SPRINTS

# Small OS Design

Version 1.0

May 2023

Presented by
**Abdelrhman Walaa**

Presented to
**Sprints**

# Table Of Content

**Small OS Design**

# 1. Project Introduction

In this project, the task is to design a small OS with a priority based preemptive scheduler based on time-triggered.

## 1.1. System Requirements

### 1.1.1. Hardware Requirements

1. **ATmega32** microcontroller
2. **PBUTTON0** to stop the SOS
3. **PBUTTON1** to run the SOS
4. **Two LEDs** to be toggled

### 1.1.2. Software Requirements

1. Implement an application that calls SOS module and use **2** tasks
2. **Task_1**: Toggle LED_0 (Every **300** Millisecond)
3. **Task_2**: Toggle LED_1 (Every **500** Millisecond)
4. Make sure these tasks occur periodically and forever
5. When pressing **BUTTON 0**, the SOS will stop
6. When pressing **BUTTON 1**, the SOS will run

# 2. High Level Design

## 2.1. System Architecture

### 2.1.1. Definition

*Layered Architecture* (*Figure 1*) describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software.

*Microcontroller Abstraction Layer* (*MCAL*) is a software module that directly accesses on-chip MCU peripheral modules and external devices that are mapped to memory, and makes the upper software layer independent of the MCU.

*Hardware Abstraction Layer* (*HAL*) is a layer of programming that allows a computer OS to interact with a hardware device at a general or abstract level rather than at a detailed hardware level.

*Middleware Layer* (*MWL*) is usually the software layer that mediates between application software and the kernel or device driver software.

### 2.1.2. Layered Architecture

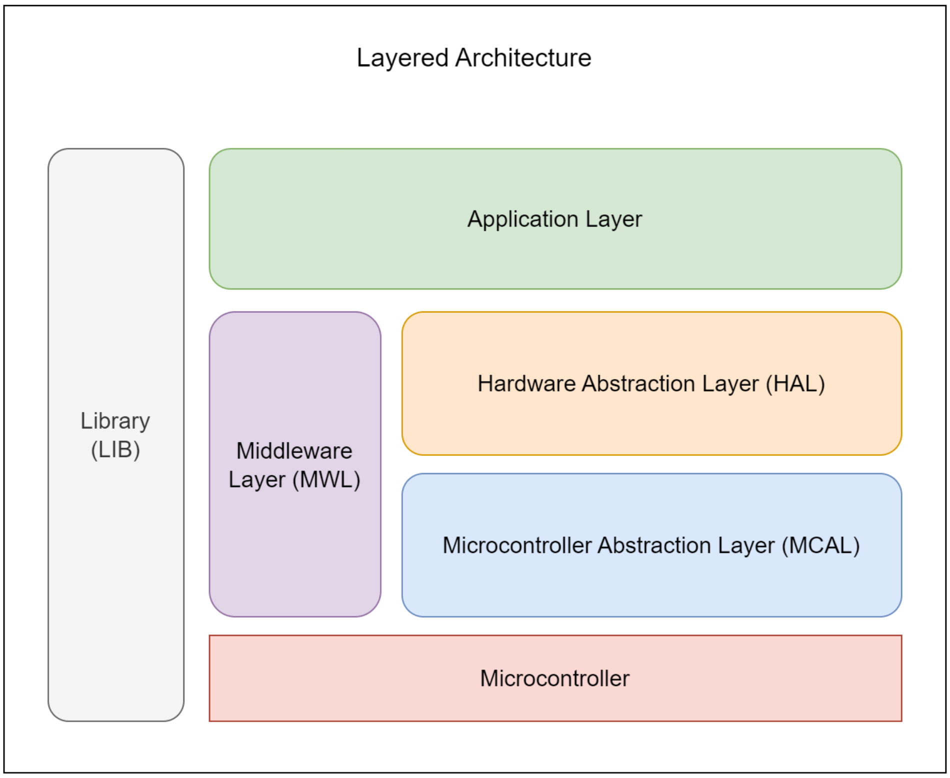



*Figure 1. Layered Architecture Design*

## 2.2. System Modules

### 2.2.1. Definition

A *Module* is a distinct assembly of components that can be easily added, removed or replaced in a larger system. Generally, a *Module* is not functional on its own.

In computer hardware, a *Module* is a component that is designed for easy replacement.

