return PBUTTON_NOK;
         }
         else
         {
            if(PUSH_BTN_PULL_UP == A_pbConfig[btnNumber].PUSH_BTN_connection)
            {
                if(DIO_HIGH == pin_logic_status)
                {
                    *btn_state = PUSH_BTN_STATE_RELEASED;
                }
                else
                {
                    *btn_state = PUSH_BTN_STATE_PRESSED;
                }
            }
            else if(PUSH_BTN_PULL_DOWN == A_pbConfig[btnNumber].PUSH_BTN_connection)
            {
                if(DIO_HIGH == pin_logic_status)
                {
                    *btn_state = PUSH_BTN_STATE_PRESSED;
                }
                else
                {
                    *btn_state = PUSH_BTN_STATE_RELEASED;
                }
            }
         }
     return en_l_errorState;
}
```

2.3.4. led:



```
    /*struct to store led attributes*/
typedef struct LEDS{
    u8 port;
    u8 pin;
    u8 state;
}LEDS;

/*initializes led according to given arguments */
EN_ledError_t HLED_init(LEDS *led);

/*function to turn the LED on*/
EN_ledError_t HLED_on(LEDS *led);

/*function to turn the LED off*/
EN_ledError_t HLED_off(LEDS *led);

/*function to toggle the LED state*/
EN_ledError_t HLED_toggle(LEDS *led);
```

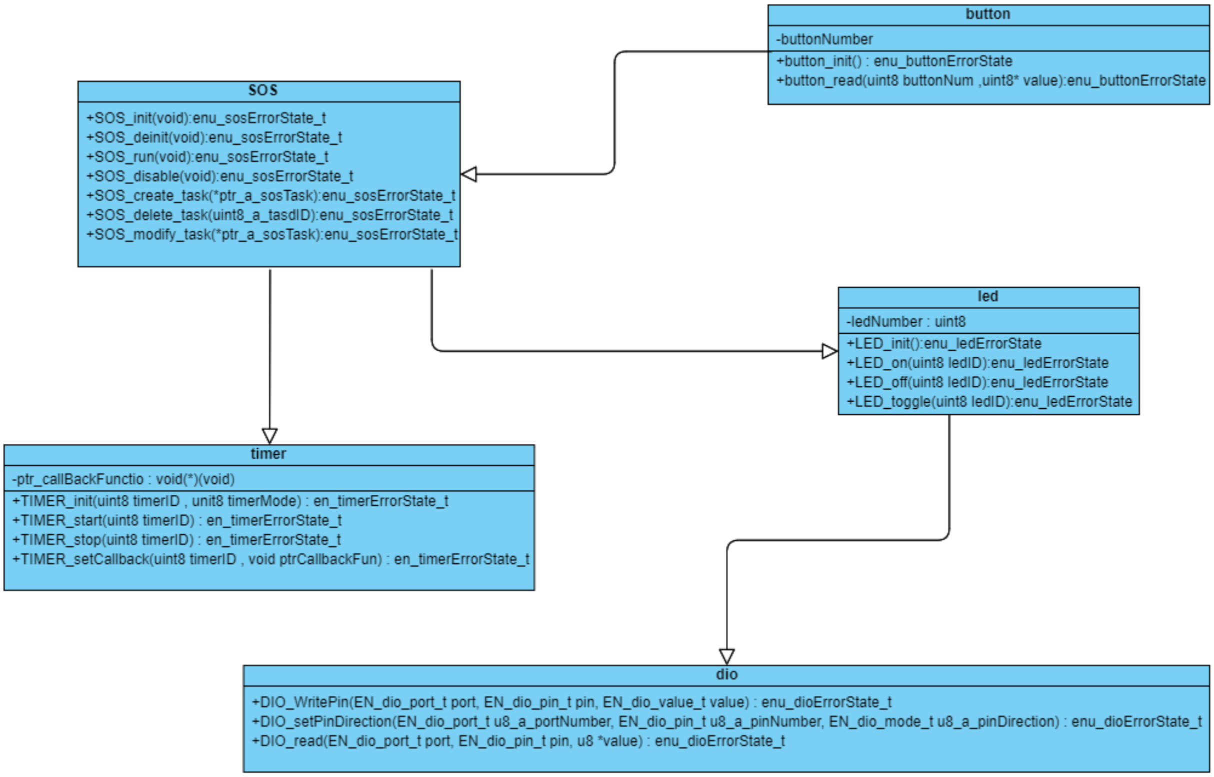
2.3.5 External Interrupt:



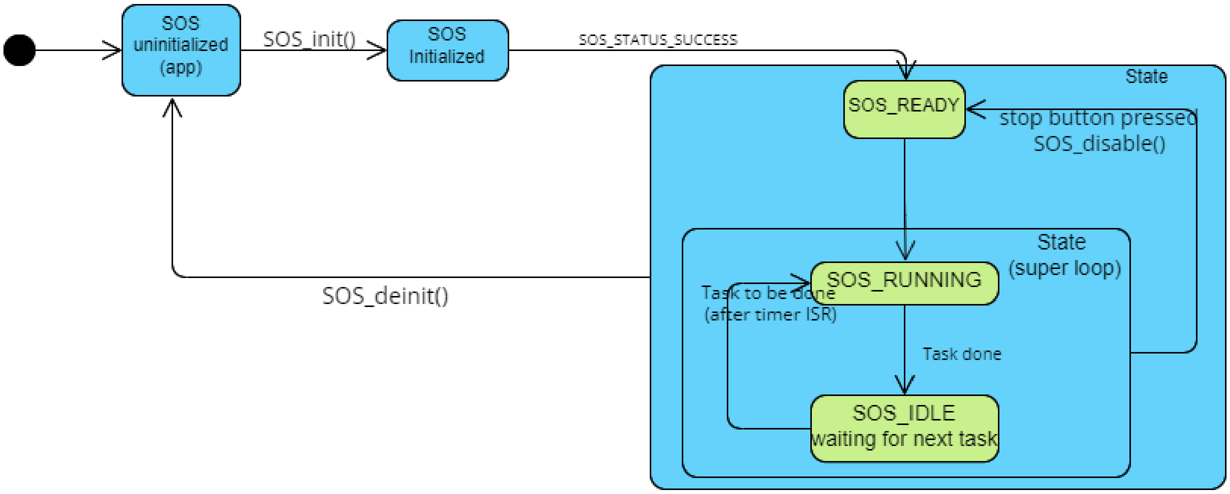
```
/*
Function: EXT_vINTERRUPT_Init
Description: Initializes an external interrupt on a micro-controller with the
             specified configuration settings.
Parameters:
-  EXT_INTx : A pointer to an ST_EXT_INTERRUPTS_CFG struct that contains the configuration settings
              for the external interrupt.

Overall, the EXT_vINTERRUPT_Init function provides a way to initialize an external interrupt on a
micro-controller with the desired configuration settings. By using this function, the software can set
up and handle external interrupts based on the specific interrupt number and sense control mode, and
execute the appropriate ISR when the interrupt is triggered.
*/
 EXT_INTERRUPT_ErrorCode EXT_vINTERRUPT_Init(void);

/*
Function: EXT_vINTERRUPT_Denit
Description: Deinitializes an external interrupt on a micro-controller with
             the specified configuration settings.

Parameters:
-  EXT_INTx : A pointer to an ST_EXT_INTERRUPTS_CFG struct that contains the configuration
              settings for the external interrupt.


Overall, the EXT_vINTERRUPT_Denit function provides a way to deinitialize an external
interrupt on a micro-controller with the desired configuration settings. By using this
function, the software can remove the interrupt and associated ISR, freeing up resources
and ensuring proper operation of the micro-controller.
*/
EXT_INTERRUPT_ErrorCode EXT_vINTERRUPT_Denit(void);

/*function used to set the sense control of the interrup -ex:rising/falling edge-*/
EXT_INTERRUPT_ErrorCode EXT_vINTERRUPT_setSenseControl(void);

