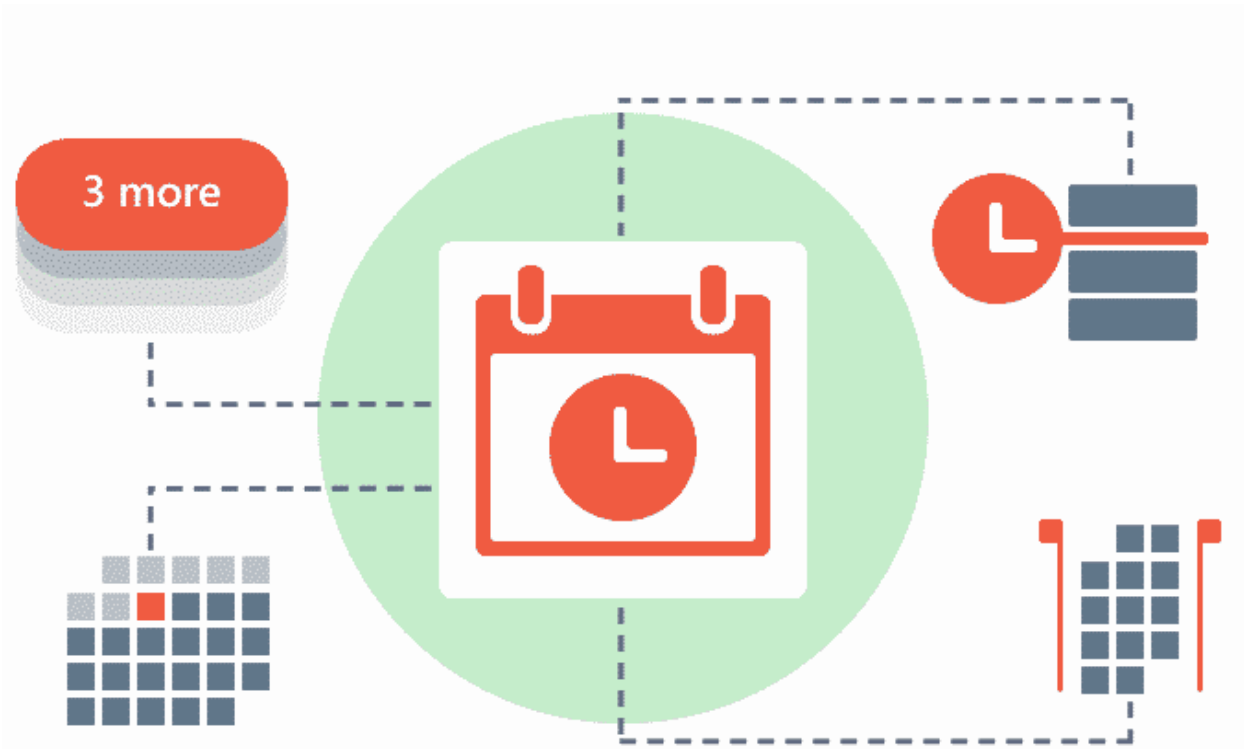


Design a Small OS

Prepared by:

Nadeen Adel

Table Of Content:

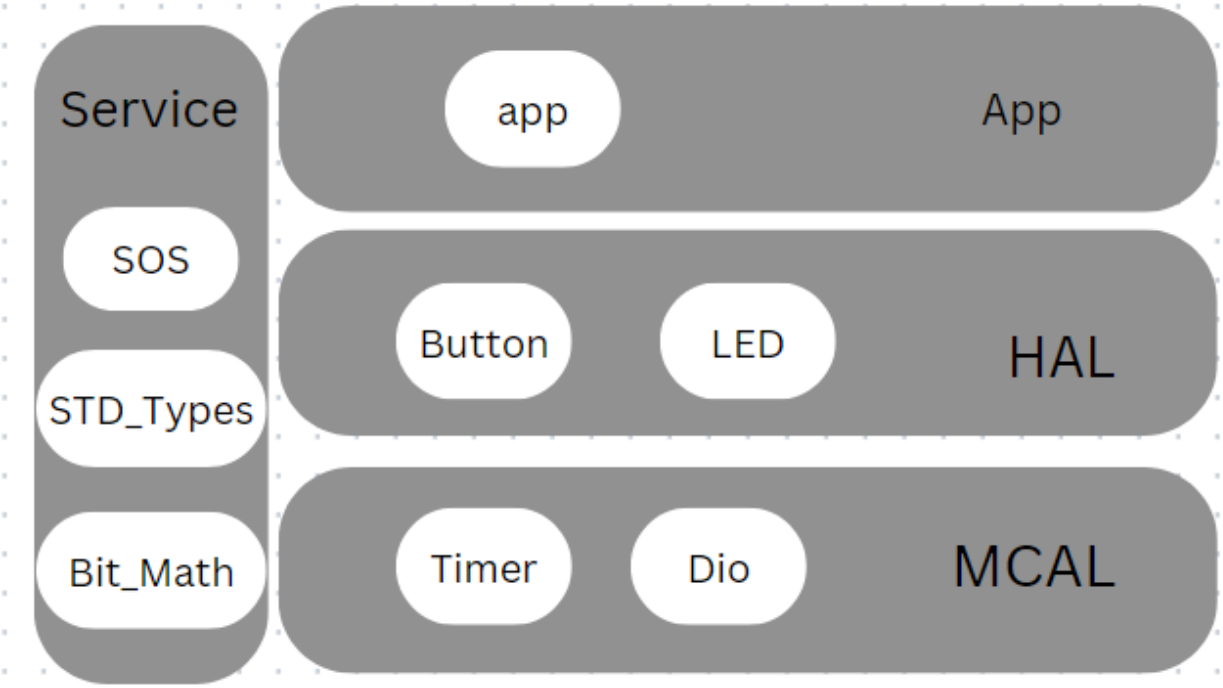
Table Of Content: 1
1. Introduction: 2
2. High Level Design: 3
2.1 Layered Architecture: 3
2.2 Modules Description: 3
2.3 Drivers' Documentation: 4
2.3.1 DIO: 4
2.3.2 Timer: 5
2.3.3 LED: 6
2.3.4 Push Button: 6
3. Low Level Design: 7
3.1Flowcharts: 7
3.2 Configurations: 13
3.2.1 DIO: 13
3.2.2 Timer: 16
3.2.3 Push Button: 17
3.2.4 LED: 18
OS APIs: 18
SOS Class Diagram : 22
System Sequence Diagram: 23
SOS State Machine: 24