### 2.2.2. Design

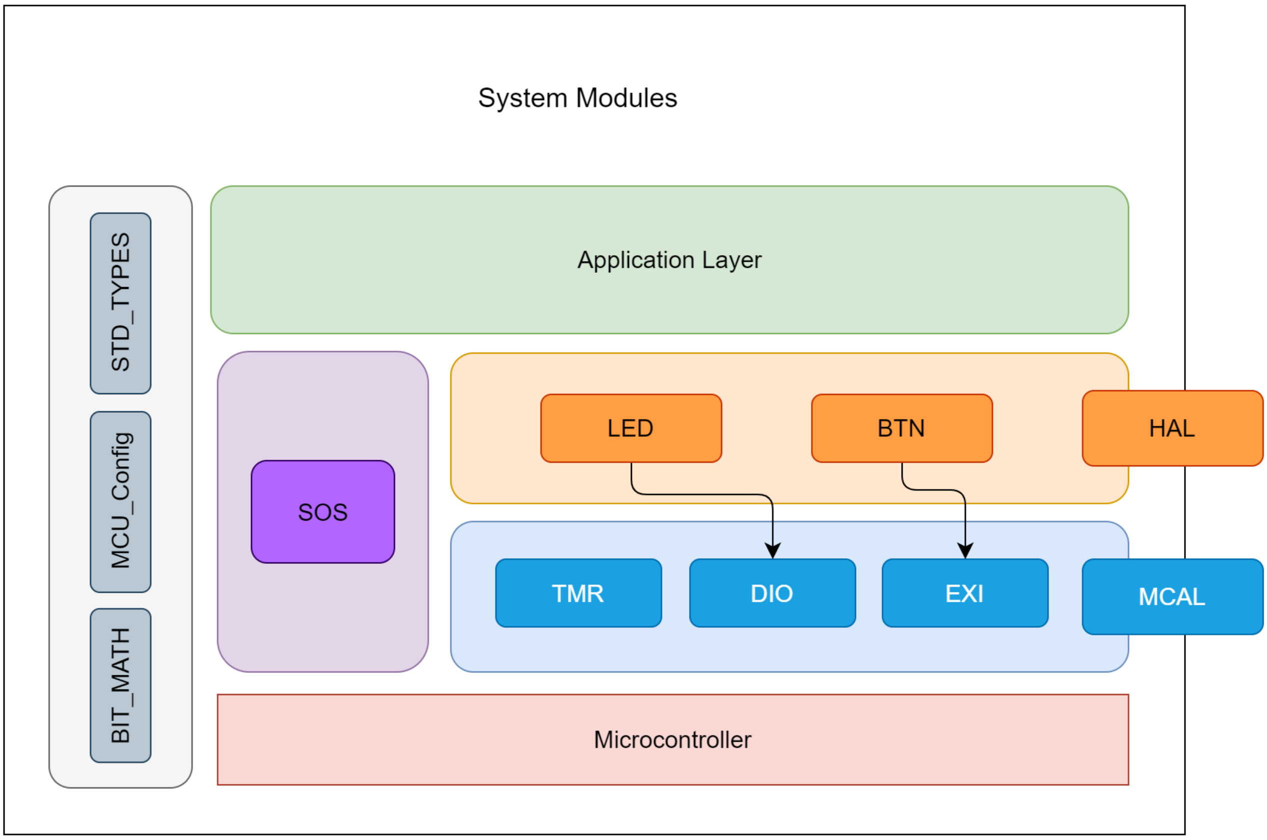



*Figure 2. System Modules Design*

## 2.3. Modules Description

### 2.3.1. DIO Module

The *DIO* (Digital Input/Output) module is responsible for reading input signals from the system's sensors (such as buttons) and driving output signals to the system's actuators (such as *LEDs*). It provides a set of APIs to configure the direction and mode of each pin (input/output, pull-up/down resistor), read the state of an input pin, and set the state of an output pin.

### 2.3.2. EXI Module

The *EXI* (External Interrupt) module is responsible for detecting external events that require immediate attention from the microcontroller, such as a button press. It provides a set of APIs to enable/disable external interrupts for specific pins, set the interrupt trigger edge (rising/falling/both), and define an interrupt service routine (*ISR*) that will be executed when the interrupt is triggered.

### 2.3.3. TMR Module

It is a module which is responsible for calculations and configurations of timer zero.
It provides an API to start counting till the setted time in ms then stop counting after finishing.

### 2.3.4. LED Module

*LED* (Light Emitting Diode) is responsible for controlling the state of the systems' LEDs.
It provides a set of APIs to turn off, to turn on, or to toggle the state of each led.

### 2.3.5. BTN Module

In most of the embedded electronic projects you may want to use a *BTN* (Push Button) switch to give user inputs to the microcontroller. Push Button is basically a small controlling device that is pressed to operate any electrical device.

### 2.3.6. SOS Module

*SOS* (Small Operating System) is the design of a small OS with a priority based on Non preemptive scheduler based on time triggered Used to executing multiple tasks at different time intervals by design simple scheduler a scheduler can be viewed as a simple operating system that allows tasks to be called periodically or (less commonly) on a one-shot basis.
Also a scheduler can be viewed as a single timer interrupt service routine that is shared between many different tasks. As a result, only one timer needs to be initialized, and any changes to the timing generally require only one function to be altered.

## 2.4. Drivers' Documentation (APIs)

### 2.4.1 Definition

An *API* is an *Application Programming Interface* that defines a set of *routines*, *protocols* and *tools* for creating an application. An *API* defines the high level interface of the behavior and capabilities of the component and its inputs and outputs.

An *API* should be created so that it is generic and implementation independent. This allows for the API to be used in multiple applications with changes only to the implementation of the API and not the general interface or behavior.

### 2.4.2. MCAL APIs

#### 2.4.2.1. DIO Driver APIs

```
Precompile Configurations Snippet:

/* Initial Directions of Port A Pins*/
/* Options: DIO_U8_INITIAL_INPUT
 *          DIO_U8_INITIAL_OUTPUT
 */

/* PORTA */
#define DIO_U8_PA0_INITIAL_DIRECTION    DIO_U8_INITIAL_INPUT
#define DIO_U8_PA1_INITIAL_DIRECTION    DIO_U8_INITIAL_INPUT
                •
                •
#define DIO_U8_PA7_INITIAL_DIRECTION    DIO_U8_INITIAL_INPUT

/* Initial Values of Port A Pins */
/* Options:  DIO_U8_INPUT_FLOATING
 *           DIO_U8_INPUT_PULLUP_RESISTOR
 *           DIO_U8_OUTPUT_LOW
 *           DIO_U8_OUTPUT_HIGH
 */

/* PORTA */
#define DIO_U8_PA0_INITIAL_VALUE       DIO_U8_INPUT_FLOATING
#define DIO_U8_PA1_INITIAL_VALUE       DIO_U8_INPUT_FLOATING
                •
                •
#define DIO_U8_PA7_INITIAL_VALUE       DIO_U8_INPUT_FLOATING
```

```
Linking Configurations Snippet:

There are no Linking Configurations as the defined APIs below could change the
DIO peripheral Configurations during the Runtime.

| Name: DIO_vdInitialization
| Input: void
| Output: void
| Description: Function to initialize DIO peripheral.
|
vd DIO_vdInitialization (void)

| Name: DIO_u8SetPinDirection
| Input: u8 PortId, u8 PinId, and u8 PinDirection
| Output: u8 Error or No Error
| Description: Function to set Pin direction.
|
u8 DIO_u8SetPinDirection (u8 Cpy_u8PortId, u8 Cpy_u8PinId, u8
Cpy_u8PinDirection)

| Name: DIO_u8SetPinValue
| Input: u8 PortId, u8 PinId, and u8 PinValue
| Output: u8 Error or No Error
| Description: Function to set Pin value.
|
u8 DIO_u8SetPinValue (u8 Cpy_u8PortId, u8 Cpy_u8PinId, u8 Cpy_u8PinValue)

| Name: DIO_u8GetPinValue
| Input: u8 PortId, u8 PinId, and Pointer to u8 ReturnedPinValue
| Output: u8 Error or No Error
| Description: Function to get Pin value.
|
u8 DIO_u8GetPinValue (u8 Cpy_u8PortId, u8 Cpy_u8PinId, u8
*Cpy_pu8ReturnedPinValue)

| Name: DIO_u8TogglePinValue
| Input: u8 PortId and u8 PinId
| Output: u8 Error or No Error
| Description: Function to toggle Pin value.
|
u8 DIO_u8TogglePinValue (u8 Cpy_u8PortId, u8 Cpy_u8PinId)
```