/*function to set up the external interrupt*/
EXT_INTERRUPT_ErrorCode EXT_INTERRUPT_SetInterruptHandler(void);
```

2.4. UML :

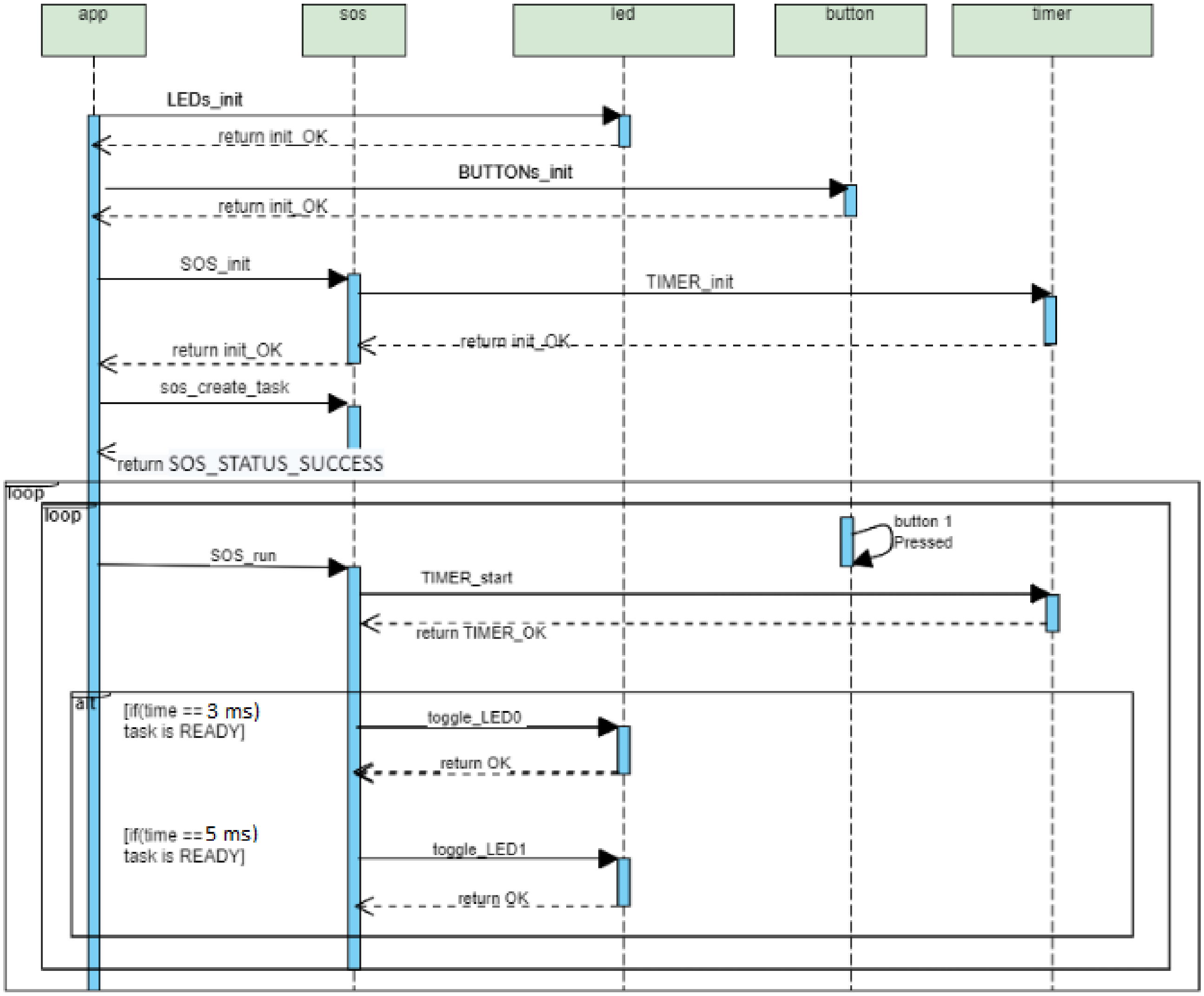
2.4.1 Class diagram :


button
-buttonNumber
+button_init() : enu_buttonErrorState
+button_read(uint8 buttonNum ,uint8* value):enu_buttonErrorState
SOS
+SOS_init(void):enu_sosErrorState_t
+SOS_deinit(void):enu_sosErrorState_t
+SOS_run(void):enu_sosErrorState_t
+SOS_disable(void):enu_sosErrorState_t
+SOS_create_task(*ptr_a_sosTask):enu_sosErrorState_t
+SOS_delete_task(uint8_a_tasdID):enu_sosErrorState_t
+SOS_modify_task(*ptr_a_sosTask):enu_sosErrorState_t
led
-ledNumber : uint8
+LED_init():enu_ledErrorState
+LED_on(uint8 ledID):enu_ledErrorState
+LED_off(uint8 ledID):enu_ledErrorState
+LED_toggle(uint8 ledID):enu_ledErrorState
timer
-ptr_callBackFunctio : void(*)(void)
+TIMER_init(uint8 timerID , unit8 timerMode) : en_timerErrorState_t
+TIMER_start(uint8 timerID) : en_timerErrorState_t
+TIMER_stop(uint8 timerID) : en_timerErrorState_t
+TIMER_setCallback(uint8 timerID , void ptrCallbackFun) : en_timerErrorState_t
dio
+DIO_WritePin(EN_dio_port_t port, EN_dio_pin_t pin, EN_dio_value_t value) : enu_dioErrorState_t
+DIO_setPinDirection(EN_dio_port_t u8_a_portNumber, EN_dio_pin_t u8_a_pinNumber, EN_dio_mode_t u8_a_pinDirection) : enu_dioErrorState_t
+DIO_read(EN_dio_port_t port, EN_dio_pin_t pin, u8 *value) : enu_dioErrorState_t

2.4.2. State machine :



2.5. Sequence diagram :

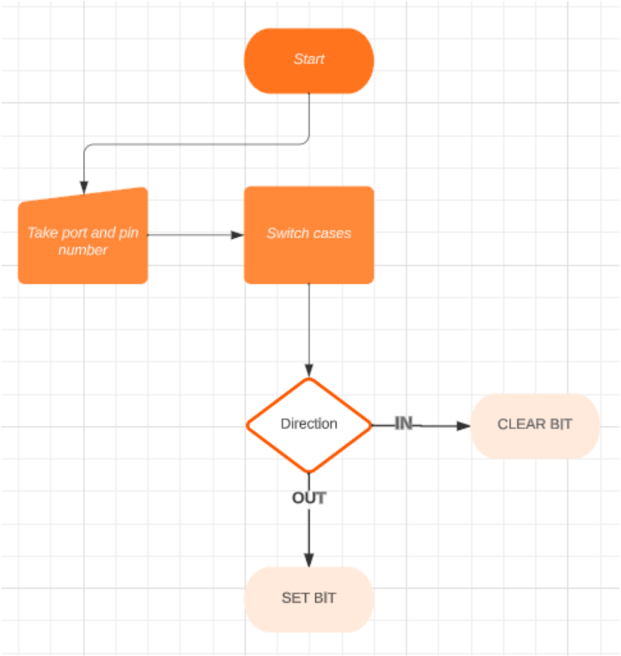


3. Low-Level Design :

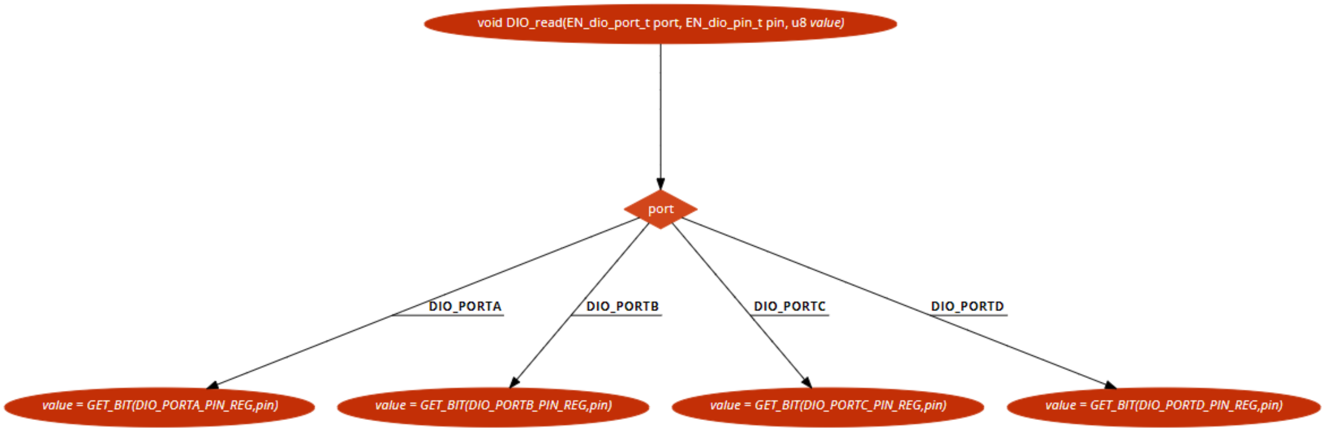
3.1Flowcharts:

3.1.1. dio :

EN_dioError_t DIO_WritePin(EN_dio_port_t port, EN_dio_pin_t pin, EN_dio_value_t value)

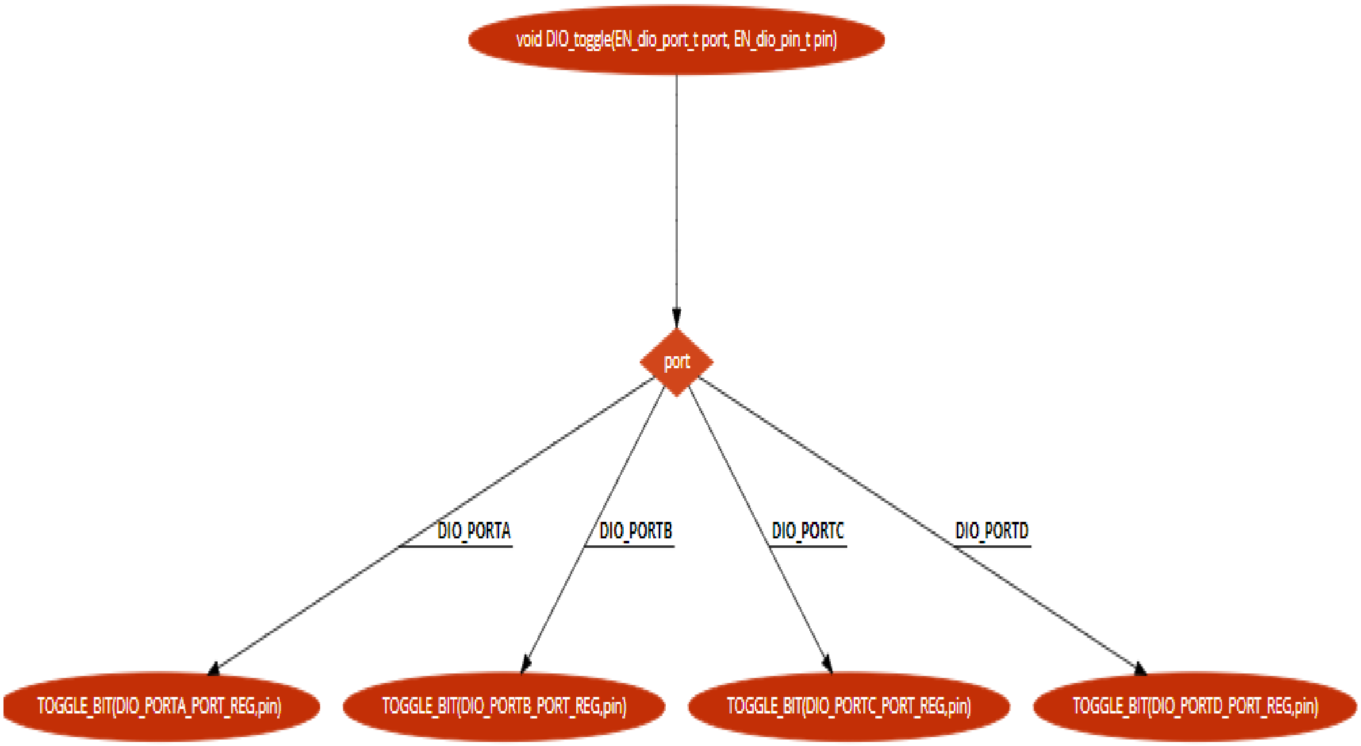


EN_dioError_t DIO_read(EN_dio_port_t port, EN_dio_pin_t pin, u8 *value)



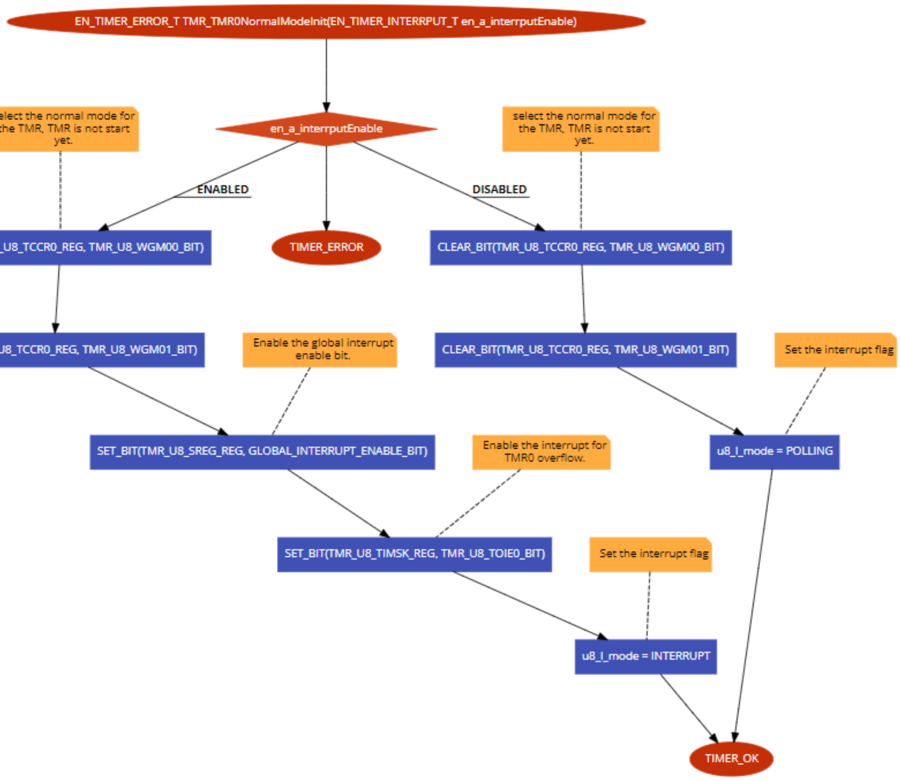
void DIO_read(EN_dio_port_t port, EN_dio_pin_t pin, u8 value)
port
DIO_PORTA
DIO_PORTB
DIO_PORTC
DIO_PORTD