1. Introduction:

This project aims to create a small and efficient RTOS tailored for resource-constrained systems. SOS is designed to be flexible, easy to use, and well-optimized for embedded applications where real-time task management is crucial. The system employs a preemptive priority-based scheduling algorithm to ensure efficient and deterministic task execution.

2. High Level Design:

2.1 Layered Architecture:



2.2 Modules Description:

- DIO: controls GPIO pins.
- Timer: controls timing of the tasks in the program .
- LED:controls led state in the program.
- Button: controls start and stop of the system.
- SOS: operating system that manages application process.
- App: Contains main logic of the code.

2.3 Drivers' Documentation:

2.3.1 DIO:

```c
/*
 * 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:

```
/**
 * @brief Initializes TIMER0 in Normal mode and optionally enables interrupts.
 * This function initializes TIMER0 in Normal mode. If the 'en_a_interrputEnable' parameter
 * is set to EN_TIMER_INTERRUPT_ENABLE, it enables TIMER0 interrupt. If set to
 * EN_TIMER_INTERRUPT_DISABLE, interrupts are not enabled.
 * @param en_a_interrputEnable: Specifies whether to enable TIMER0 interrupts or not.
 * @return EN_TIMER_ERROR_T: An error code indicating the success or failure of the initialization.
 */

EN_TIMER_ERROR_T TIMER_timer0NormalModeInit(EN_TIMER_INTERRPUT_T en_a_interrputEnable);


/**
 * @brief Starts TIMER0 with the specified prescaler value.
 * This function starts TIMER0 with the given prescaler value 'u16_a_prescaler'.
 * @param u16_a_prescaler: The prescaler value to set for TIMER0.
 * @return EN_TIMER_ERROR_T: An error code indicating the success or failure of starting TIMER0.
 */

EN_TIMER_ERROR_T TIMER_timer0Start(u16 u16_a_prescaler);

/**
 * @brief Stops TIMER0.

 * This function stops TIMER0.
 * @param None.
 * @return None.
 */

void TIMER_timer0Stop(void);
```

```
/**
 * @brief Sets an overflow (OVF) callback function for TIMER2.
 * This function allows you to specify a custom callback function 'void_a_pfOvfInterruptAction'
 * that will be executed when TIMER2 overflows.
 * @param void_a_pfOvfInterruptAction: A function pointer to the custom overflow callback function.
 * @return EN_TIMER_ERROR_T: An error code indicating the success or failure of setting the callback.
 */

EN_TIMER_ERROR_T TMR_ovfSetCallback(void (*void_a_pfOvfInterruptAction)(void));
```

2.3.3 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);
```