### 2.4.2.2. EXI Driver APIs

```
Precompile and Linking Configurations Snippet:

There is no need for Precompile Configurations nor Linking Configurations as
the defined APIs below could change the EXI peripheral Configurations during
the Runtime.

| Name: EXI_u8EnablePIE
| Input: u8 InterruptId and u8 SenseControl
| Output: u8 Error or No Error
| Description: Function to enable and configure Peripheral Interrupt Enable (PIE),
|              by setting relevant bit for each interrupt in GICR register, then
|              configuring Sense Control in MCUCR (case interrupt 0 or 1) or MCUCSR
|              (case interrupt 2) registers.
|
u8 EXI_u8EnablePIE (u8 Cpy_u8InterruptId, u8 Cpy_u8SenseControl)

| Name: EXI_u8DisablePIE
| Input: u8 InterruptId
| Output: u8 Error or No Error
| Description: Function to disable Peripheral Interrupt Enable (PIE), by clearing
|              relevant bits for each interrupt in the GICR register.
|
u8 EXI_u8DisablePIE (u8 Cpy_u8InterruptId)

| Name: EXI_u8SetCallBack
| Input: u8 InterruptId and Pointer to function INTInterruptAction that takes void and
|        returns void
| Output: u8 Error or No Error
| Description: Function to receive an address of a function (in APP Layer) to be
|              called back in ISR function of the passed Interrupt (InterruptId), the
|              address is passed through a pointer to function (INTInterruptAction),
|              and then pass this address to the ISR function.
|
u8 EXI_u8INTSetCallBack (u8 Cpy_u8InterruptId,
void(*Cpy_pfINTInterruptAction)(void))
```

## 2.4.2.3. TMR Driver APIs

```
Precompile Configurations Snippet:

/* TMR0 Waveform Generation Mode Select */
/* Options: TMR_U8_TMR_0_NORMAL_MODE
 *          TMR_U8_TMR_0_PWM_PHASE_CORRECT_MODE
 *          TMR_U8_TMR_0_CTC_MODE
 *          TMR_U8_TMR_0_FAST_PWM_MODE
 */
#define TMR_U8_TMR_0_MODE_SELECT              TMR_U8_TMR_0_NORMAL_MODE

/* TMR0 Compare Match Output Mode Select */
/* Options: TMR_U8_TMR_0_DISCONNECT_OC0_PIN  // Any Mode
 *          TMR_U8_TMR_0_TOG_OC0_PIN         // Non-PWM Modes only
 *          TMR_U8_TMR_0_CLR_OC0_PIN         // Any Mode (PWM -> Non-Inverting Mode)
 *          TMR_U8_TMR_0_SET_OC0_PIN         // Any Mode (PWM -> Inverting Mode)
 */
#define TMR_U8_TMR_0_COMP_OUTPUT_MODE         TMR_U8_TMR_0_DISCONNECT_OC0_PIN

/* TMR0 Interrupt Select */
/* Options: TMR_U8_TMR_0_NO_INTERRUPT
 *          TMR_U8_TMR_0_COMP_INTERRUPT
 *          TMR_U8_TMR_0_OVF_INTERRUPT
 */
#define TMR_U8_TMR_0_INTERRUPT_SELECT         TMR_U8_TMR_0_NO_INTERRUPT

/* TMR0 Clock Select */
/* Options: TMR_U8_TMR_0_NO_CLOCK_SOURCE // No clock source (Timer/Counter0 stopped)
 *          TMR_U8_TMR_0_NO_PRESCALER    // CLK IO/1 (No prescaling)
 *          TMR_U8_TMR_0_8_PRESCALER     // CLK IO/8 (From prescaler)
 *          TMR_U8_TMR_0_64_PRESCALER    // CLK IO/64 (From prescaler)
 *          TMR_U8_TMR_0_256_PRESCALER   // CLK IO/256 (From prescaler)
 *          TMR_U8_TMR_0_1024_PRESCALER  // CLK IO/1024 (From prescaler)
 *          TMR_U8_TMR_0_EXTERNAL_CLOCK_SOURCE_FALL_EDGE
 *          //External clock source on T0 pin. Clock on falling edge.
 *          TMR_U8_TMR_0_EXTERNAL_CLOCK_SOURCE_RISE_EDGE
 *          // External clock source on T0 pin. Clock on rising edge.
 */
#define TMR_U8_TMR_0_CLOCK_SELECT             TMR_U8_TMR_0_NO_CLOCK_SOURCE

/* TMR0 Other Configurations */
#define TMR_U8_TMR_0_PRELOAD_VALUE            0
#define TMR_U8_TMR_0_COMPARE_VALUE            0
#define TMR_U16_TMR_0_NUM_OF_OVERFLOWS        1
```

Linking Configurations Snippet:

There are no Linking Configurations as the defined APIs below could change the DIO peripheral Configurations during the Runtime.

```
| Name: TMR_vdTMR0Initialization
| Input: void
| Output: void
| Description: Function to Initialize TMR0 peripheral.
|
vd TMR_vdTMR0Initialization (void)

| Name: TMR_vdTMR2Initialization
| Input: void
| Output: void
| Description: Function to Initialize TMR2 peripheral.
|
vd TMR_vdTMR2Initialization (void)

| Name: TMR_u8EnableTMR
| Input: u8 TimerId
| Output: u8 Error or No Error
| Description: Function to Enable TMR peripheral.
|
u8 TMR_u8EnableTMR (u8 Cpy_u8TimerId)

| Name: TMR_u8DisableTMR
| Input: u8 TimerId
| Output: u8 Error or No Error
| Description: Function to Disable TMR peripheral.
|
u8 TMR_u8DisableTMR (u8 Cpy_u8TimerId)

| Name: TMR_u8DelayMS
| Input: u8 TimerId, u8 InterruptionMode, and u32 Delay
| Output: u8 Error or No Error
| Description: Function to use TMR peripheral as Delay in MS.
|
u8 TMR_u8DelayMS (u8 Cpy_u8TimerId, u8 Cpy_u8InterruptionMode, u32
Cpy_u32Delay)

| Name: TMR_u8DelayUS
| Input: u8 TimerId, u8 InterruptionMode, and u32 Delay
| Output: u8 Error or No Error
| Description: Function to use TMR peripheral as Delay in US.
|
u8 TMR_u8DelayUS (u8 Cpy_u8TimerId, u8 Cpy_u8InterruptionMode, u32
Cpy_u32Delay)
```

```
| Name: TMR_u8OVFSetCallBack
| Input: u8 TimerId and Pointer to function OVFInterruptAction taking void and
|        returning void
| Output: u8 Error or No Error
| Description: Function to receive an address of a function (in APP Layer) to be
|              called back in ISR function of the passed Timer (TimerId), the address
|              is passed through a pointer to function (OVFInterruptAction), and then
|              pass this address to the ISR function.
|
u8 TMR_u8OVFSetCallBack (u8 Cpy_u8TimerId,
void(*Cpy_pfOVFInterruptAction)(void))

| Name: TMR_u8GetOVFFlagStatus
| Input: u8 TimerId and Pointer to u8 ReturnedFlagStatus
| Output: u8 Error or No Error
| Description: Function to Get status of the OVF Flag in TMR peripheral.
|
u8 TMR_u8GetOVFFlagStatus (u8 Cpy_u8TimerId, u8 *Cpy_pu8FlagStatus)

| Name: TMR_u8ClearOVFFlag
| Input: u8 TimerId
| Output: u8 Error or No Error
| Description: Function to Clear the OVF Flag in TMR peripheral.
|
u8 TMR_u8ClearOVFFlag (u8 Cpy_u8TimerId)
```