value = GET_BIT(DIO_PORTA_PIN_REG,pin)
value = GET_BIT(DIO_PORTB_PIN_REG,pin)
value = GET_BIT(DIO_PORTC_PIN_REG,pin)
value = GET_BIT(DIO_PORTD_PIN_REG,pin)

EN_dioError_t DIO_toggle(EN_dio_port_t port, EN_dio_pin_t pin)

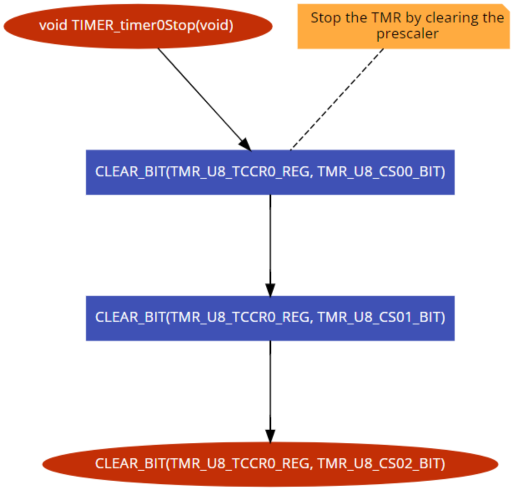

void DIO_toggle(EN_dio_port_t port, EN_dio_pin_t pin)
port
DIO_PORTA
DIO_PORTB
DIO_PORTC
DIO_PORTD
TOGGLE_BIT(DIO_PORTA_PORT_REG,pin)
TOGGLE_BIT(DIO_PORTB_PORT_REG,pin)
TOGGLE_BIT(DIO_PORTC_PORT_REG,pin)
TOGGLE_BIT(DIO_PORTD_PORT_REG,pin)

3.1.2. Timer :

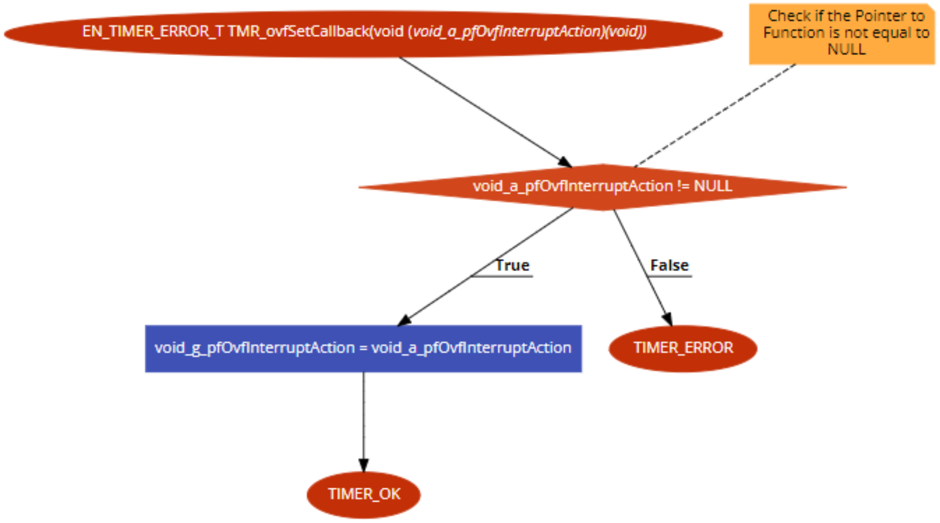
EN_TIMER_ERROR_T TMR_TMR0NormalModeInit(EN_TIMER_INTERRPUT_T en_a_interrputEnable)


EN_TIMER_ERROR_T TMR_TMR0NormalModeInit(EN_TIMER_INTERRPUT_T en_a_interrputEnable)
en_a_interrputEnable
select the normal mode for the TMR, TMR is not start yet.
select the normal mode for the TMR, TMR is not start yet.
ENABLED
DISABLED
CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_WGM00_BIT)
TIMER_ERROR
CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_WGM00_BIT)
CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_WGM01_BIT)
Enable the global interrupt enable bit.
CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_WGM01_BIT)
Set the interrupt flag
SET_BIT(TMR_U8_SREG_REG, GLOBAL_INTERRUPT_ENABLE_BIT)
Enable the interrupt for TMR0 overflow.
u8_l_mode = POLLING
SET_BIT(TMR_U8_TIMSK_REG, TMR_U8_TOIE0_BIT)
Set the interrupt flag
u8_l_mode = INTERRUPT
TIMER_OK

EN_TIMER_ERROR_T TIMER_timer0Stop(void)

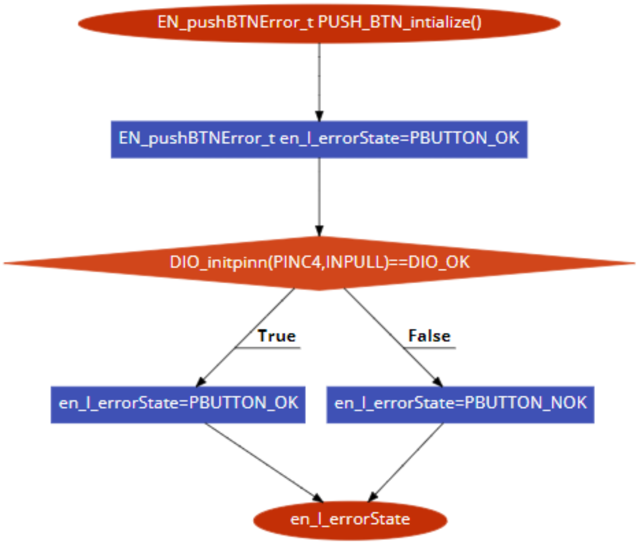

void TIMER_timer0Stop(void)
Stop the TMR by clearing the prescaler
CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS00_BIT)
CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS01_BIT)
CLEAR_BIT(TMR_U8_TCCR0_REG, TMR_U8_CS02_BIT)

EN_TIMER_ERROR_T TMR_ovfSetCallback(void (*void_a_pfOvfInterruptAction)(void))

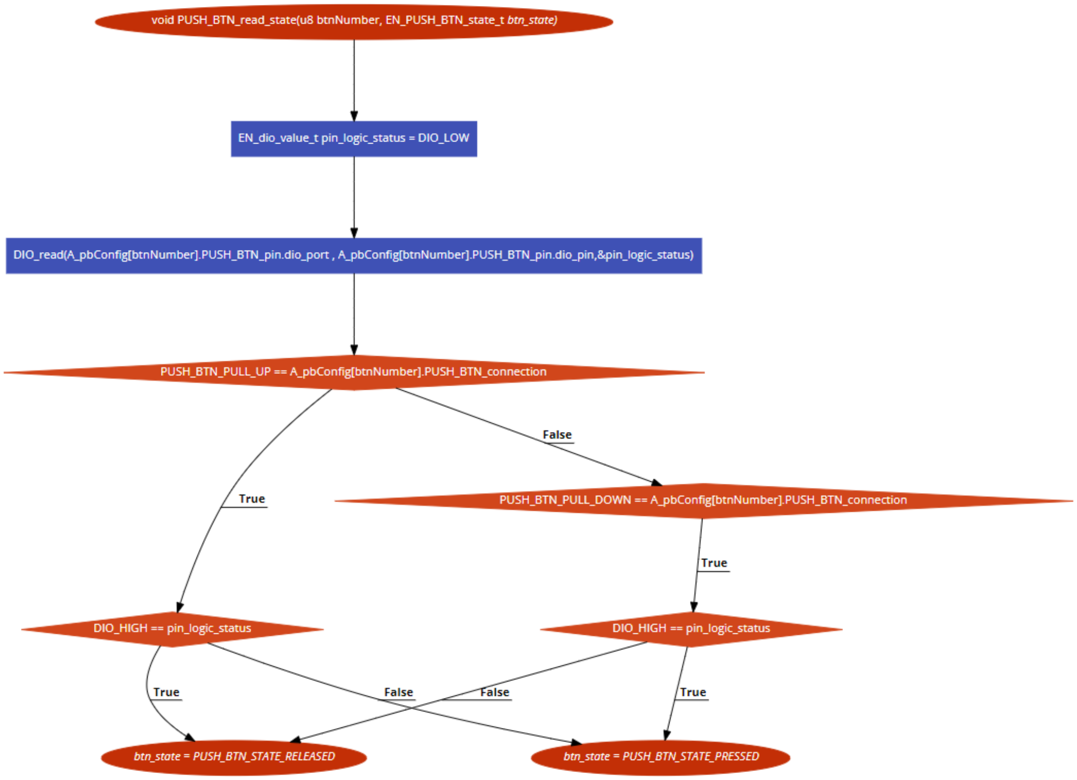

EN_TIMER_ERROR_T TMR_ovfSetCallback(void (*void_a_pfOvfInterruptAction)(void))