2.3.4 Push Button:

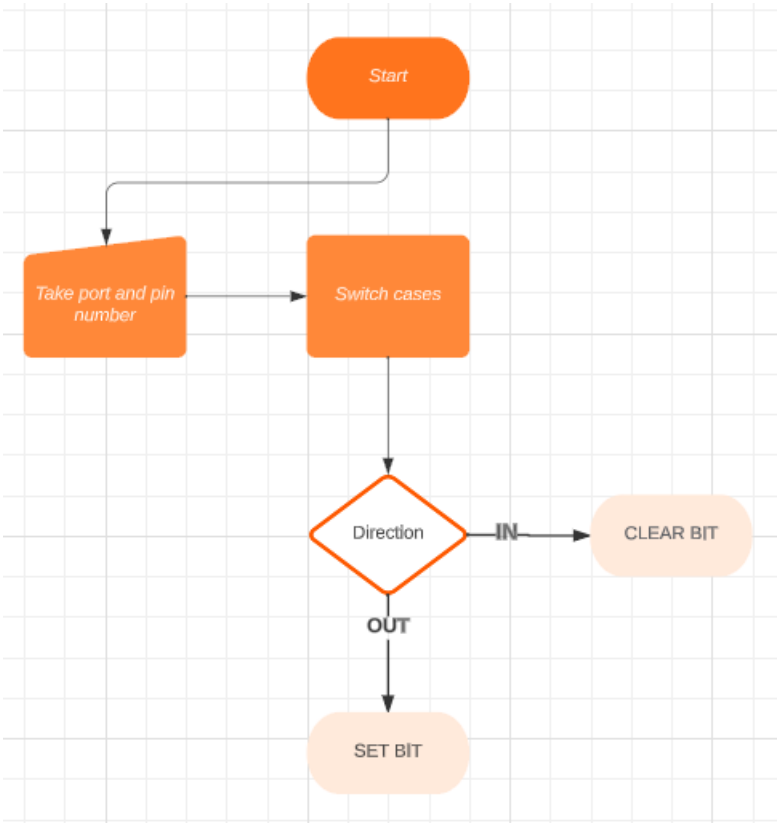
```
/*
Function: PUSH_BTN_intialize
Description: Initializes a push button based on the configuration settings specified in the input parameter.
Overall, the PUSH_BTN_intialize function provides a way to initialize a push button based on the specified
configuration settings. By using this function, the software can ensure that the pin used by the push button
is configured correctly and the push button is ready for use.
*/
EN_pushBTNError_t PUSH_BTN_intialize();

/*
Function     : PUSH_BTN_read_state
Description  : Reads the current en_g_state of a push button and returns its value.

Parameters:
-  btn       : A pointer to an ST_PUSH_BTN_t struct that contains the configuration settings and current en_g_state
               information for the push button.
-  btn_state : A pointer to an EN_PUSH_BTN_state_t enum where the current en_g_state of the push button
               will be stored.

Overall, the PUSH_BTN_read_state function provides a way to read the current en_g_state of a push button and return
its value. By using this function, the software can determine whether the push button is currently pressed or
released and take appropriate action based on its en_g_state.
*/
EN_pushBTNError_t PUSH_BTN_read_state(u8 btnNumber, EN_PUSH_BTN_state_t *btn_state);
```

3. Low Level Design:

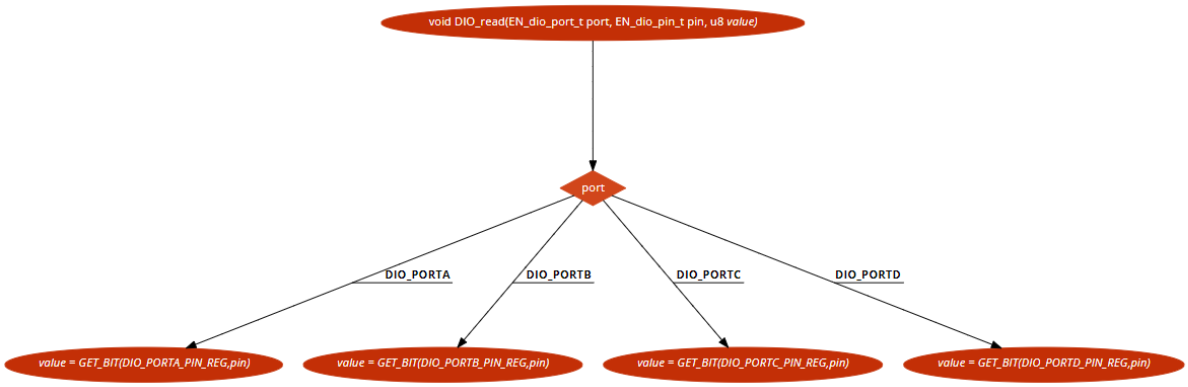
3.1Flowcharts:

DIO:

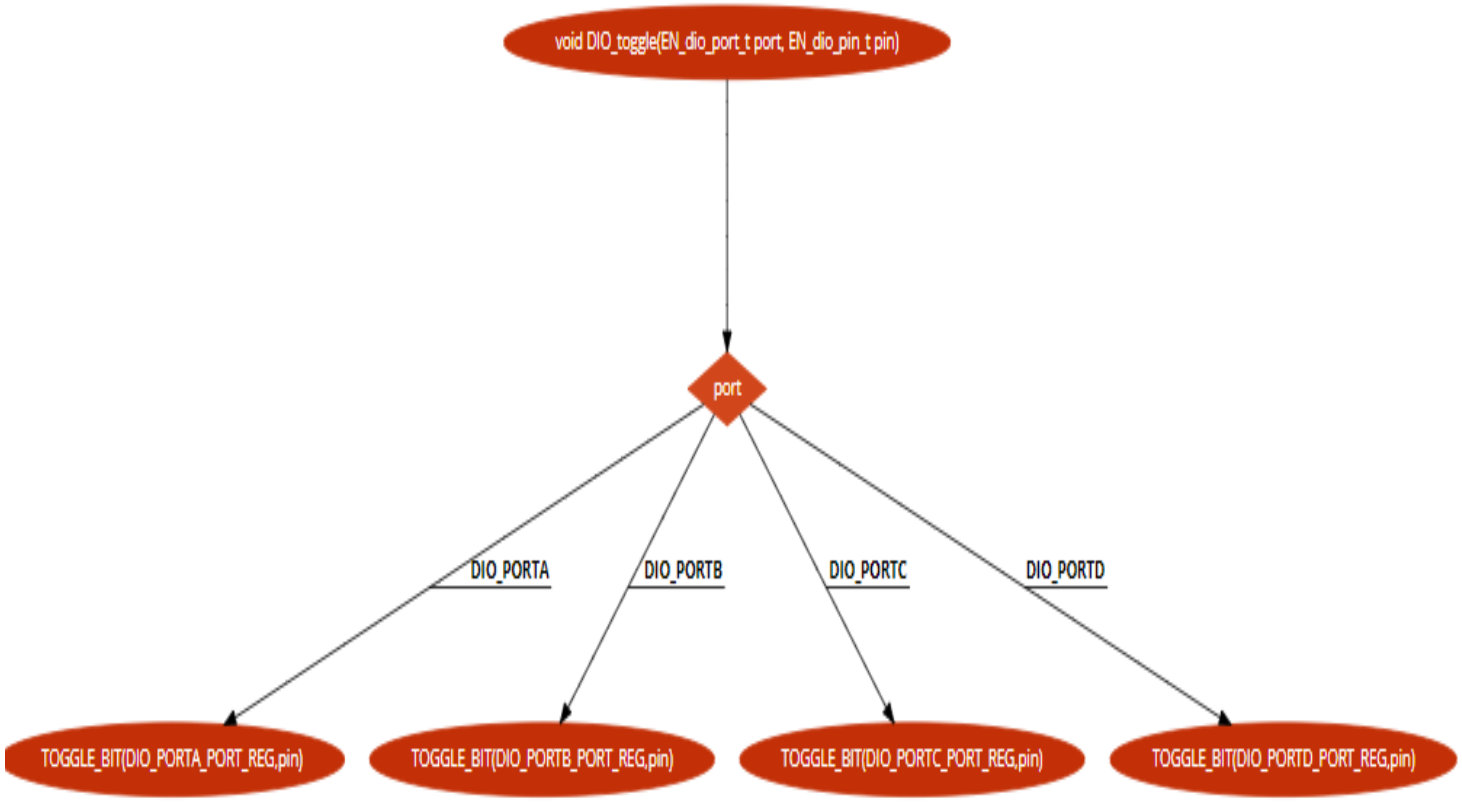
EN_dioError_t DIO_WritePin(EN_dio_port_t port, EN_dio_pin_t pin, EN_dio_value_t value)