## 2.4.3. HAL APIs

### 2.4.3.1. BTN Driver APIs

```
| Name: BTN_u8InitializationNMLMode
| Input: u8 DIOPortId and u8 DIOPinId
| Output: u8 Error or No Error
| Description: Function to initialize BTN pin in NML Mode.
|
u8 BTN_u8InitializationNMLMode (u8 Cpy_u8DIOPortId, u8 Cpy_u8DIOPinId)

| Name: BTN_u8InitializationEXIMode
| Input: u8 EXIId, u8 EXISenseControl, and Pointer to Function that takes
|        void and returns void
| Output: u8 Error or No Error
| Description: Function to initialize BTN pin in EXI Mode.
|
u8 BTN_u8InitializationEXIMode (u8 Cpy_u8EXIId, u8 Cpy_u8EXISenseControl, void
(*Cpy_vpfEXIAction)(void))

| Name: BTN_u8GetBTNState
| Input: u8 DIOPortId, u8 DIOPinId and Pointer to u8 ReturnedBTNState
| Output: u8 Error or No Error
| Description: Function to get BTN state.
|
u8 BTN_u8GetBTNState (u8 Cpy_u8DIOPortId, u8 Cpy_u8DIOPinId, u8
*Cpy_pu8ReturnedBTNState)
```

### 2.4.3.2. LED Driver APIs

```
| Name: LED_u8Initialization
| Input: u8 LedId
| Output: u8 Error or No Error
| Description: Function to initialize LED peripheral.
|
u8 LED_u8Initialization (u8 Cpy_u8LedId)

| Name: LED_u8SetLEDPin
| Input: u8 LedId and u8 Operation
| Output: u8 Error or No Error
| Description: Function to switch LED on, off, or toggle.
|
u8 LED_u8SetLEDPin (u8 Cpy_u8LedId, u8 Cpy_u8Operation)
```

## 2.4.4. MWL APIs

### 2.4.4.1. SOS Driver APIs

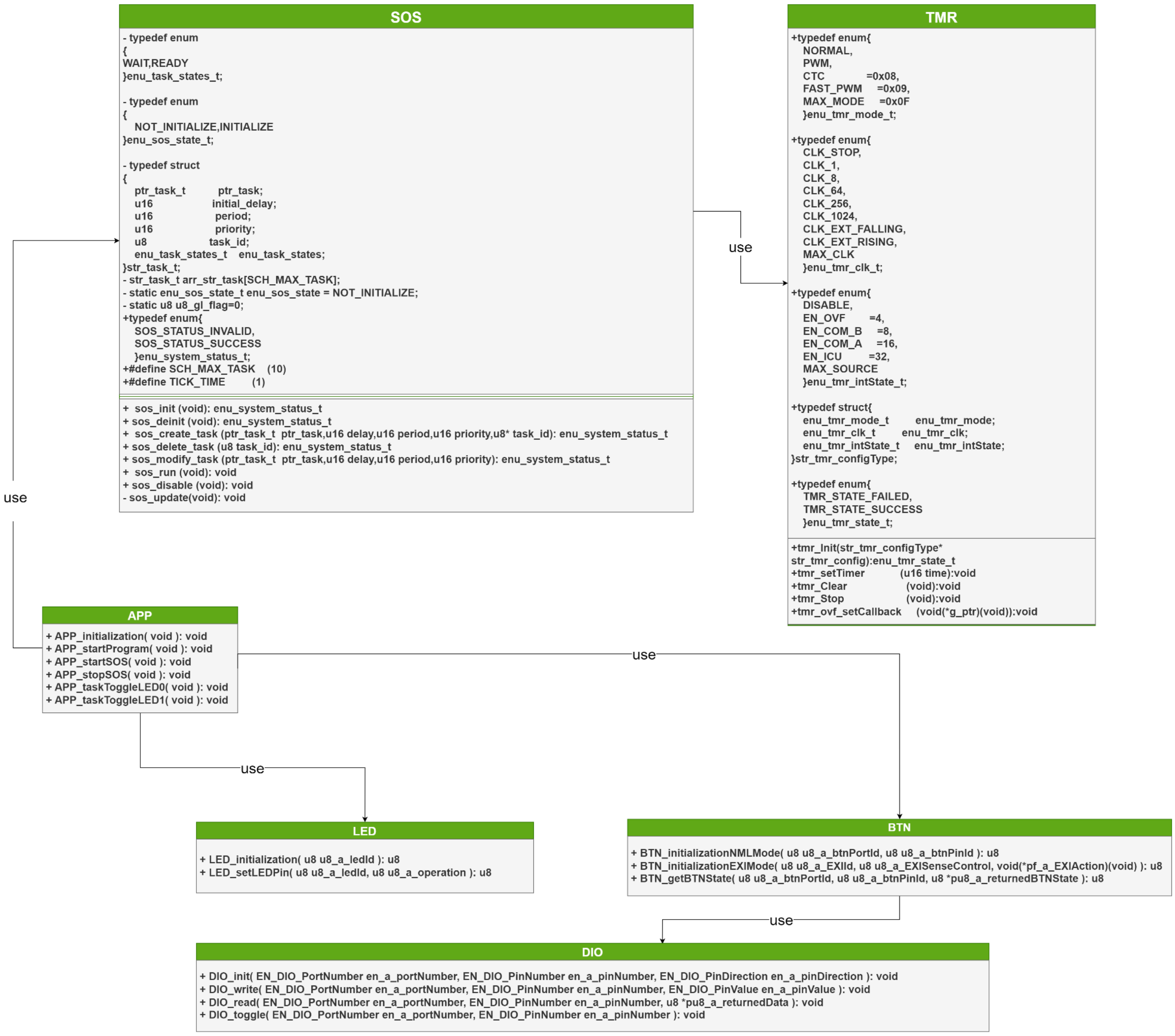
```
| Name: SOS_u8Initialization
| Input: void
| Output: u8 Error or No Error
| Description: Function to initialize SOS database.
|
u8 SOS_u8Initialization (void)

| Name: SOS_u8Deinitialization
| Input: void
| Output: u8 Error or No Error
| Description: Function to reset the SOS database to invalid values.
|
u8 SOS_u8Deinitialization (void)

| Name: SOS_u8CreateTask
| Input: Pointer to en Task
| Output: u8 Error or No Error
| Description: Function to create a new task and add it to the SOS database.
|
u8 SOS_u8CreateTask (SOS_enTask_t *Cpy_penTask)

| Name: SOS_vdEnableScheduler
| Input: void
| Output: void
| Description: Function to enable the scheduler.
|
vd SOS_vdEnableScheduler (void)

| Name: SOS_vdDisableScheduler
| Input: void
| Output: void
| Description: Function to disable the scheduler.
|
vd SOS_vdDisableScheduler (void)
```