Check if the Pointer to Function is not equal to NULL
void_a_pfOvfInterruptAction != NULL
True
False
void_g_pfOvfInterruptAction = void_a_pfOvfInterruptAction
TIMER_ERROR
TIMER_OK

3.1.3. Push Button:

EN_pushBTNError_t PUSH_BTN_intialize()

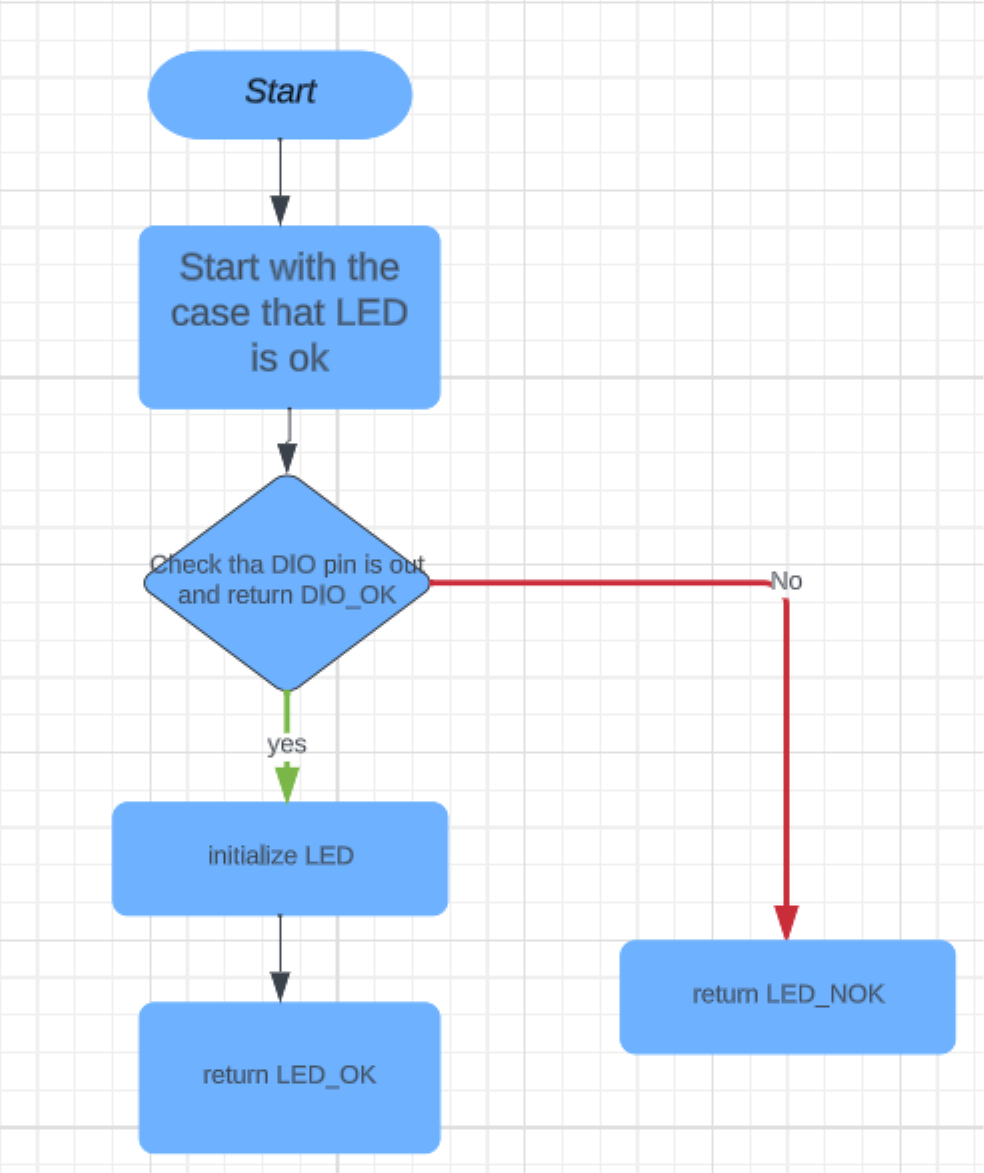

EN_pushBTNError_t PUSH_BTN_intialize()
EN_pushBTNError_t en_l_errorState=PBUTTON_OK
DIO_initpinn(PINC4,INPULL)==DIO_OK
True
False
en_l_errorState=PBUTTON_OK
en_l_errorState=PBUTTON_NOK
en_l_errorState

EN_pushBTNError_t PUSH_BTN_read_state(u8 btnNumber, EN_PUSH_BTN_state_t *btn_state)

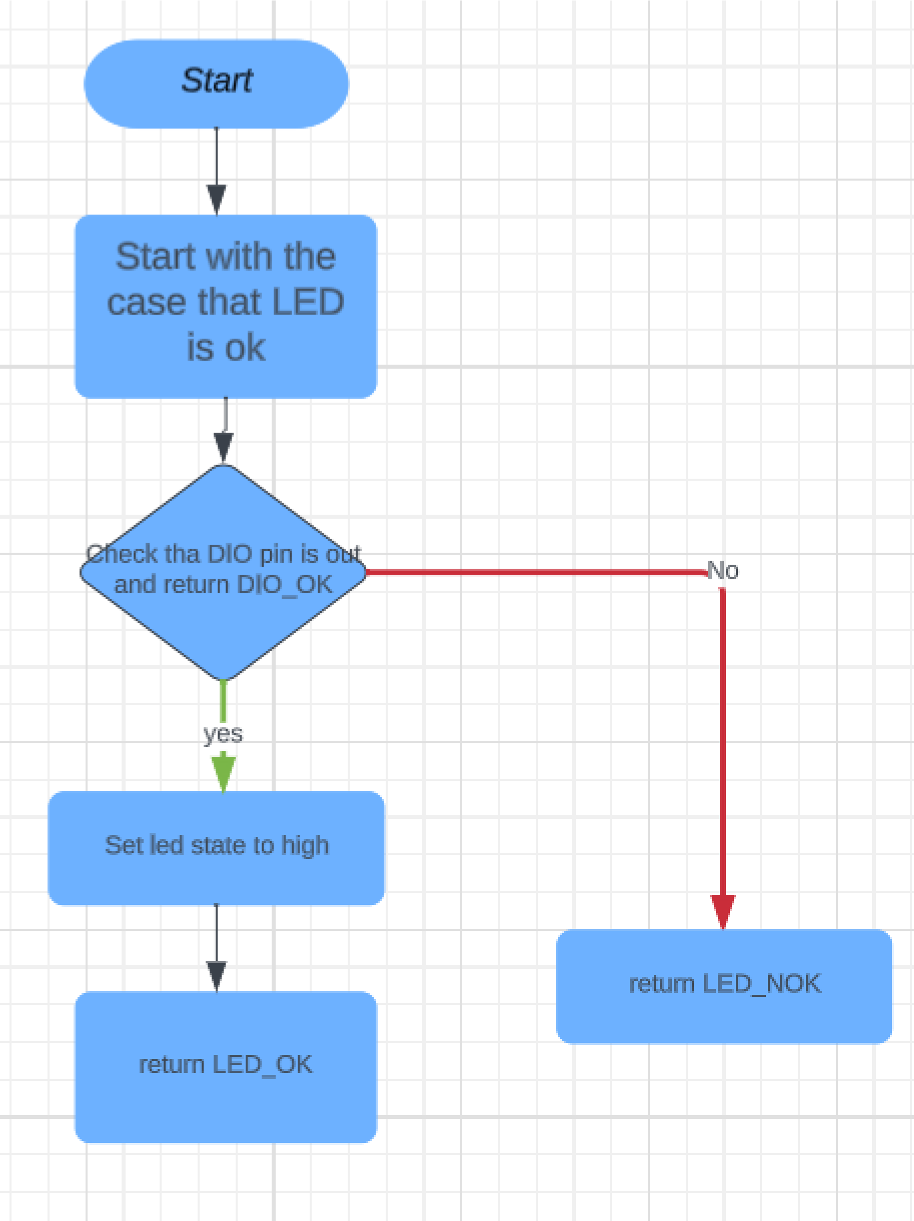

void PUSH_BTN_read_state(u8 btnNumber, EN_PUSH_BTN_state_t btn_state)
EN_dio_value_t pin_logic_status = DIO_LOW
DIO_read(A_pbConfig[btnNumber].PUSH_BTN_pin.dio_port , A_pbConfig[btnNumber].PUSH_BTN_pin.dio_pin,&pin_logic_status)
PUSH_BTN_PULL_UP == A_pbConfig[btnNumber].PUSH_BTN_connection
False
True
PUSH_BTN_PULL_DOWN == A_pbConfig[btnNumber].PUSH_BTN_connection
True
DIO_HIGH == pin_logic_status
DIO_HIGH == pin_logic_status
True
False
False
True
btn_state = PUSH_BTN_STATE_RELEASED
btn_state = PUSH_BTN_STATE_PRESSED

3.1.4. LED:

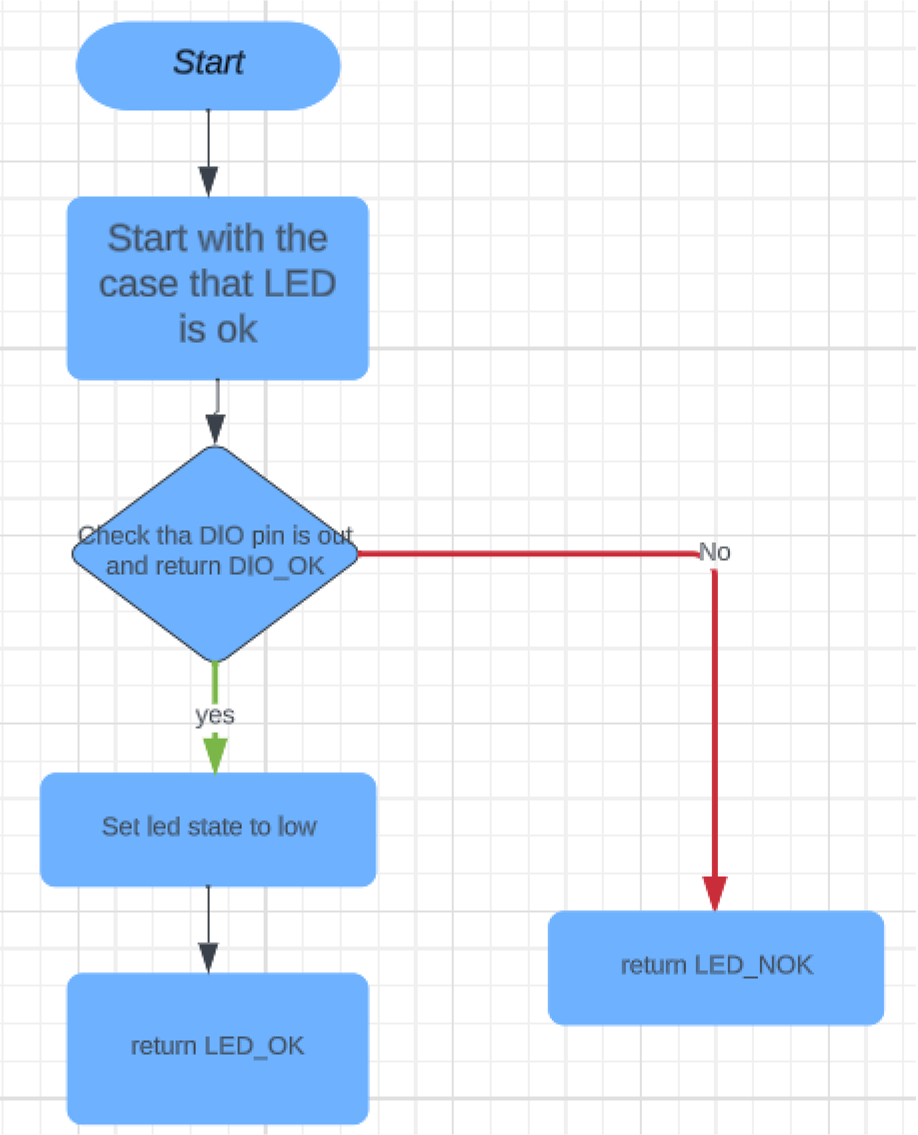
EN_ledError_t HLED_init(LEDS *led)


Start
Start with the case that LED is ok
Check tha DIO pin is out and return DIO_OK
No
yes
initialize LED
return LED_NOK
return LED_OK

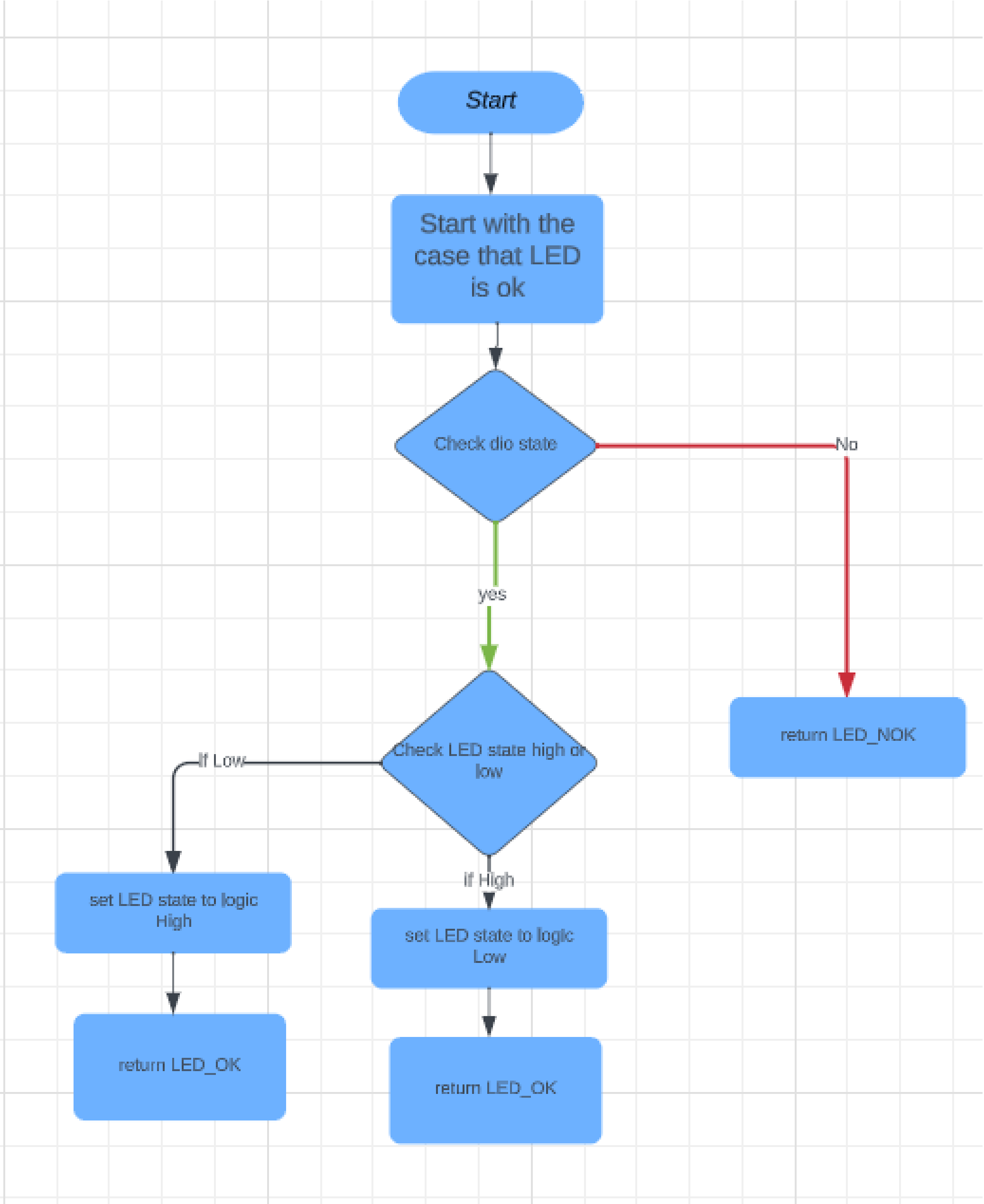
EN_ledError_t HLED_on(LEDS *led)


Start
Start with the case that LED is ok
Check tha DIO pin is out and return DIO_OK
No
yes
Set led state to high
return LED_NOK
return LED_OK

EN_ledError_t HLED_off(LEDS *led)

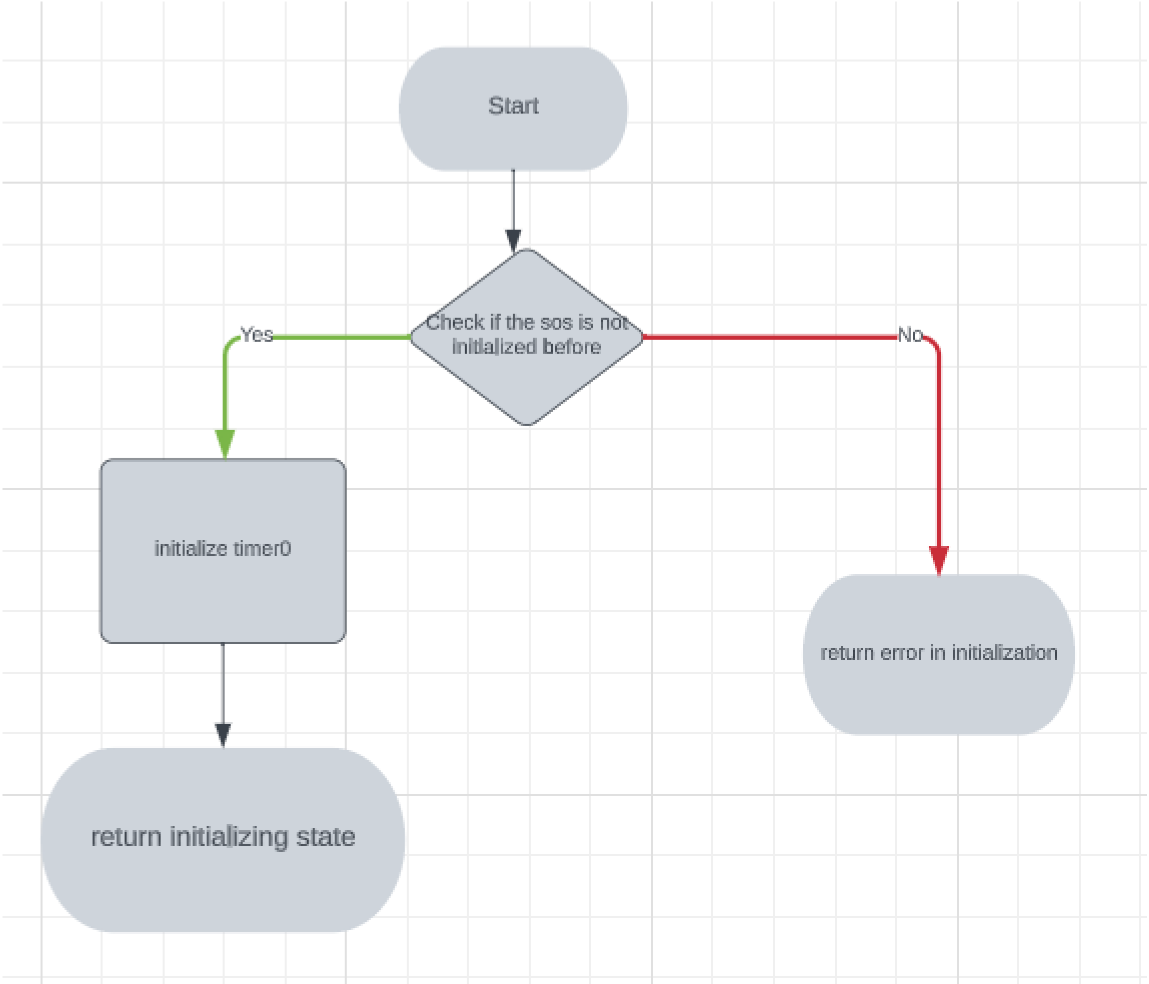

Start
Start with the case that LED is ok
Check tha DIO pin is out and return DIO_OK
No
yes
Set led state to low
return LED_NOK
return LED_OK

EN_ledError_t HLED_toggle(LEDS *led)

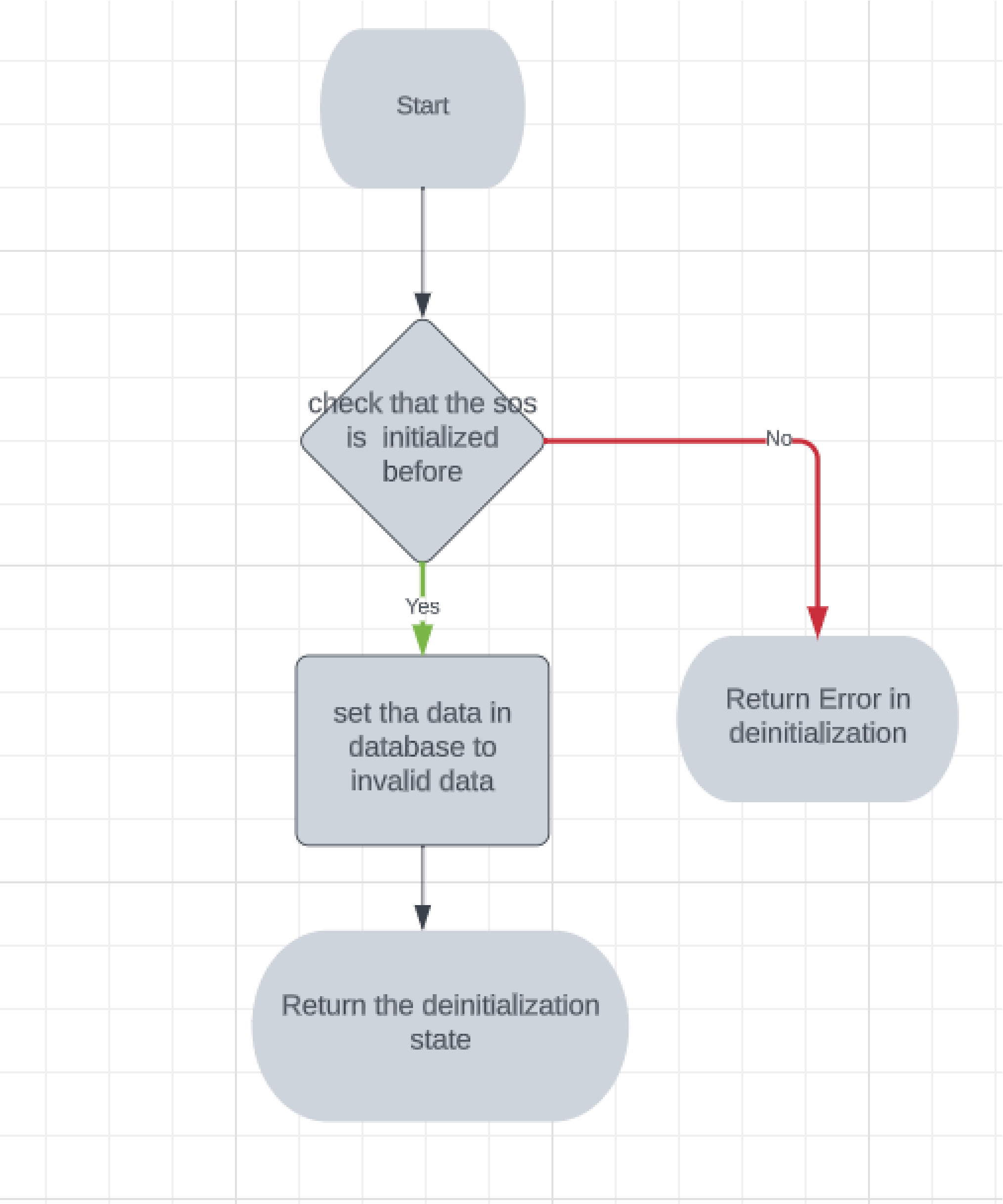

Start
Start with the case that LED is ok
Check dio state