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

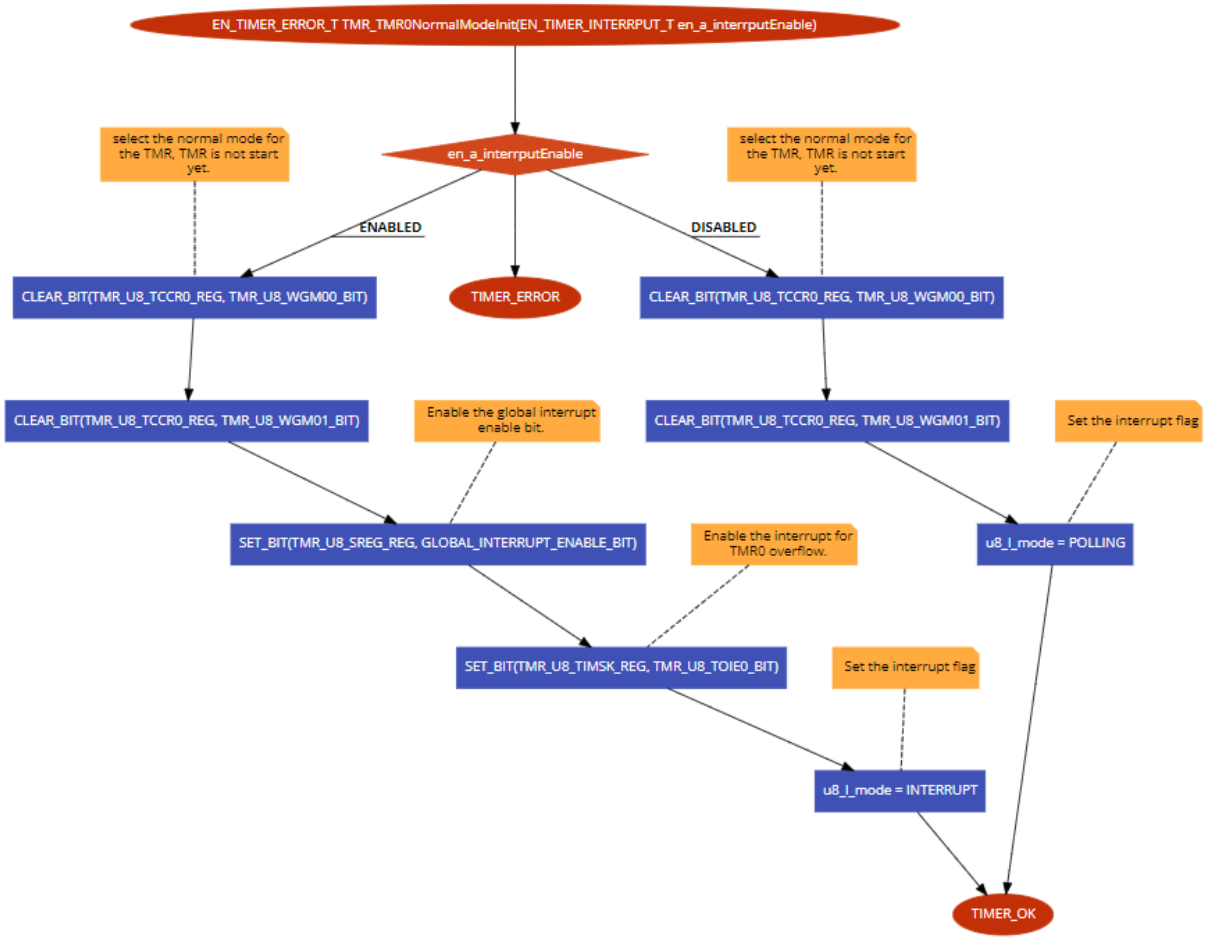


EN_dioError_t DIO_toggle(EN_dio_port_t port, EN_dio_pin_t pin)

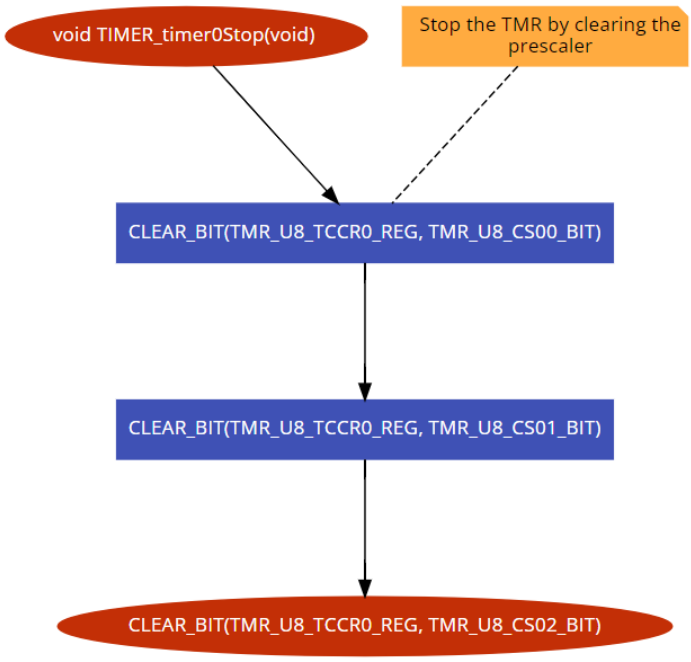


Timer:

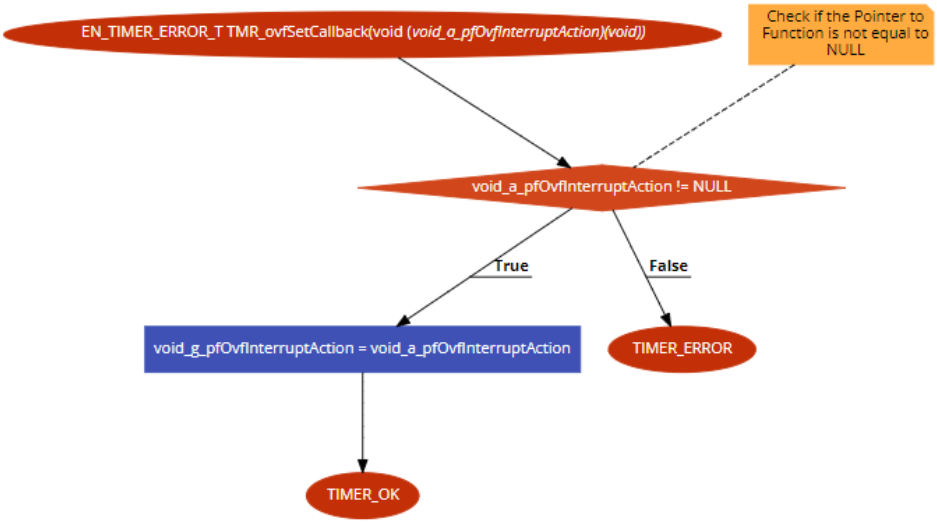
EN_TIMER_ERROR_T TMR_TMR0NormalModeInit(EN_TIMER_INTERRPUT_T en_a_interrputEnable)