## 2.4.5. APP APIs

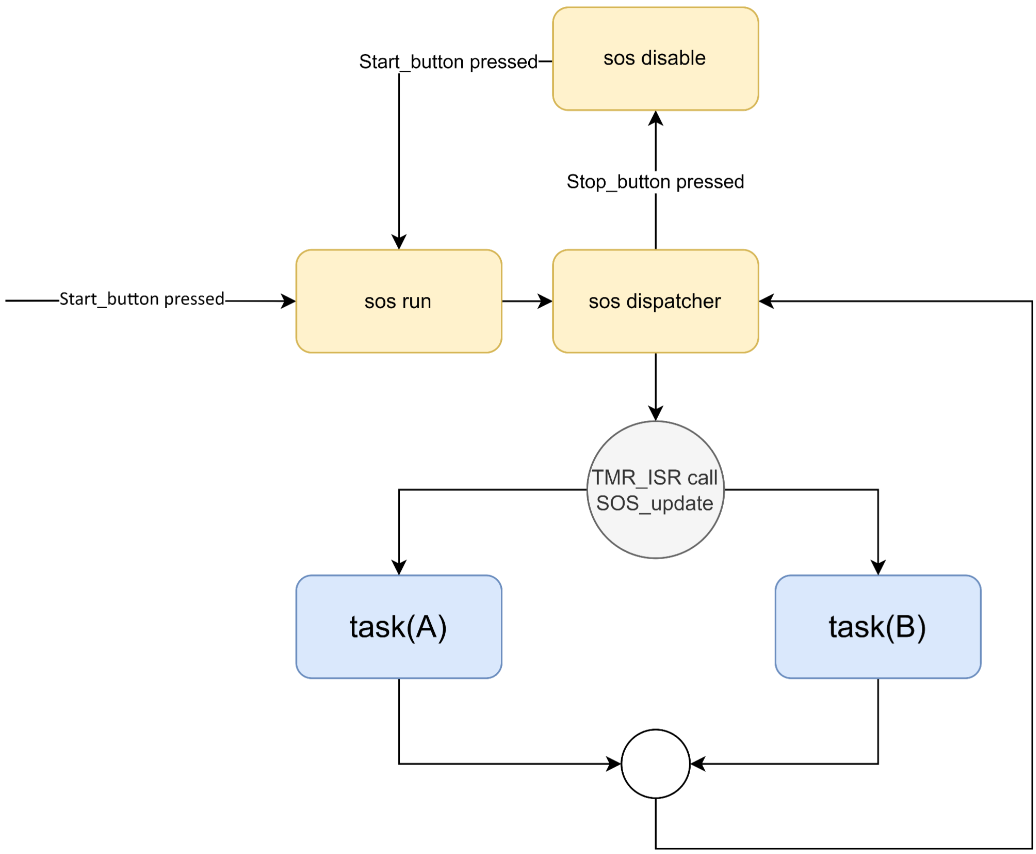
```
| Name: APP_vdInitialization
| Input: void
| Output: void
| Description: Function to Initialize the Application.
|
vd APP_vdInitialization (void)

| Name: APP_vdStartProgram
| Input: void
| Output: void
| Description: Function to Start the basic flow of the Application.
|
vd APP_vdStartProgram (void)

| Name: APP_vdStartSOS
| Input: void
| Output: void
| Description: Function to Start the Operating System.
|
vd APP_vdStartSOS (void)

| Name: APP_vdStopSOS
| Input: void
| Output: void
| Description: Function to Stop the Operating System.
|
vd APP_vdStopSOS (void)

| Name: APP_vdTaskToggleLED0
| Input: void
| Output: void
| Description: Function to implement task 1 logic.
|
vd APP_vdTaskToggleLED0 (void)

| Name: APP_vdTaskToggleLED1
| Input: void
| Output: void
| Description: Function to implement task 2 logic.
|
vd APP_vdTaskToggleLED1 (void)
```