No
yes
return LED_NOK
Check LED state high or low
If Low
If High
set LED state to logic High
set LED state to logic Low
return LED_OK
return LED_OK

3.1.5. SOS:

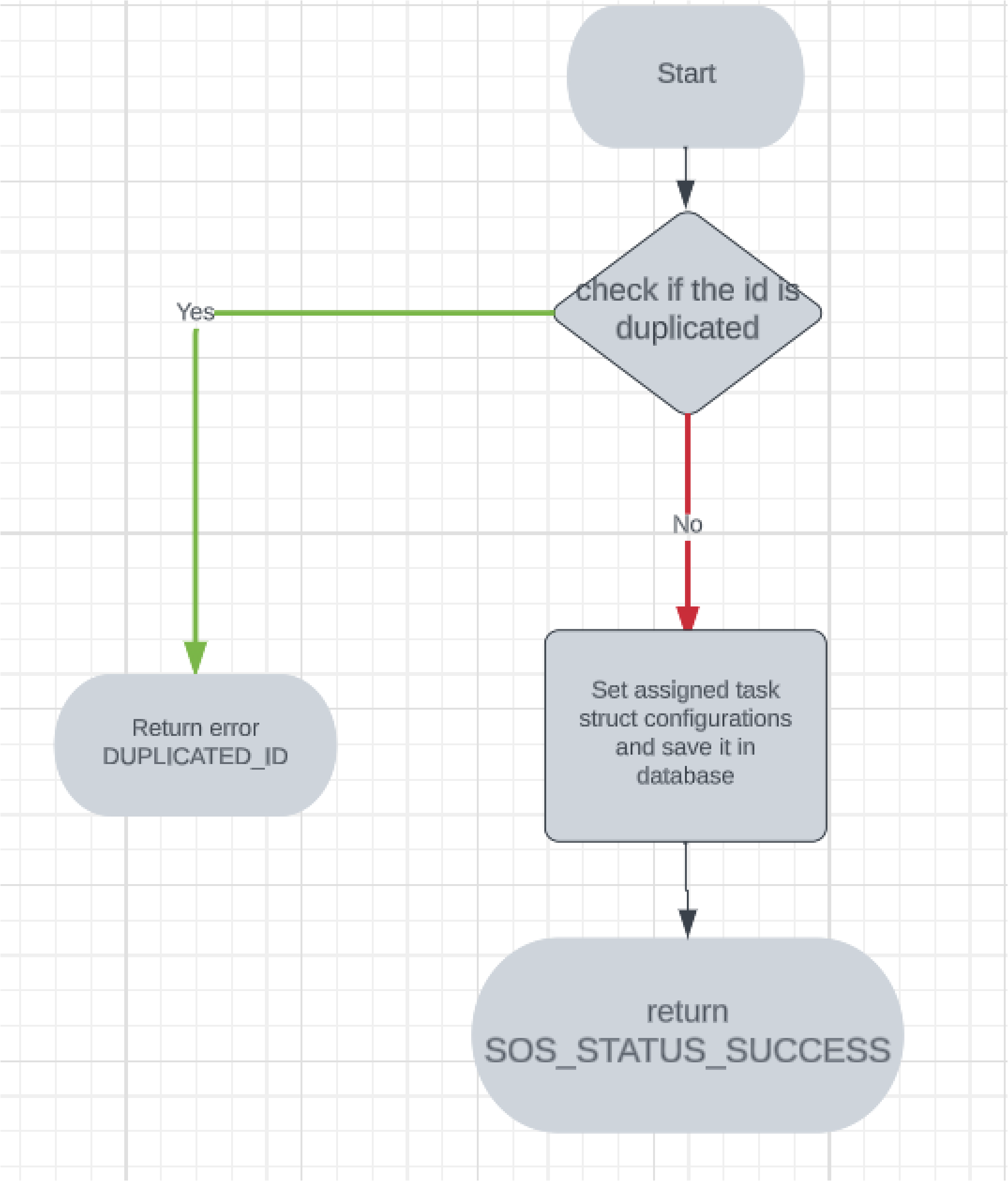
enu_sosErrorState_t sos_init (void);


Start
Check if the sos is not initialized before
Yes
No
initialize timer0
return error in initialization
return initializing state

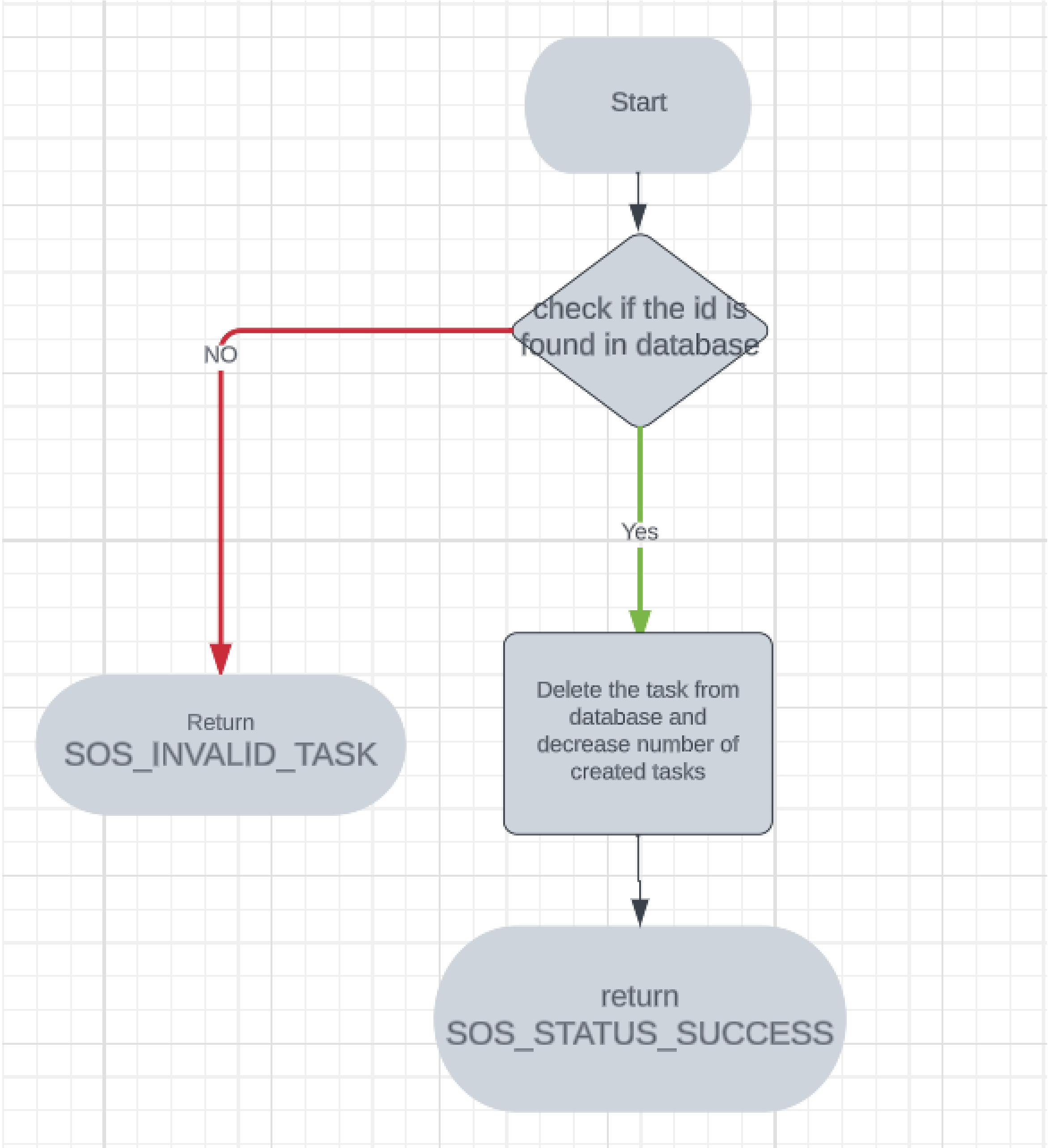
enu_sosErrorState_t sos_deinit (void);


Start
check that the sos is initialized before
No
Yes
set tha data in database to invalid data
Return Error in deinitialization
Return the deinitialization state

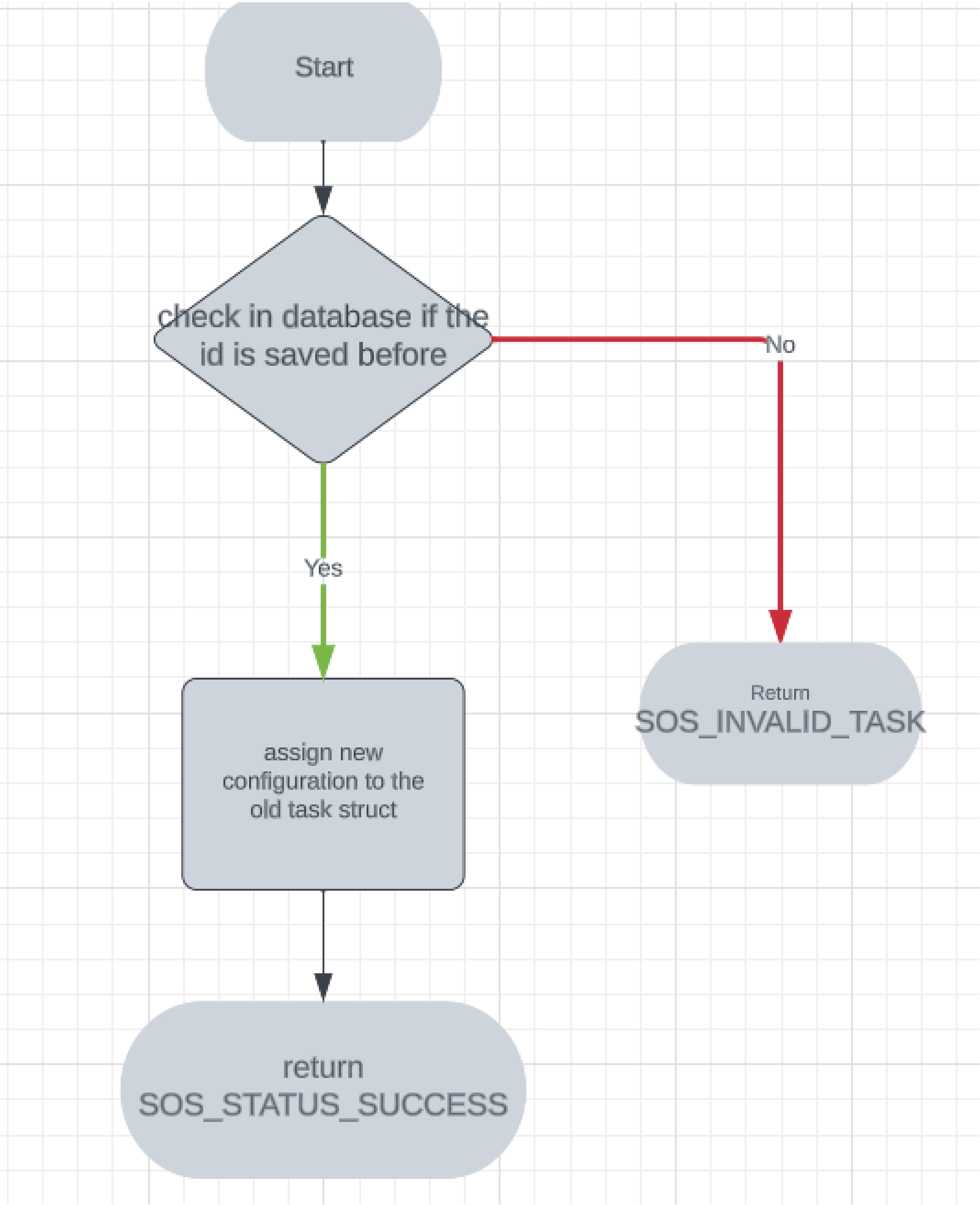
enu_sosErrorState_t sos_create_task(uint8_t task_id,str_sosTask_t *ptr_str_sosTask);


Start
check if the id is duplicated
Yes
No
Return error DUPLICATED_ID
Set assigned task struct configurations and save it in database
return SOS_STATUS_SUCCESS

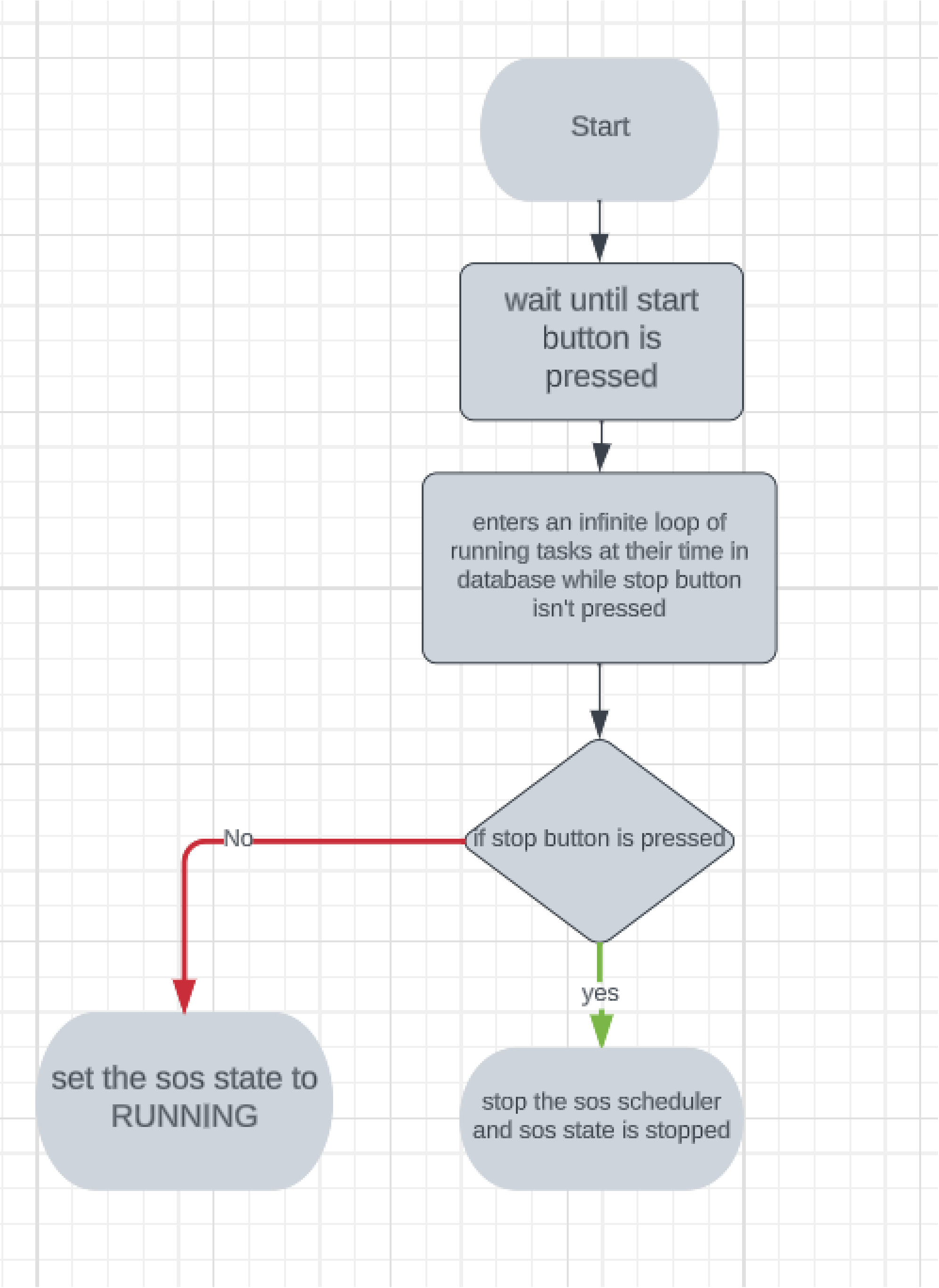
enu_sosErrorState_t sos_delete_task (uint8_t task_id);


Start
check if the id is found in database
NO
Yes
Return SOS_INVALID_TASK
Delete the task from database and decrease number of created tasks
return SOS_STATUS_SUCCESS

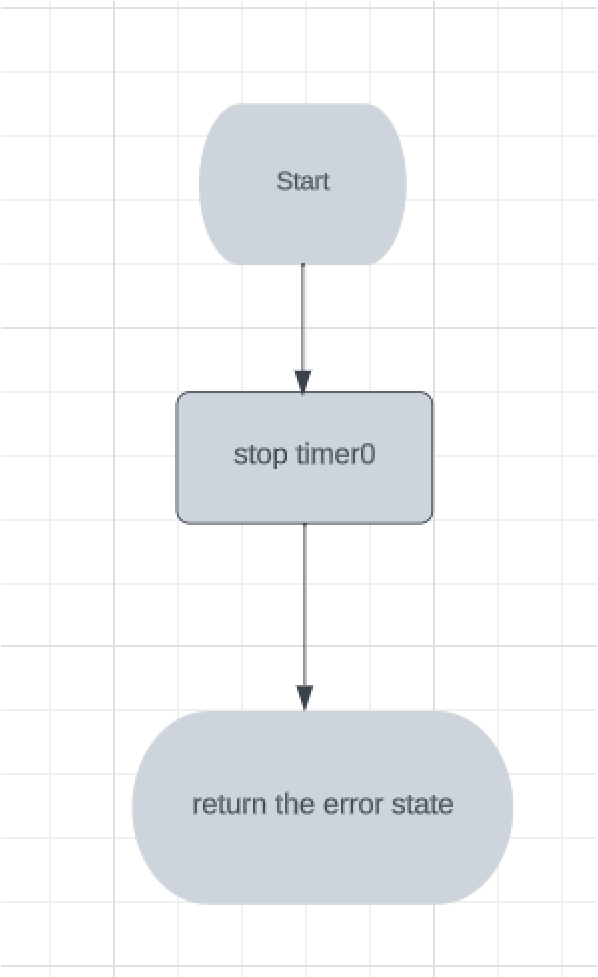
enu_sosErrorState_t sos_modify_task (uint8_t task_id,str_sosTask_t *ptr_str_sosTask);


Start
check in database if the id is saved before
No
Yes
assign new configuration to the old task struct
Return SOS_INVALID_TASK
return SOS_STATUS_SUCCESS

enu_sosErrorState_t sos_run(void);

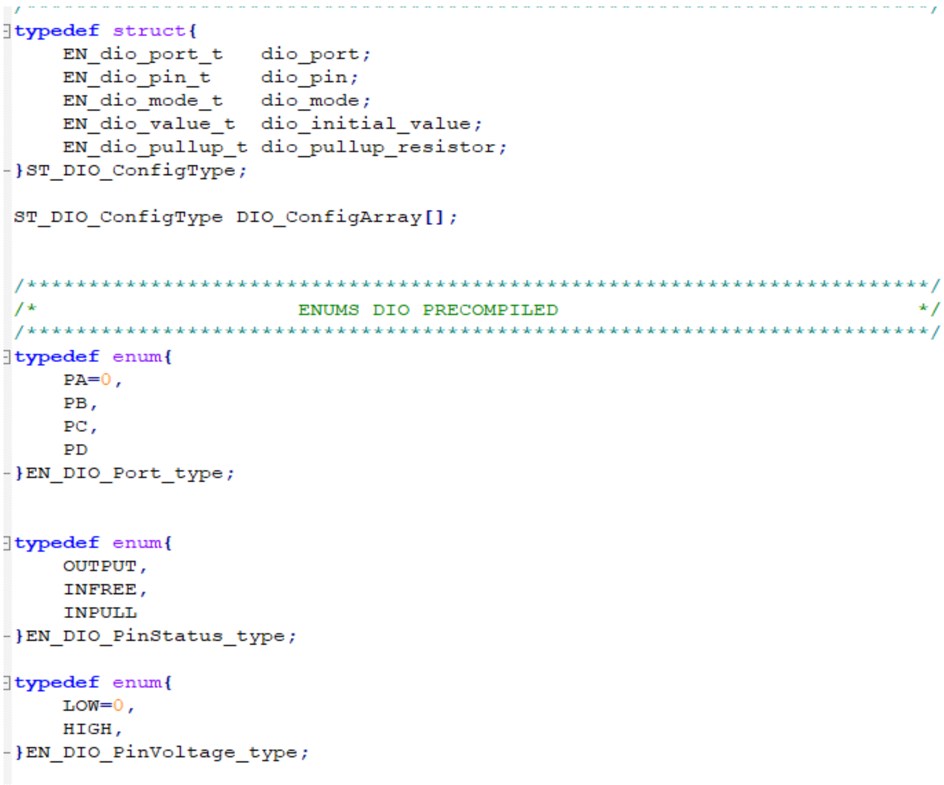

Start
wait until start button is pressed
enters an infinite loop of running tasks at their time in database while stop button isn't pressed
if stop button is pressed
No
yes
set the sos state to RUNNING
stop the sos scheduler and sos state is stopped

enu_sosErrorState_t sos_disable(void);

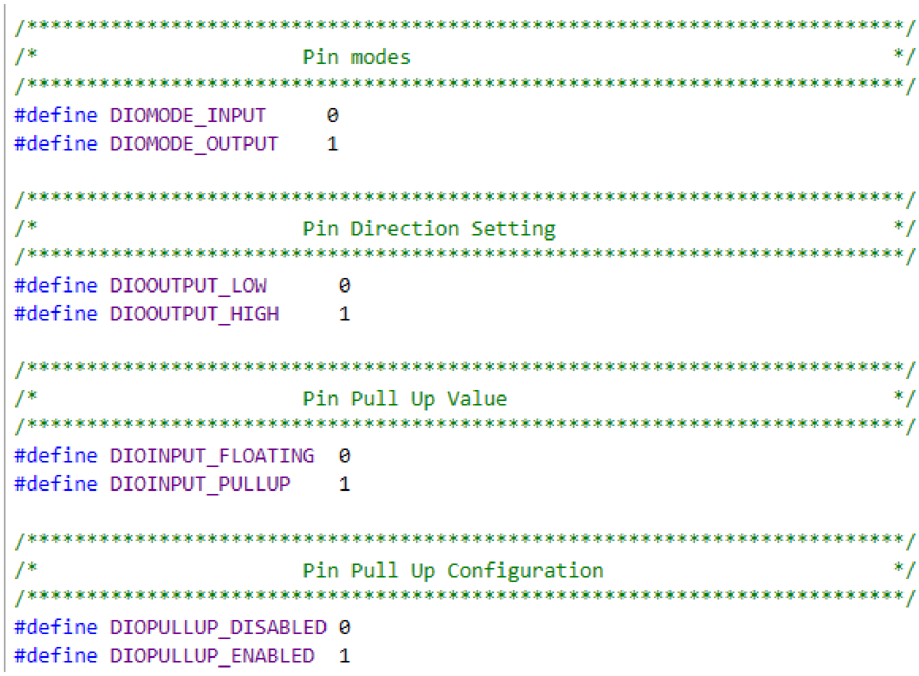


3.2 Configurations:

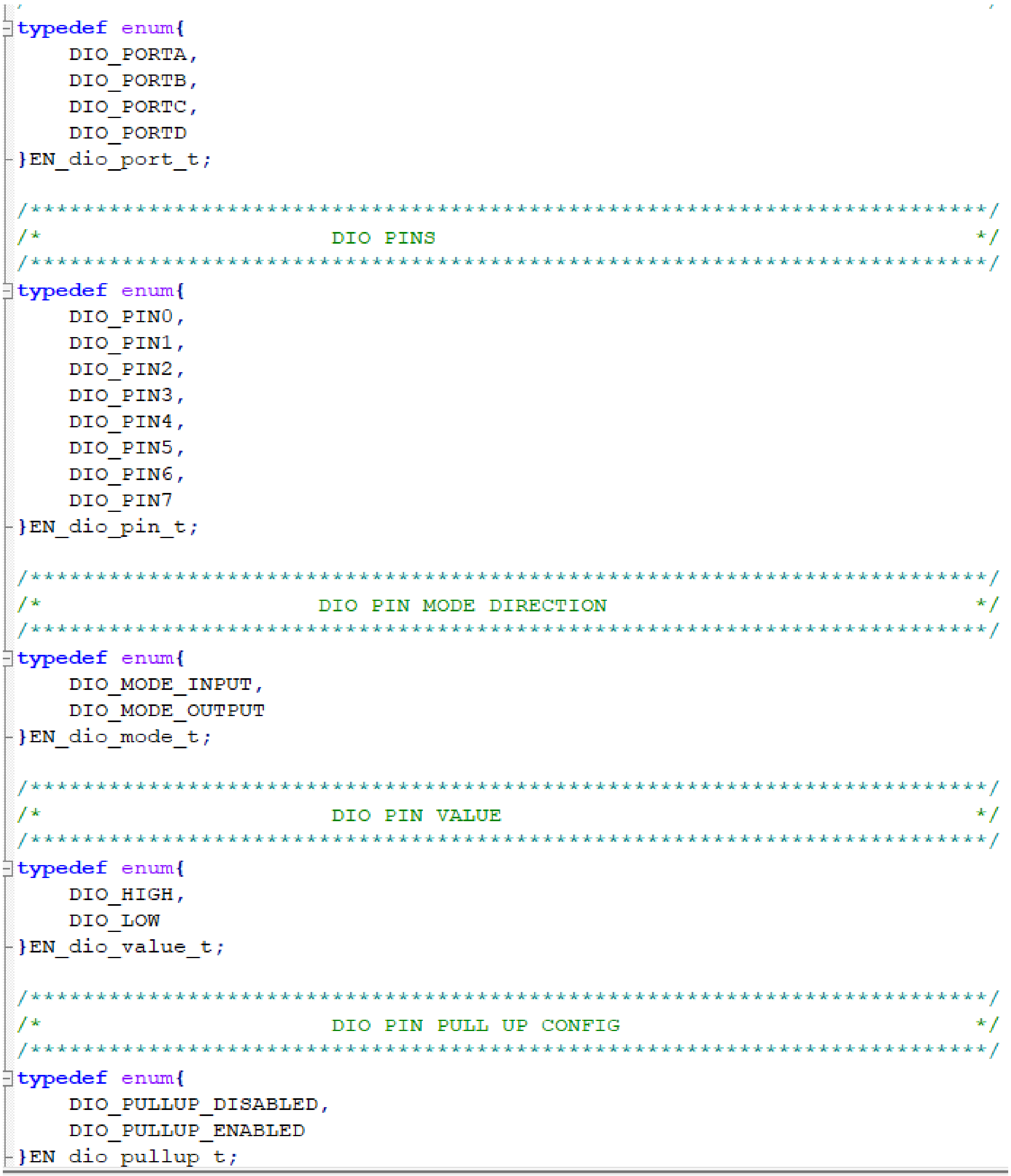
3.2.1. dio :