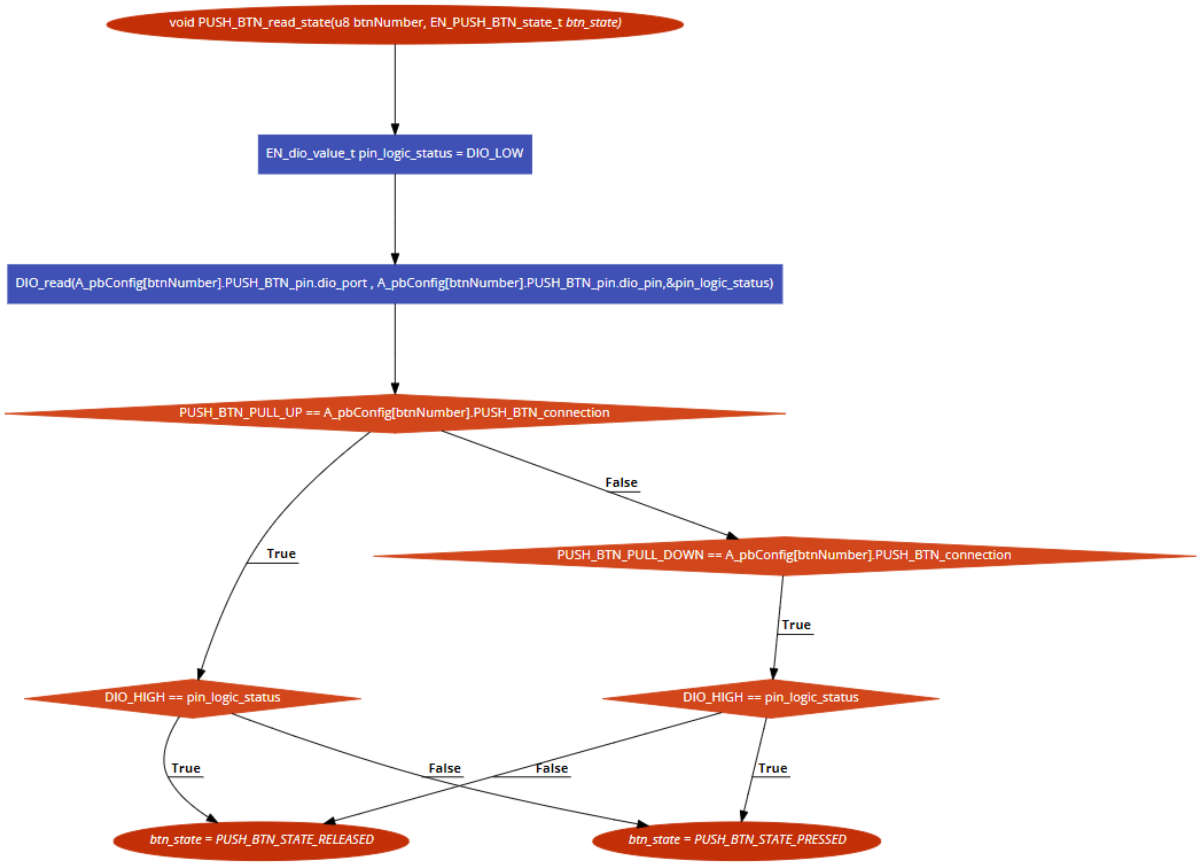
EN_TIMER_ERROR_T TIMER_timer0Stop(void)



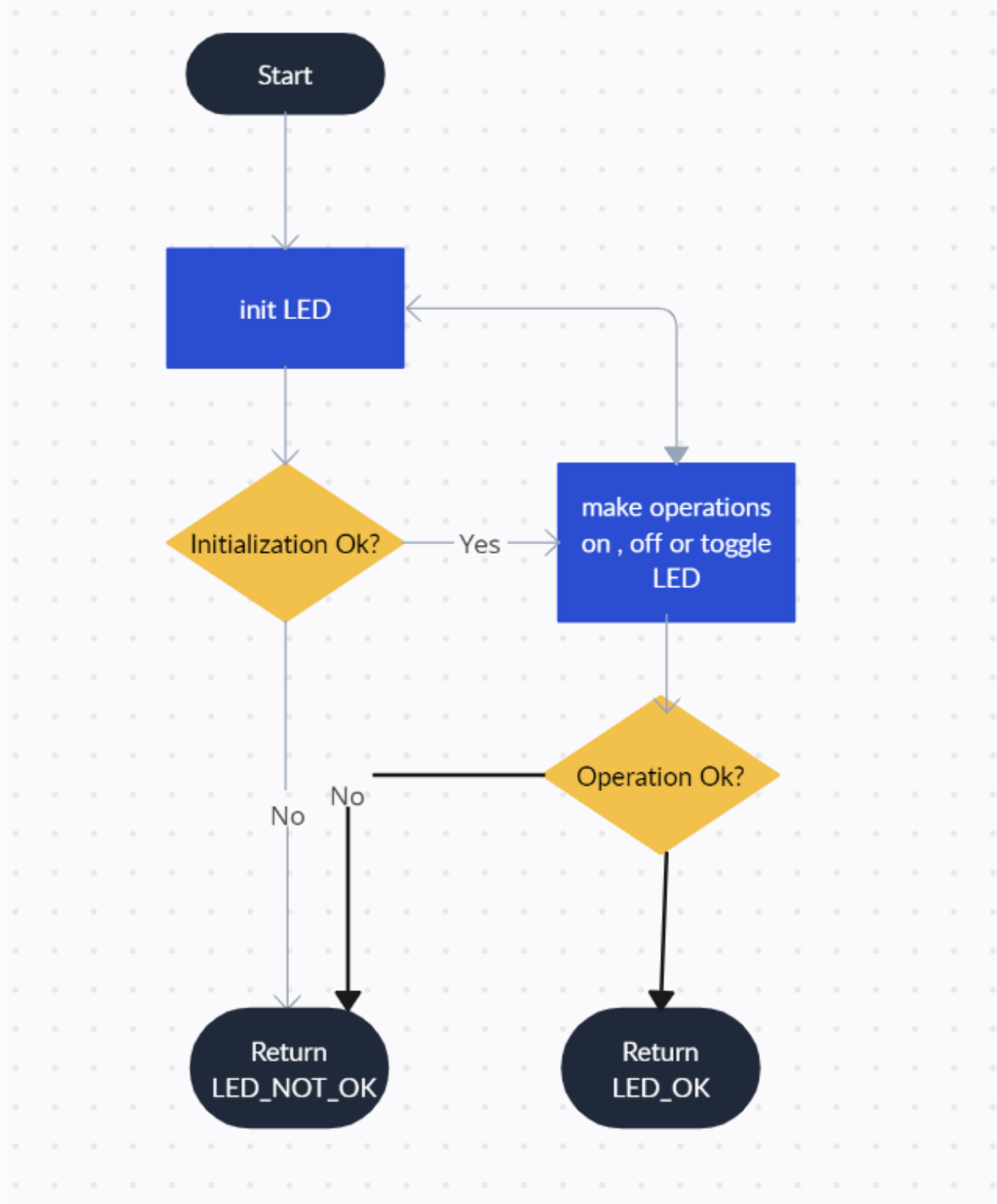
EN_TIMER_ERROR_T TMR_ovfSetCallback(void (*void_a_pfOvfInterruptAction)(void))