## 2.5. UML

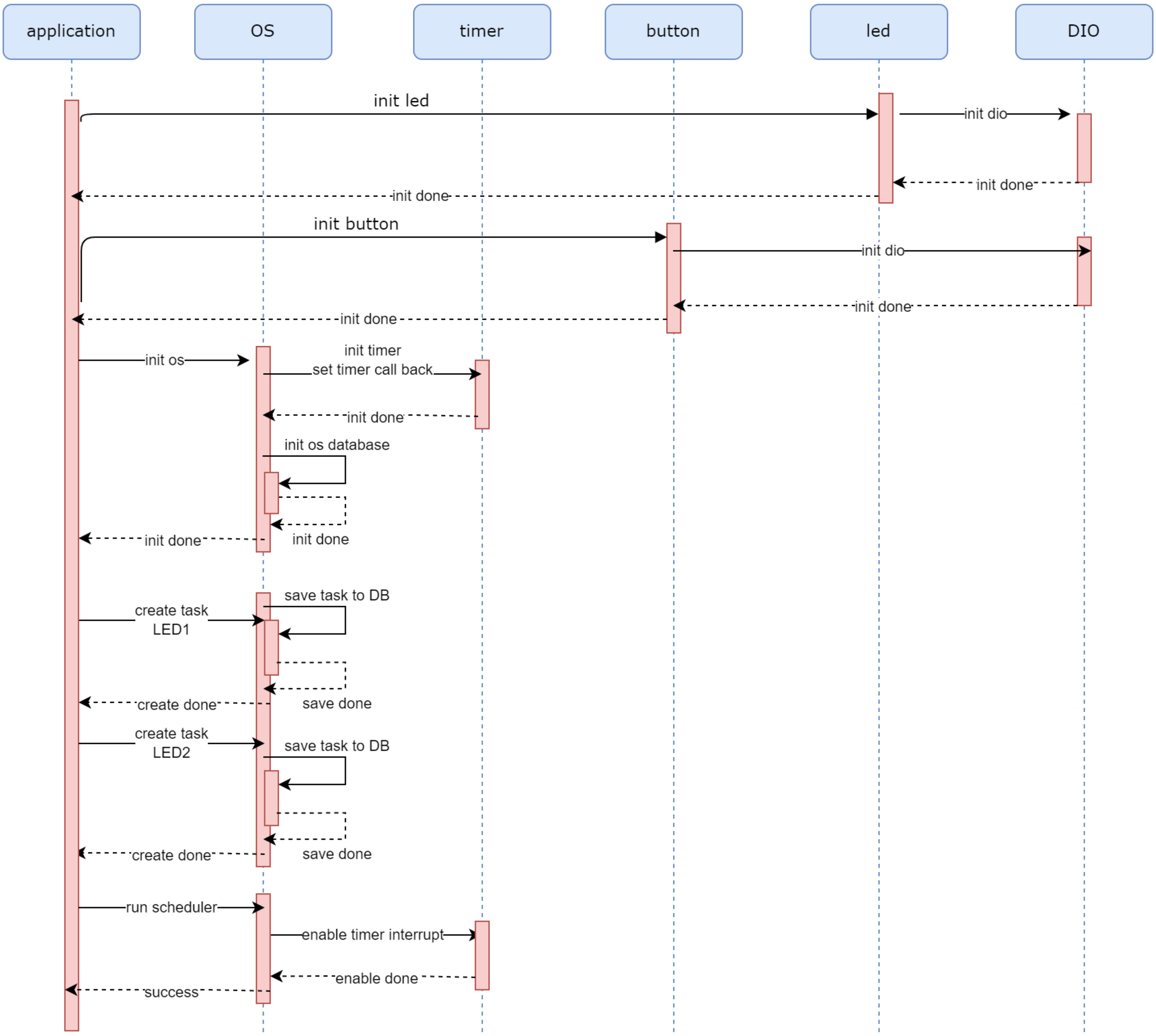
### 2.5.1. Class Diagram



## 2.5.2. State Machine Diagram

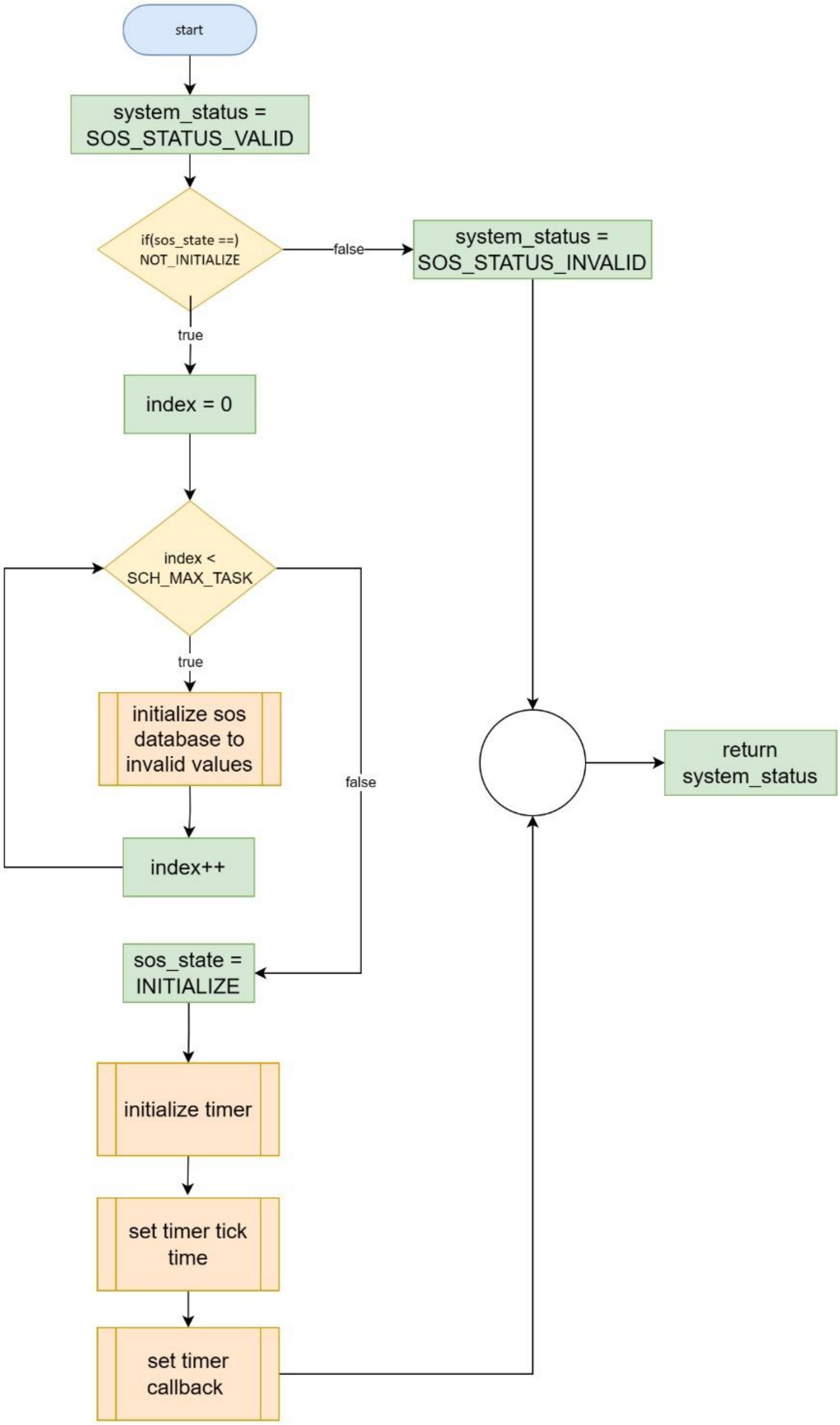


## 2.5.3. Sequence Diagram

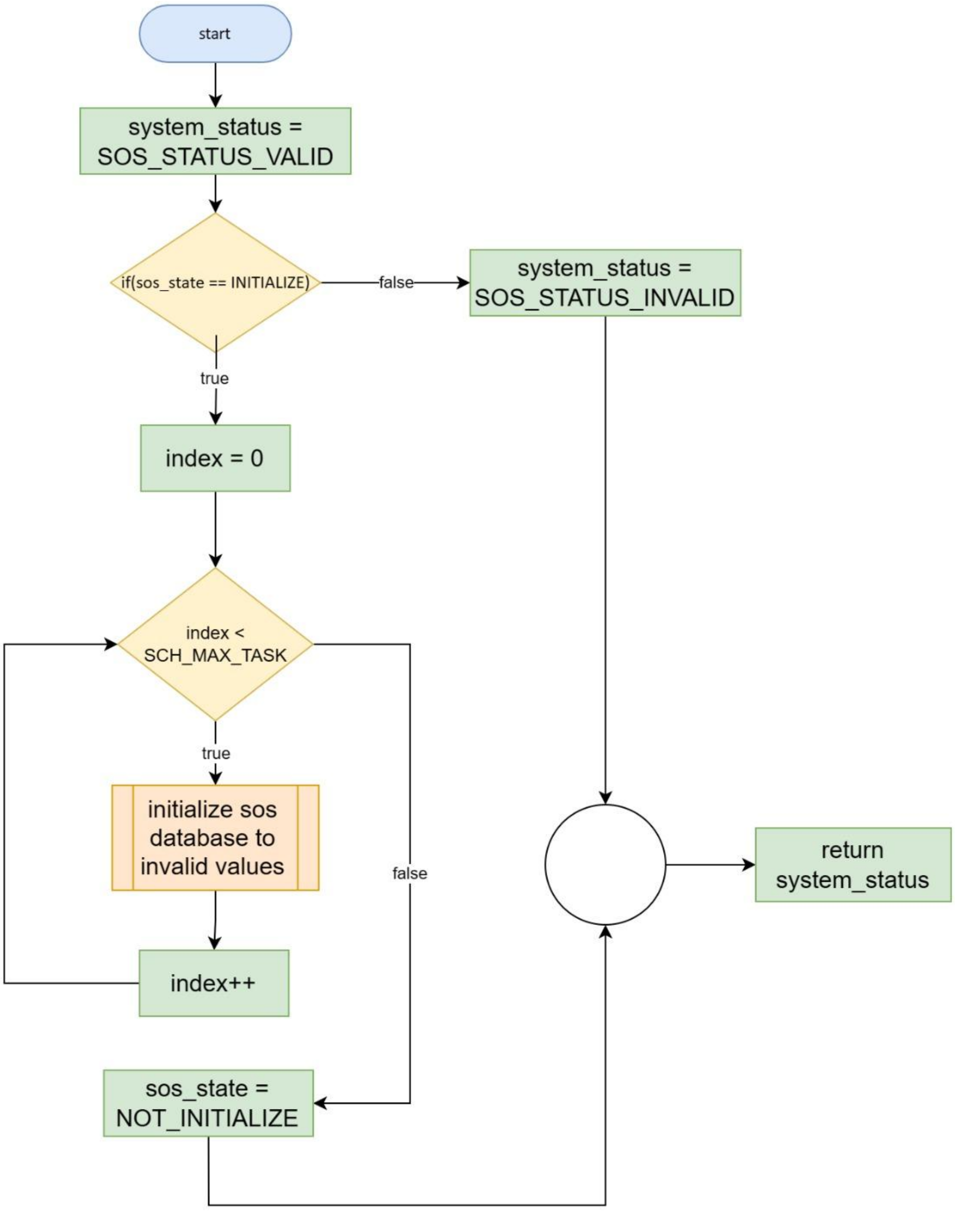


## 2.5.4. Flowchart Diagram

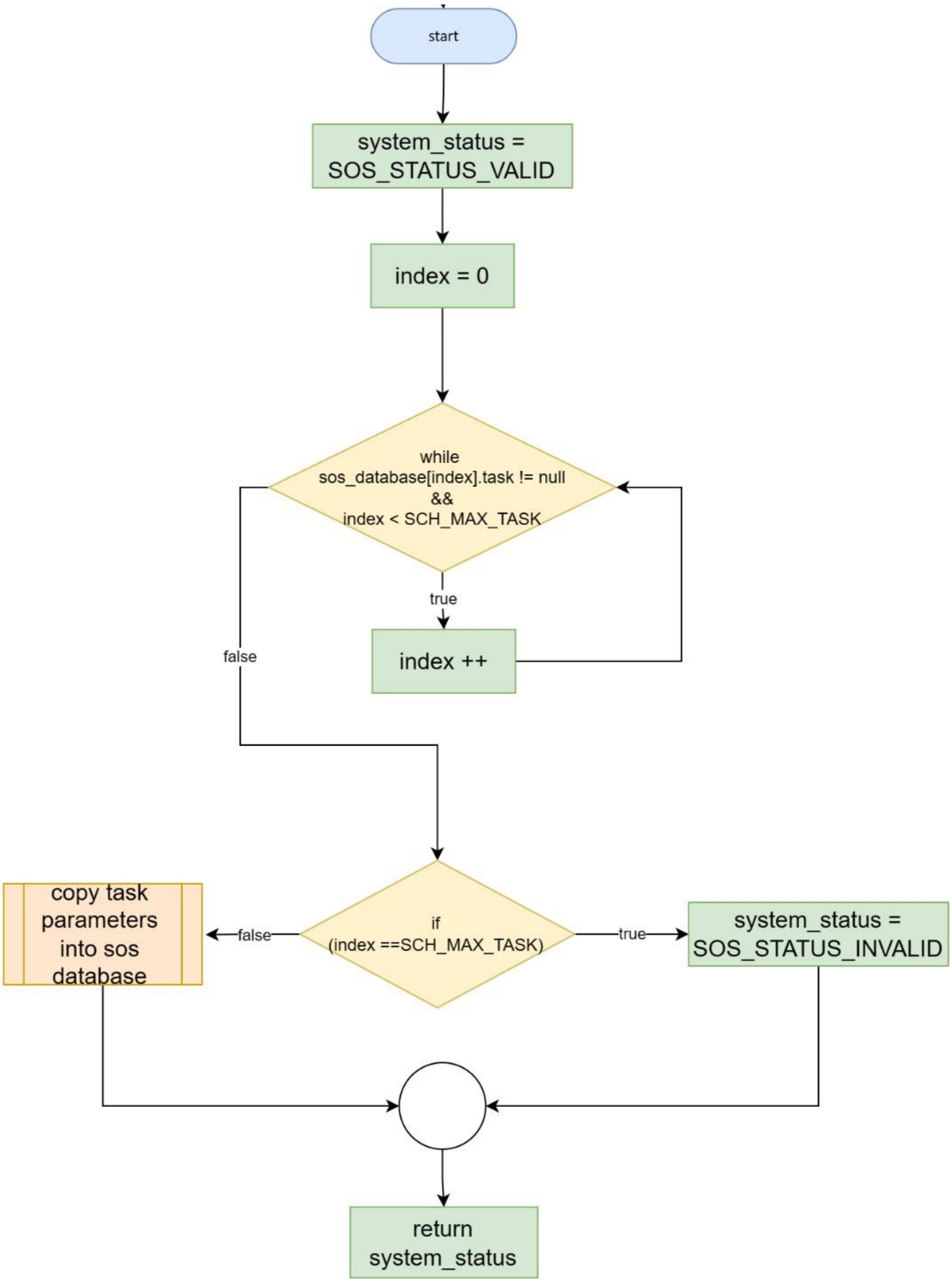
*A. SOS_u8Initialization*



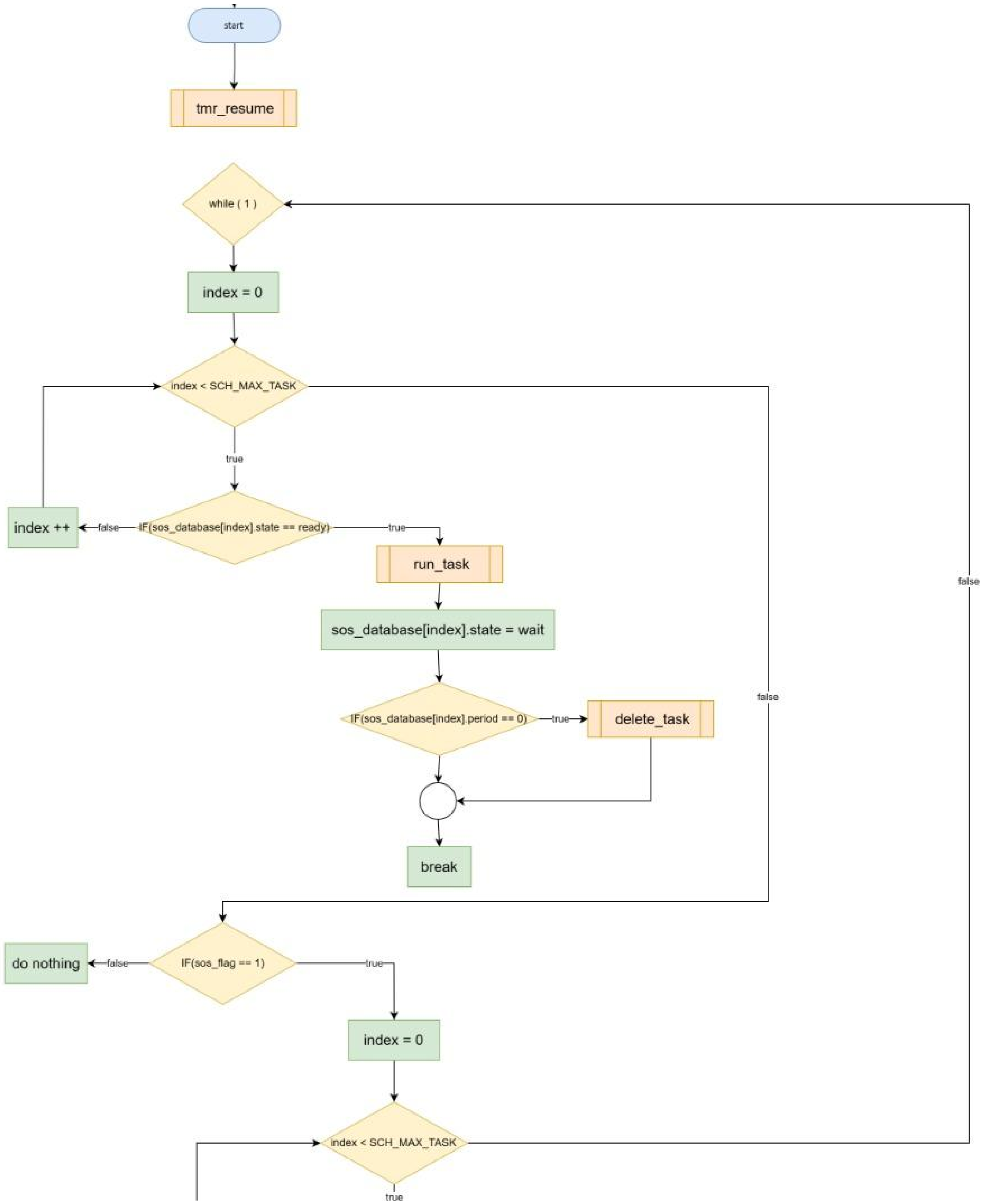
### B. SOS_u8Deintialization



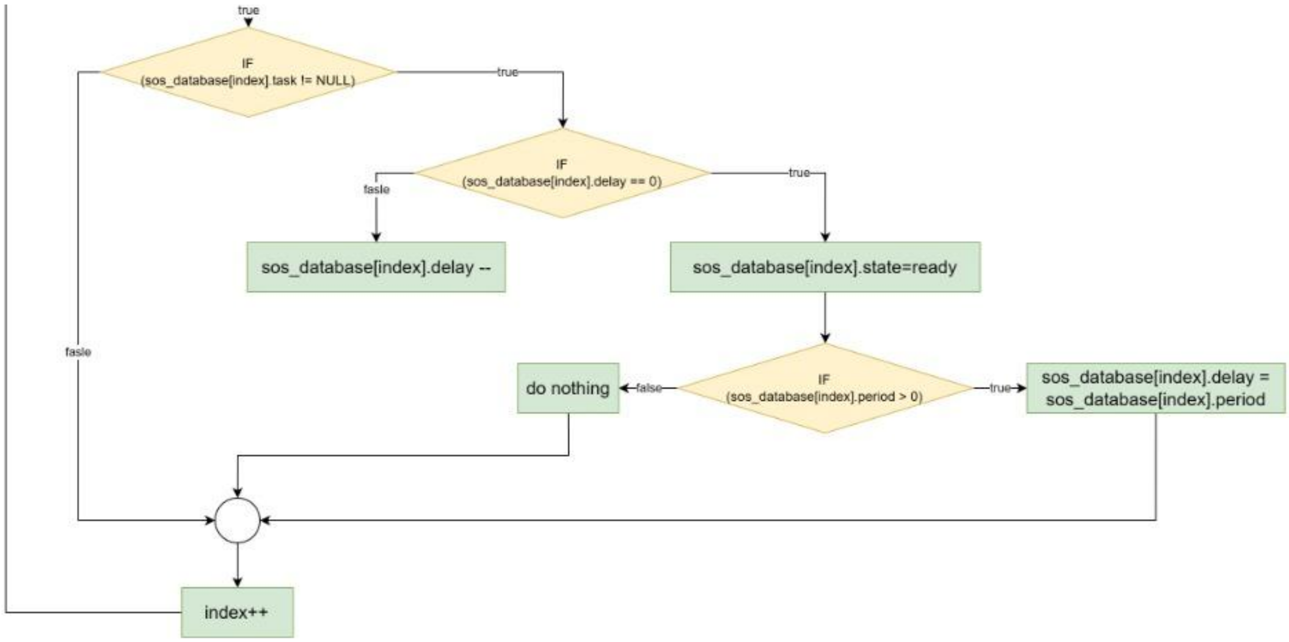
### C. SOS_u8CreateTask



*D. SOS_vdEnableScheduler*




*Continue the flowchart on the next page*


true
IF
(sos_database[index].task != NULL)
true
IF
(sos_database[index].delay == 0)
fasle
true
sos_database[index].delay --
sos_database[index].state=ready
fasle
do nothing
false
IF
(sos_database[index].period > 0)
true
sos_database[index].delay =
sos_database[index].period
index++

## 4. References


1. Draw IO
2. Layered Architecture | Baeldung on Computer Science
3. Block Diagram Maker | Free Block Diagram Online | Lucidchart
4. Microcontroller Abstraction Layer (MCAL) | Renesas
5. Hardware Abstraction Layer - an overview | ScienceDirect Topics
6. Guide to Embedded Systems Architecture - Part 1: Defining middleware - EE Times
7. Top Five Differences Between Layers And Tiers | Skill-Lync
8. What is a module in software, hardware and programming?
9. Embedded Basics – API's vs HAL's
10. Using Push Button Switch with Atmega32 and Atmel Studio