```
/*************************************************************************/
typedef struct{
    EN_dio_port_t   dio_port;
    EN_dio_pin_t    dio_pin;
    EN_dio_mode_t   dio_mode;
    EN_dio_value_t  dio_initial_value;
    EN_dio_pullup_t dio_pullup_resistor;
}ST_DIO_ConfigType;

ST_DIO_ConfigType DIO_ConfigArray[];


/*************************************************************************/
/*                       ENUMS DIO PRECOMPILED                           */
/*************************************************************************/
typedef enum{
    PA=0,
    PB,
    PC,
    PD
}EN_DIO_Port_type;


typedef enum{
    OUTPUT,
    INFREE,
    INPULL
}EN_DIO_PinStatus_type;

typedef enum{
    LOW=0,
    HIGH,
}EN_DIO_PinVoltage_type;
```

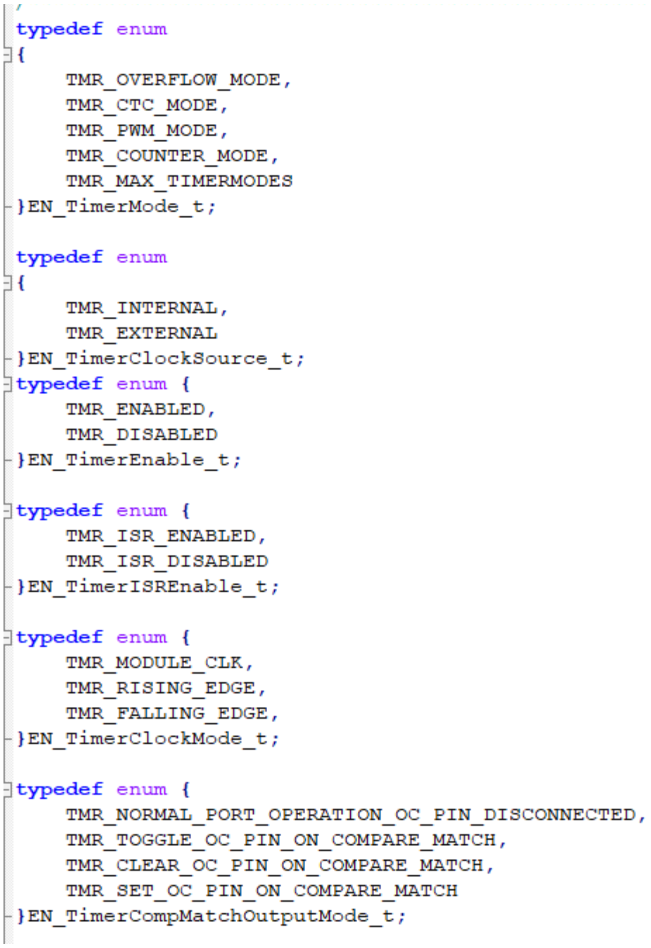


```
/*************************************************************************/
/*                        Pin modes                                      */
/*************************************************************************/
#define DIOMODE_INPUT     0
#define DIOMODE_OUTPUT    1

/*************************************************************************/
/*                        Pin Direction Setting                          */
/*************************************************************************/
#define DIOOUTPUT_LOW      0
#define DIOOUTPUT_HIGH     1

/*************************************************************************/
/*                        Pin Pull Up Value                              */
/*************************************************************************/
#define DIOINPUT_FLOATING  0
#define DIOINPUT_PULLUP    1

/*************************************************************************/
/*                        Pin Pull Up Configuration                      */
/*************************************************************************/
#define DIOPULLUP_DISABLED 0
#define DIOPULLUP_ENABLED  1
```

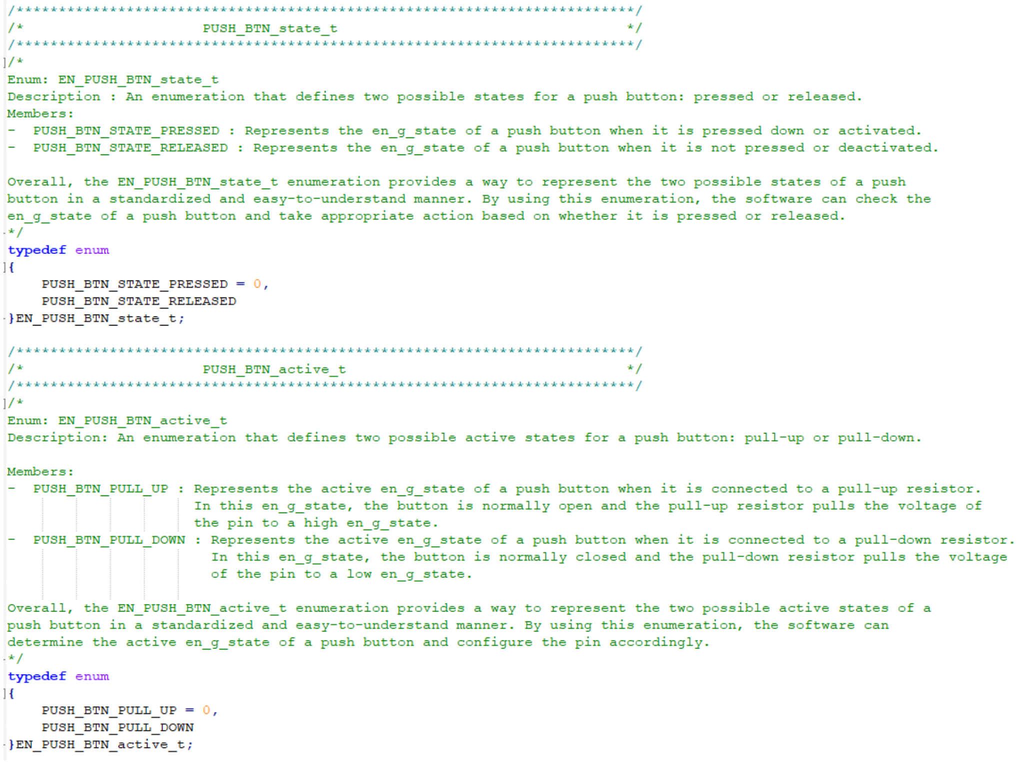


```c
typedef enum{
    DIO_PORTA,
    DIO_PORTB,
    DIO_PORTC,
    DIO_PORTD
}EN_dio_port_t;

/*************************************************************************/
/*                      DIO PINS                                         */
/*************************************************************************/
typedef enum{
    DIO_PIN0,
    DIO_PIN1,
    DIO_PIN2,
    DIO_PIN3,
    DIO_PIN4,
    DIO_PIN5,
    DIO_PIN6,
    DIO_PIN7
}EN_dio_pin_t;

/*************************************************************************/
/*                     DIO PIN MODE DIRECTION                            */
/*************************************************************************/
typedef enum{
    DIO_MODE_INPUT,
    DIO_MODE_OUTPUT
}EN_dio_mode_t;

/*************************************************************************/
/*                      DIO PIN VALUE                                    */
/*************************************************************************/
typedef enum{
    DIO_HIGH,
    DIO_LOW
}EN_dio_value_t;

/*************************************************************************/
/*                      DIO PIN PULL UP CONFIG                           */
/*************************************************************************/
typedef enum{
    DIO_PULLUP_DISABLED,
    DIO_PULLUP_ENABLED
}EN_dio_pullup_t;
```

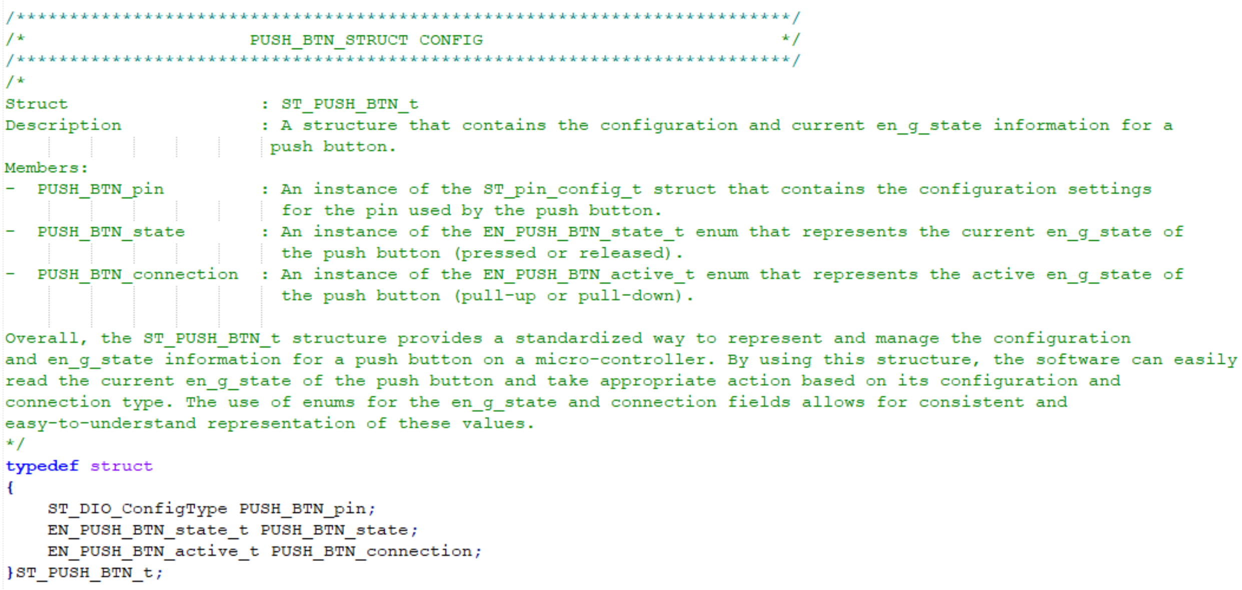
3.2.2. timer :



```c
typedef enum
{
    TMR_OVERFLOW_MODE,
    TMR_CTC_MODE,
    TMR_PWM_MODE,
    TMR_COUNTER_MODE,
    TMR_MAX_TIMERMODES
}EN_TimerMode_t;

typedef enum
{
    TMR_INTERNAL,
    TMR_EXTERNAL
}EN_TimerClockSource_t;
typedef enum {
    TMR_ENABLED,
    TMR_DISABLED
}EN_TimerEnable_t;

typedef enum {
    TMR_ISR_ENABLED,
    TMR_ISR_DISABLED
}EN_TimerISREnable_t;

typedef enum {
    TMR_MODULE_CLK,
    TMR_RISING_EDGE,
    TMR_FALLING_EDGE,
}EN_TimerClockMode_t;

typedef enum {
    TMR_NORMAL_PORT_OPERATION_OC_PIN_DISCONNECTED,
    TMR_TOGGLE_OC_PIN_ON_COMPARE_MATCH,
    TMR_CLEAR_OC_PIN_ON_COMPARE_MATCH,
    TMR_SET_OC_PIN_ON_COMPARE_MATCH
}EN_TimerCompMatchOutputMode_t;
```

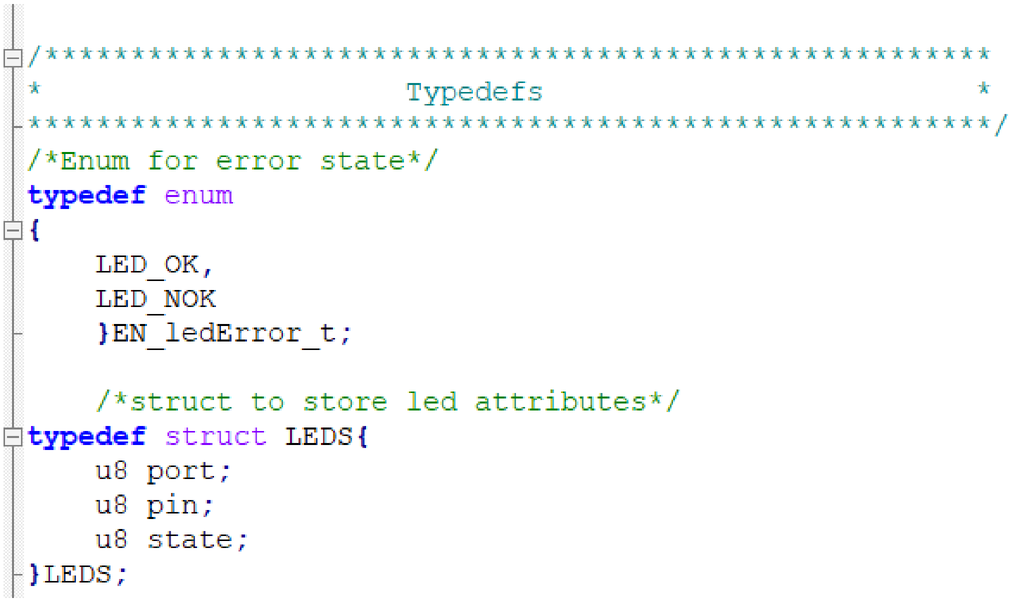


```c
/************************************************************************/
/*                       PUSH_BTN_state_t                               */
/************************************************************************/
/*
Enum: EN_PUSH_BTN_state_t
Description : An enumeration that defines two possible states for a push button: pressed or released.
Members:
-  PUSH_BTN_STATE_PRESSED : Represents the en_g_state of a push button when it is pressed down or activated.
-  PUSH_BTN_STATE_RELEASED : Represents the en_g_state of a push button when it is not pressed or deactivated.

Overall, the EN_PUSH_BTN_state_t enumeration provides a way to represent the two possible states of a push
button in a standardized and easy-to-understand manner. By using this enumeration, the software can check the
en_g_state of a push button and take appropriate action based on whether it is pressed or released.
*/
typedef enum
{
    PUSH_BTN_STATE_PRESSED = 0,
    PUSH_BTN_STATE_RELEASED
}EN_PUSH_BTN_state_t;

/************************************************************************/
/*                       PUSH_BTN_active_t                              */
/************************************************************************/
/*
Enum: EN_PUSH_BTN_active_t
Description: An enumeration that defines two possible active states for a push button: pull-up or pull-down.

Members:
-  PUSH_BTN_PULL_UP : Represents the active en_g_state of a push button when it is connected to a pull-up resistor.
                      In this en_g_state, the button is normally open and the pull-up resistor pulls the voltage of
                      the pin to a high en_g_state.
-  PUSH_BTN_PULL_DOWN : Represents the active en_g_state of a push button when it is connected to a pull-down resistor.
                        In this en_g_state, the button is normally closed and the pull-down resistor pulls the voltage
                        of the pin to a low en_g_state.

Overall, the EN_PUSH_BTN_active_t enumeration provides a way to represent the two possible active states of a
push button in a standardized and easy-to-understand manner. By using this enumeration, the software can
determine the active en_g_state of a push button and configure the pin accordingly.
*/
typedef enum
{
    PUSH_BTN_PULL_UP = 0,
    PUSH_BTN_PULL_DOWN
}EN_PUSH_BTN_active_t;
```



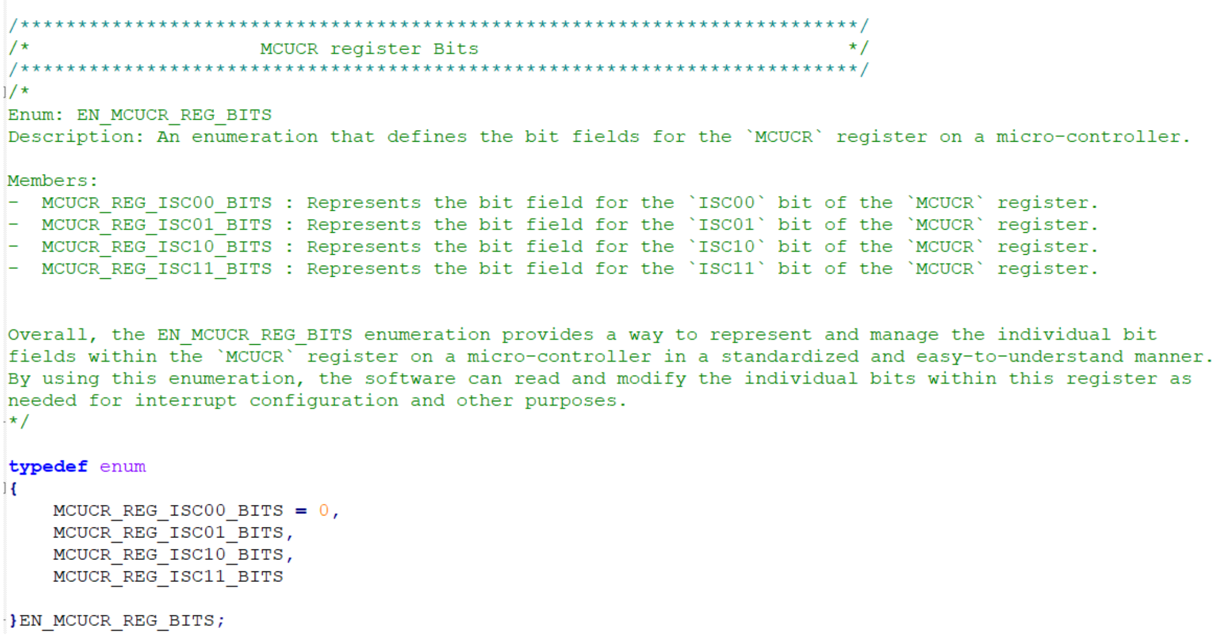
```c
/************************************************************************/
/*                       PUSH_BTN_STRUCT CONFIG                         */
/************************************************************************/
/*
Struct                  : ST_PUSH_BTN_t
Description             : A structure that contains the configuration and current en_g_state information for a
                         push button.
Members:
-  PUSH_BTN_pin         : An instance of the ST_pin_config_t struct that contains the configuration settings
                          for the pin used by the push button.
-  PUSH_BTN_state       : An instance of the EN_PUSH_BTN_state_t enum that represents the current en_g_state of
                          the push button (pressed or released).
-  PUSH_BTN_connection  : An instance of the EN_PUSH_BTN_active_t enum that represents the active en_g_state of
                          the push button (pull-up or pull-down).

Overall, the ST_PUSH_BTN_t structure provides a standardized way to represent and manage the configuration
and en_g_state information for a push button on a micro-controller. By using this structure, the software can easily
read the current en_g_state of the push button and take appropriate action based on its configuration and
connection type. The use of enums for the en_g_state and connection fields allows for consistent and
easy-to-understand representation of these values.
*/
typedef struct
{
    ST_DIO_ConfigType PUSH_BTN_pin;
    EN_PUSH_BTN_state_t PUSH_BTN_state;
    EN_PUSH_BTN_active_t PUSH_BTN_connection;
}ST_PUSH_BTN_t;
```

3.2.4. led :

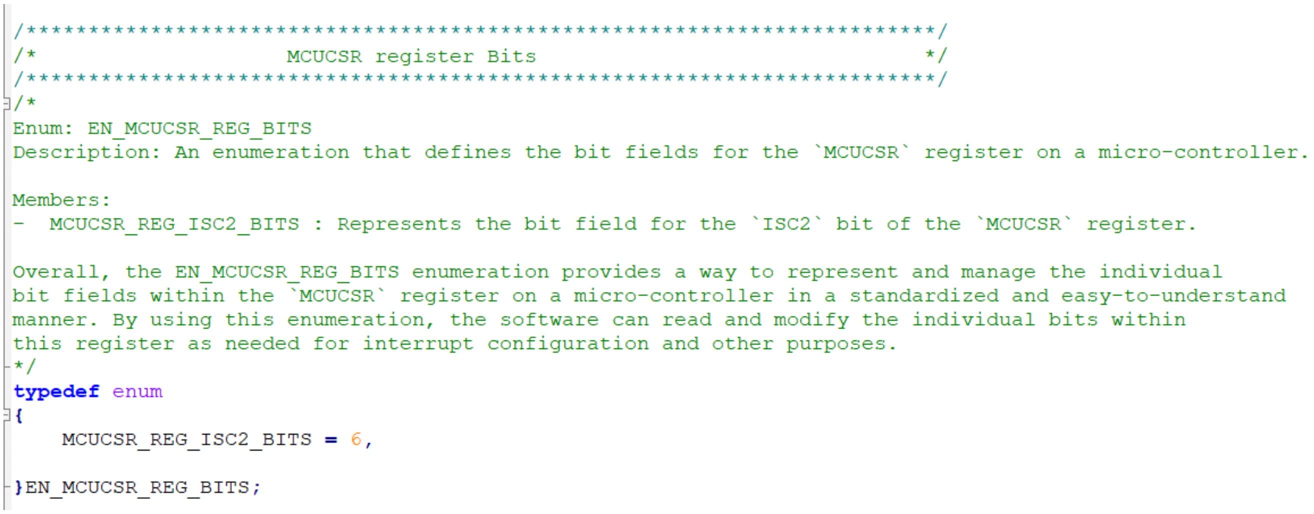


```
/********************************************************
*                      Typedefs                         *
*********************************************************/
/*Enum for error state*/
typedef enum
{
    LED_OK,
    LED_NOK
    }EN_ledError_t;