Push Button:


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
True
False
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

LED:


Start
init LED
Initialization Ok?
Yes
make operations on , off or toggle LED
Operation Ok?
No
No
Return LED_NOT_OK
Return LED_OK

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;
```

```c
/*****************************************************************************/
/*                      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:

```
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;
```

3.2.3 Push Button:

```
/*******************************************************************************/
/*                      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;
```

```
/*******************************************************************************/
/*                      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;
```

OS APIs:

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

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 already initialized

sos_create_task()

Function Name	sos_create_task
Syntax	enu_system_status_t sos_create_task (str_sosTask_t *ptr_str_sosTask);
Synch/Asynch	Synchronous
Reentrancy	Non-Reentrant
Parameters(in):	*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_modify_task()

Function Name	sos_modify_task
Syntax	enu_system_status_t sos_modify_task (str_sosTask_t *ptr_str_sosTask);
Synch/Asynch	Synchronous
Reentrancy	Non-Reentrant
Parameters(in):	*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

sos_delete_task()

Function Name	sos_delete_task
Syntax	enu_system_status_t sos_delete_task (uint8_t task_id);
Synch/Asynch	Synchronous
Reentrancy	Non-Reentrant
Parameters(in):	task_id:the id of the task to be deleted
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

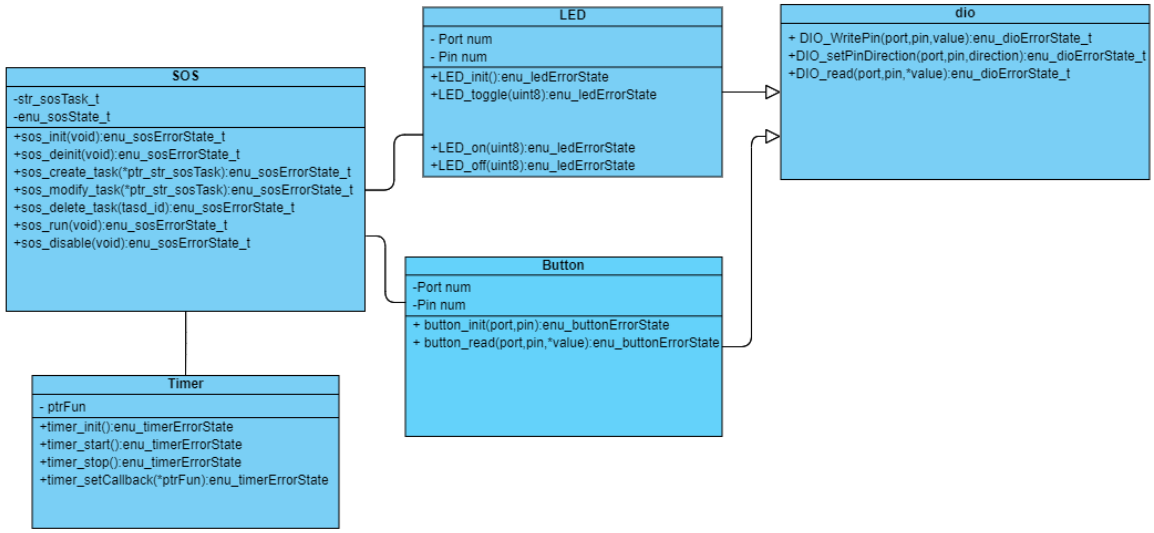
sos_run()