    /*struct to store led attributes*/
typedef struct LEDS{
    u8 port;
    u8 pin;
    u8 state;
}LEDS;
```

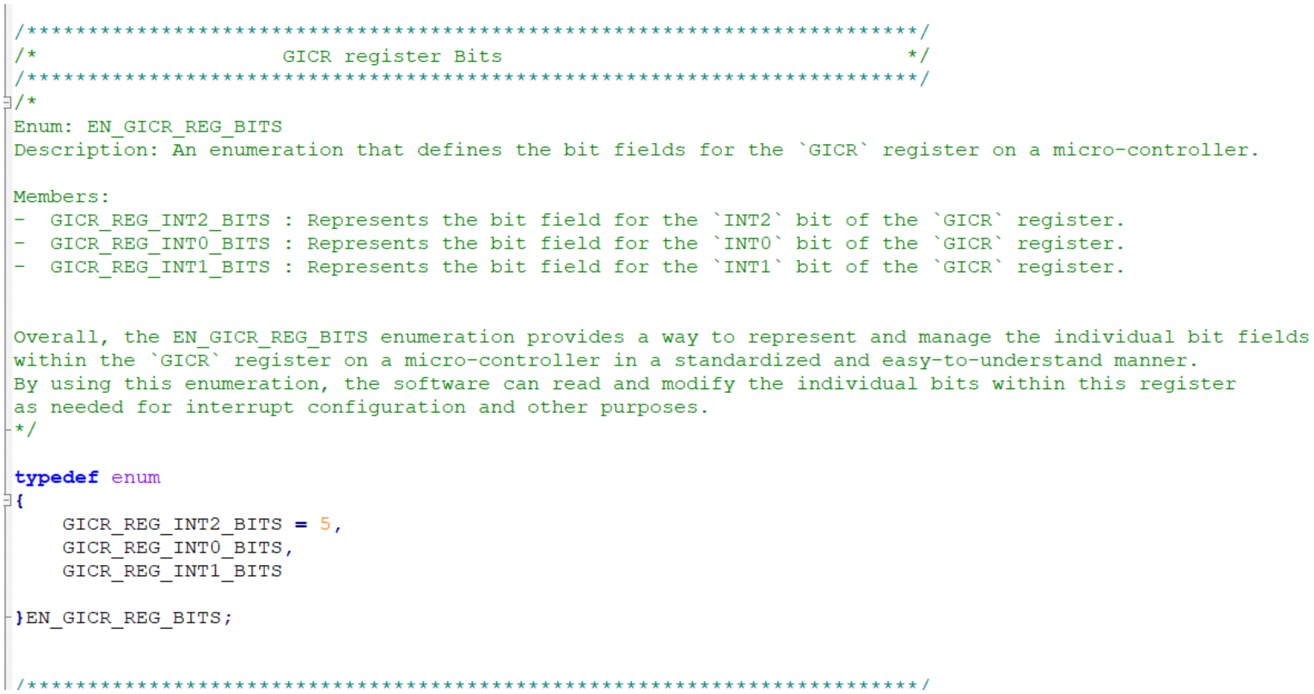
3.2.5 External Interrupt:



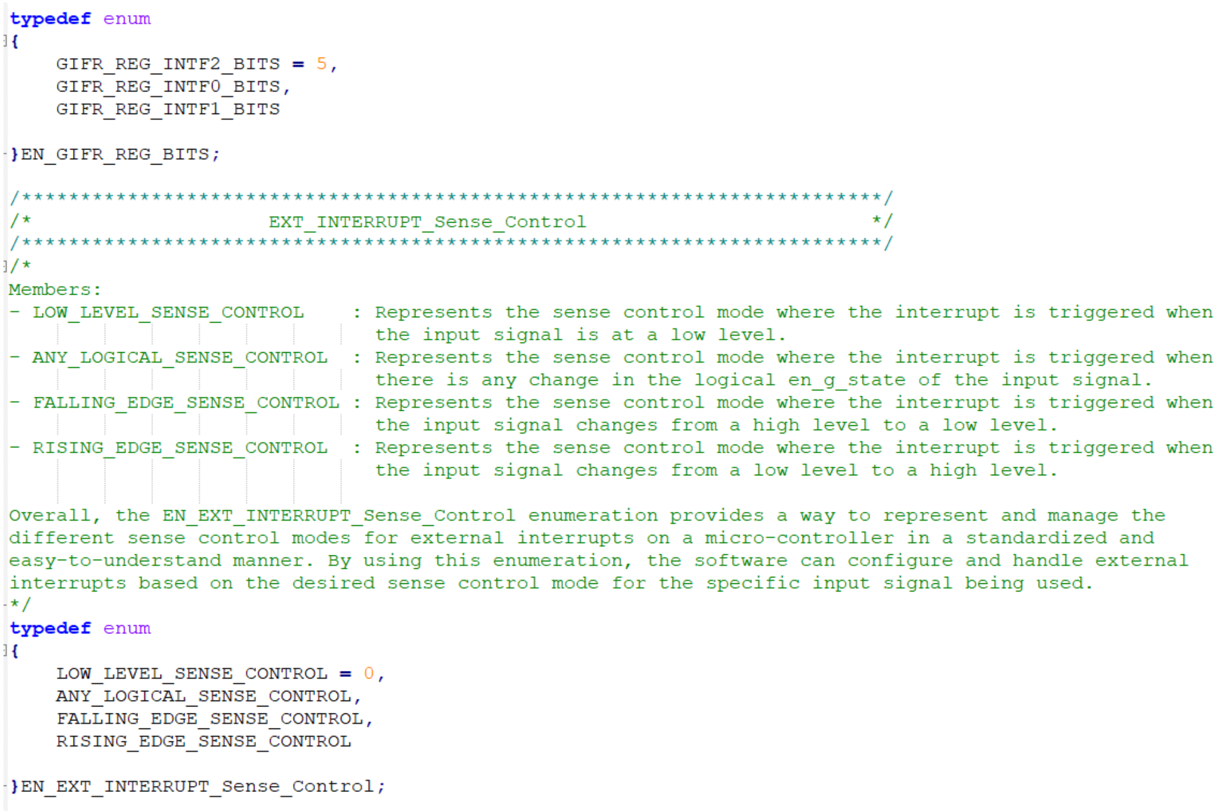
```
/*************************************************************************/
/*                    MCUCR register Bits                               */
/*************************************************************************/
/*
Enum: EN_MCUCR_REG_BITS
Description: An enumeration that defines the bit fields for the `MCUCR` register on a micro-controller.

Members:
-  MCUCR_REG_ISC00_BITS : Represents the bit field for the `ISC00` bit of the `MCUCR` register.
-  MCUCR_REG_ISC01_BITS : Represents the bit field for the `ISC01` bit of the `MCUCR` register.
-  MCUCR_REG_ISC10_BITS : Represents the bit field for the `ISC10` bit of the `MCUCR` register.
-  MCUCR_REG_ISC11_BITS : Represents the bit field for the `ISC11` bit of the `MCUCR` register.


Overall, the EN_MCUCR_REG_BITS enumeration provides a way to represent and manage the individual bit
fields within the `MCUCR` register on a micro-controller in a standardized and easy-to-understand manner.
By using this enumeration, the software can read and modify the individual bits within this register as
needed for interrupt configuration and other purposes.
*/

typedef enum
{
    MCUCR_REG_ISC00_BITS = 0,
    MCUCR_REG_ISC01_BITS,
    MCUCR_REG_ISC10_BITS,
    MCUCR_REG_ISC11_BITS

}EN_MCUCR_REG_BITS;
```



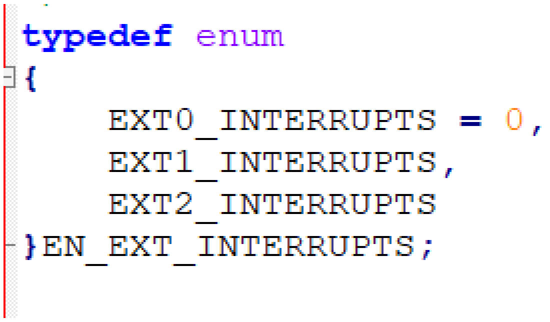
```
/*************************************************************************/
/*                    MCUCSR register Bits                               */
/*************************************************************************/
/*
Enum: EN_MCUCSR_REG_BITS
Description: An enumeration that defines the bit fields for the `MCUCSR` register on a micro-controller.

Members:
-  MCUCSR_REG_ISC2_BITS : Represents the bit field for the `ISC2` bit of the `MCUCSR` register.

Overall, the EN_MCUCSR_REG_BITS enumeration provides a way to represent and manage the individual
bit fields within the `MCUCSR` register on a micro-controller in a standardized and easy-to-understand
manner. By using this enumeration, the software can read and modify the individual bits within
this register as needed for interrupt configuration and other purposes.
*/
typedef enum
{
    MCUCSR_REG_ISC2_BITS = 6,

}EN_MCUCSR_REG_BITS;
```



```
/***********************************************************************/
/*                    GICR register Bits                               */
/***********************************************************************/
/*
Enum: EN_GICR_REG_BITS
Description: An enumeration that defines the bit fields for the `GICR` register on a micro-controller.

Members:
-  GICR_REG_INT2_BITS : Represents the bit field for the `INT2` bit of the `GICR` register.
-  GICR_REG_INT0_BITS : Represents the bit field for the `INT0` bit of the `GICR` register.
-  GICR_REG_INT1_BITS : Represents the bit field for the `INT1` bit of the `GICR` register.


Overall, the EN_GICR_REG_BITS enumeration provides a way to represent and manage the individual bit fields
within the `GICR` register on a micro-controller in a standardized and easy-to-understand manner.
By using this enumeration, the software can read and modify the individual bits within this register
as needed for interrupt configuration and other purposes.
*/

typedef enum
{
    GICR_REG_INT2_BITS = 5,
    GICR_REG_INT0_BITS,
    GICR_REG_INT1_BITS

}EN_GICR_REG_BITS;


/***********************************************************************/
```



```
typedef enum
{
    GIFR_REG_INTF2_BITS = 5,
    GIFR_REG_INTF0_BITS,
    GIFR_REG_INTF1_BITS

}EN_GIFR_REG_BITS;

/****************************************************************************/
/*                      EXT_INTERRUPT_Sense_Control                         */
/****************************************************************************/
/*
Members:
- LOW_LEVEL_SENSE_CONTROL    : Represents the sense control mode where the interrupt is triggered when
                               the input signal is at a low level.
- ANY_LOGICAL_SENSE_CONTROL  : Represents the sense control mode where the interrupt is triggered when
                               there is any change in the logical en_g_state of the input signal.
- FALLING_EDGE_SENSE_CONTROL : Represents the sense control mode where the interrupt is triggered when
                               the input signal changes from a high level to a low level.
- RISING_EDGE_SENSE_CONTROL  : Represents the sense control mode where the interrupt is triggered when
                               the input signal changes from a low level to a high level.

Overall, the EN_EXT_INTERRUPT_Sense_Control enumeration provides a way to represent and manage the
different sense control modes for external interrupts on a micro-controller in a standardized and
easy-to-understand manner. By using this enumeration, the software can configure and handle external
interrupts based on the desired sense control mode for the specific input signal being used.
*/
typedef enum
{
    LOW_LEVEL_SENSE_CONTROL = 0,
    ANY_LOGICAL_SENSE_CONTROL,
    FALLING_EDGE_SENSE_CONTROL,
    RISING_EDGE_SENSE_CONTROL

}EN_EXT_INTERRUPT_Sense_Control;
```



```
typedef enum
{
    EXT0_INTERRUPTS = 0,
    EXT1_INTERRUPTS,
    EXT2_INTERRUPTS
}EN_EXT_INTERRUPTS;
```

3.3. OS APIs :

3.3.1. sos_init :

Function Name	sos_init
Syntax	enu_system_status_t sos_init (void);
Synch/Asynch	Synchronous
Reentrancy	Non-Reentrant
Parameters(in):	None
Parameters(out):	None
Parameters(in,out):	None
Return:	SOS_STATUS_SUCCESS: In case of Successful Operation
	SOS_STATUS_INVALID_STATE: In case The SOS is already initialized

3.3.2. sos_deinit :

Function Name	sos_deinit
Syntax	enu_system_status_t sos_deinit (void);
Synch/Asynch	Synchronous
Reentrancy	Non-Reentrant
Parameters(in):	None
Parameters(out):	None
Parameters(in,out):	None
Return:	TMU_STATUS_SUCCESS: In case of Successful Operation
	SOS_STATUS_INVALID_STATE: In case The SOS is not initialized previously or is already de-initialized.

3.3.3. sos_run :

Function Name	sos_run
Syntax	enu_system_status_t sos_run (void);
Synch/Asynch	Synchronous
Reentrancy	Non-Reentrant
Parameters(in):	None
Parameters(out):	None
Parameters(in,out):	None
Return:	SOS_STATUS_SUCCESS: In case of Successful Operation
	SOS_STATUS_FAILED: In case of the SOS is already running

3.3.4. sos_disable:

Function Name	sos_disable
Syntax	enu_system_status_t sos_disable (void);
Synch/Asynch	Synchronous
Reentrancy	Non-Reentrant
Parameters(in):	None
Parameters(out):	None
Parameters(in,out):	None
Return:	SOS_STATUS_SUCCESS: In case of Successful Operation
	SOS_STATUS_FAILED: In case of the SOS is already stopped

3.3.5. sos_create_task :

Function Name	sos_create_task
Syntax	enu_system_status_t sos_create_task (uint8_t task_id,str_sosTask_t *ptr_str_sosTask);
Synch/Asynch	Synchronous
Reentrancy	Non-Reentrant
Parameters(in):	task_id:the id of the task to be created *ptr_str_sosTask:holds task's configuration
Parameters(out):	None
Parameters(in,out):	None
Return:	SOS_STATUS_SUCCESS: In case of Successful Operation
	SOS_NULL_PTR: In case of NULL pointer
	SOS_INVALID_ARG:In case of wrong arguments
	SOS_DUPLICATED_ID:In case this ID is already created

3.3.6. sos_modify_task :

Function Name	sos_modify_task
Syntax	enu_system_status_t sos_modify_task (uint8_t task_id,str_sosTask_t *ptr_str_sosTask);
Synch/Asynch	Synchronous
Reentrancy	Non-Reentrant
Parameters(in):	task_id:the id of the task to be modified *ptr_str_sosTask:holds task's configuration
Parameters(out):	None
Parameters(in,out):	None
Return:	SOS_STATUS_SUCCESS: In case of Successful Operation

	SOS_NULL_PTR: In case of NULL pointer
	SOS_INVALID_TASK:In case of wrong task not found

3.3.6. sos_delete_task :

Function Name	sos_delete_task
Syntax	enu_system_status_t sos_delete_task (uint8_t task_id,str_sosTask_t *ptr_str_sosTask);
Synch/Asynch	Synchronous
Reentrancy	Non-Reentrant
Parameters(in):	task_id:the id of the task to be deleted *ptr_str_sosTask:holds task's configuration to set to null
Parameters(out):	None
Parameters(in,out):	None
Return:	SOS_STATUS_SUCCESS: In case of Successful Operation
	SOS_INVALID_TASK:In case of wrong task not found