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 not running

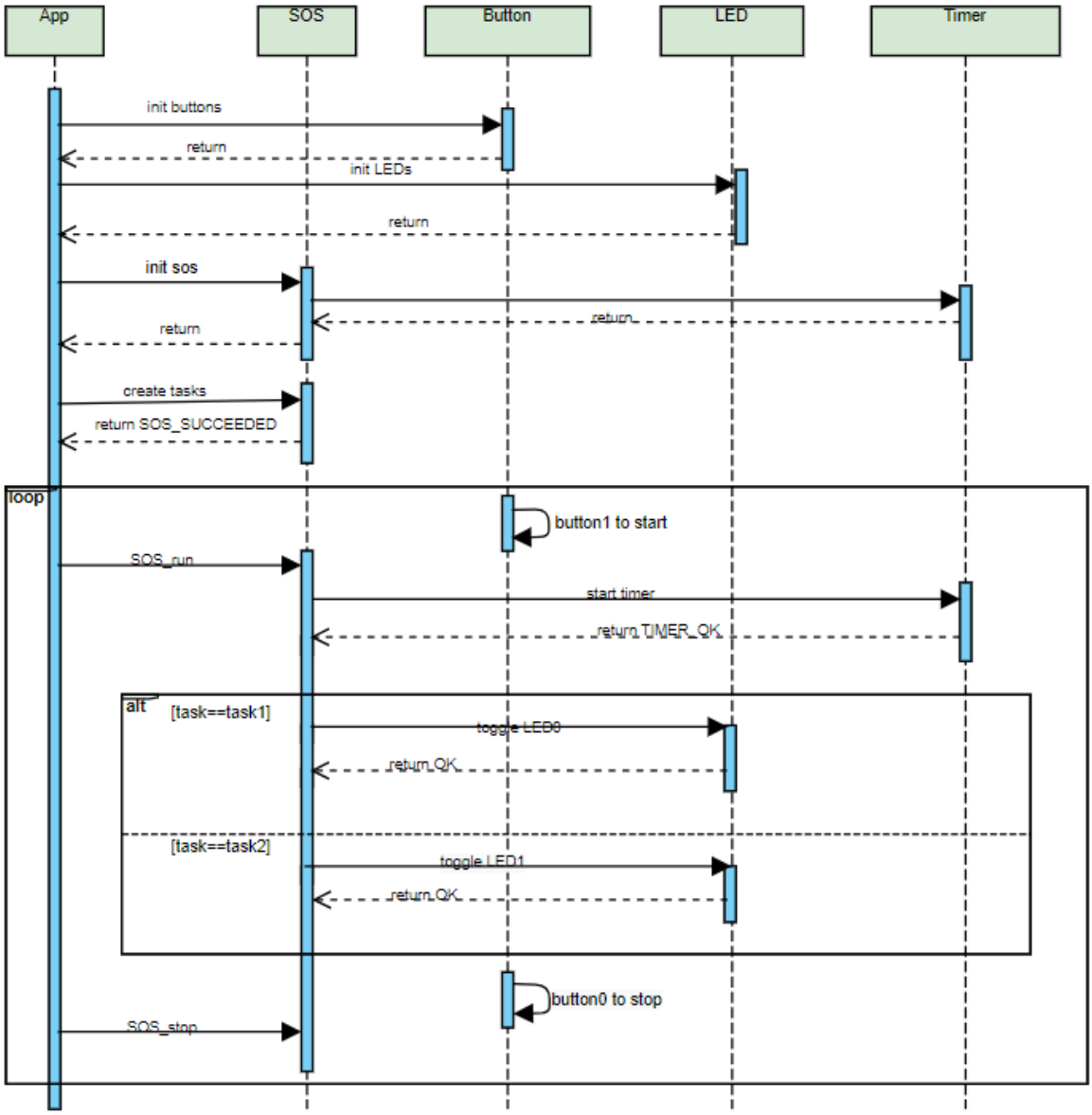
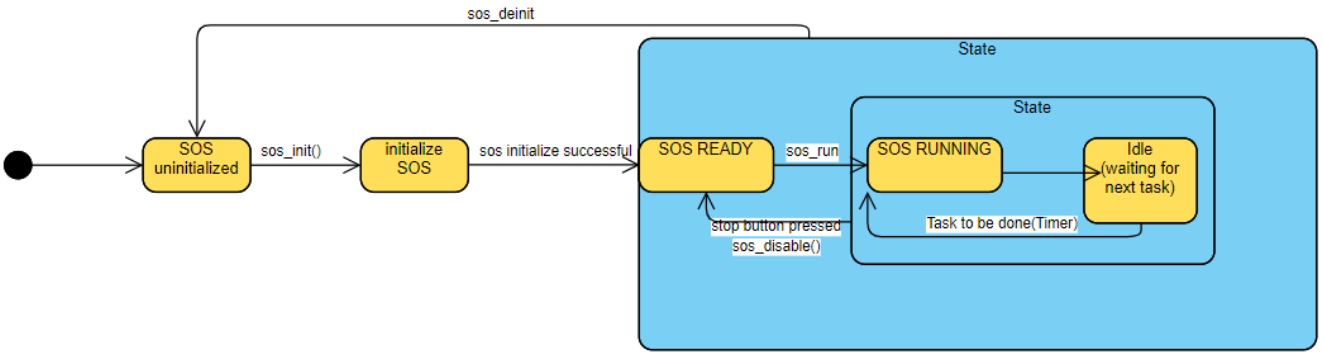
sos_disable()

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

SOS Class Diagram :



System Sequence Diagram:



SOS State Machine: