SPRINTS

30 MAY

2023

SOS DESIGN REPORT

SMALL
OPERATING
SYSTEM

Created By :
Sherif Ashraf Khadr

1 - Project Introduction

1.1 - Project Description

A small operating system with a priority-based preemptive scheduler based on a time-triggered approach is designed to efficiently manage tasks in real-time embedded systems. It assigns priorities to tasks and uses fixed time intervals to determine which task to execute next, preempting lower-priority tasks when higher-priority tasks become available. This approach ensures reliable timing and control, making it ideal for applications such as automotive and aerospace systems.

1.2 - Project Components

- 1 Atmega32 MCU
- 2 Led
- 2 Push Buttons

1.3 - Detailed Requirements

1. This OS is using a preemptive priority based scheduler
2. Sos_init function this function will initialize the SOS database

Function Name	sos_init
Syntax	enu_system_status_t sos_init(void);
Sync / Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (out)	None
Parameters (in , out)	None
Return	1 - SOS_STATUS_SUCCES : In case of successful operation 2 - SOS_STATUS_INVALID_STATE : In case the SOS is already Initialized

3. Sos_deinit function this function will reset the SOS database to invalid values

Function Name	sos_deinit
Syntax	enu_system_status_t sos_init(void);
Sync / Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (out)	None
Parameters (in , out)	None
Return	1 - SOS_STATUS_SUCCES : In case of successful operation 2 - SOS_STATUS_INVALID_STATE : In case the SOS is already de-Initialized or was not initialized previously

4. Sos_create_task API this API will create a new task and add it to the SOS database

Function Name	sos_create_task
Syntax	enu_system_status_t sos_create_task (str_task_instance_t *ptr_str_task);
Sync / Async	Asynchronous
Reentrancy	Reentrant
Parameters (in)	str_task_instance_t *ptr_str_task
Parameters (out)	None
Parameters (in , out)	None
Return	1 - TASK_STATUS_CREATE_OK: In case of successful operation 2 - TASK_STATUS_CREATE_NOK : In case of the task not created

5. Sos_delete_task API this API will delete an existing task from the SOS database

Function Name	sos_delete_task
Syntax	enu_system_status_t sos_delete_task(Uchar8_t copy_u8_task_id);
Sync / Async	Asynchronous
Reentrancy	Reentrant
Parameters (in)	Uchar8_t copy_u8_task_id
Parameters (out)	None
Parameters (in , out)	None
Return	1 - TASK_STATUS_DELETED_OK: In case of successful operation 2 - TASK_STATUS_DELETED_NOK : In case of the task not deleted

6. Sos_modify_task API this API will modify existing task parameters in the SOS database

Function Name	sos_modify_task
Syntax	enu_system_status_t sos_modify_task(str_task_instance_t *ptr_str_task);
Sync / Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	str_task_instance_t *ptr_str_task
Parameters (out)	None
Parameters (in , out)	None
Return	1 - TASK_STATUS_MODIFIED_OK: In case of successful operation

	2 - TASK_STATUS_MODIFIED_NOK : In case of the task not modified

7. Sos_run API this API will run the small scheduler

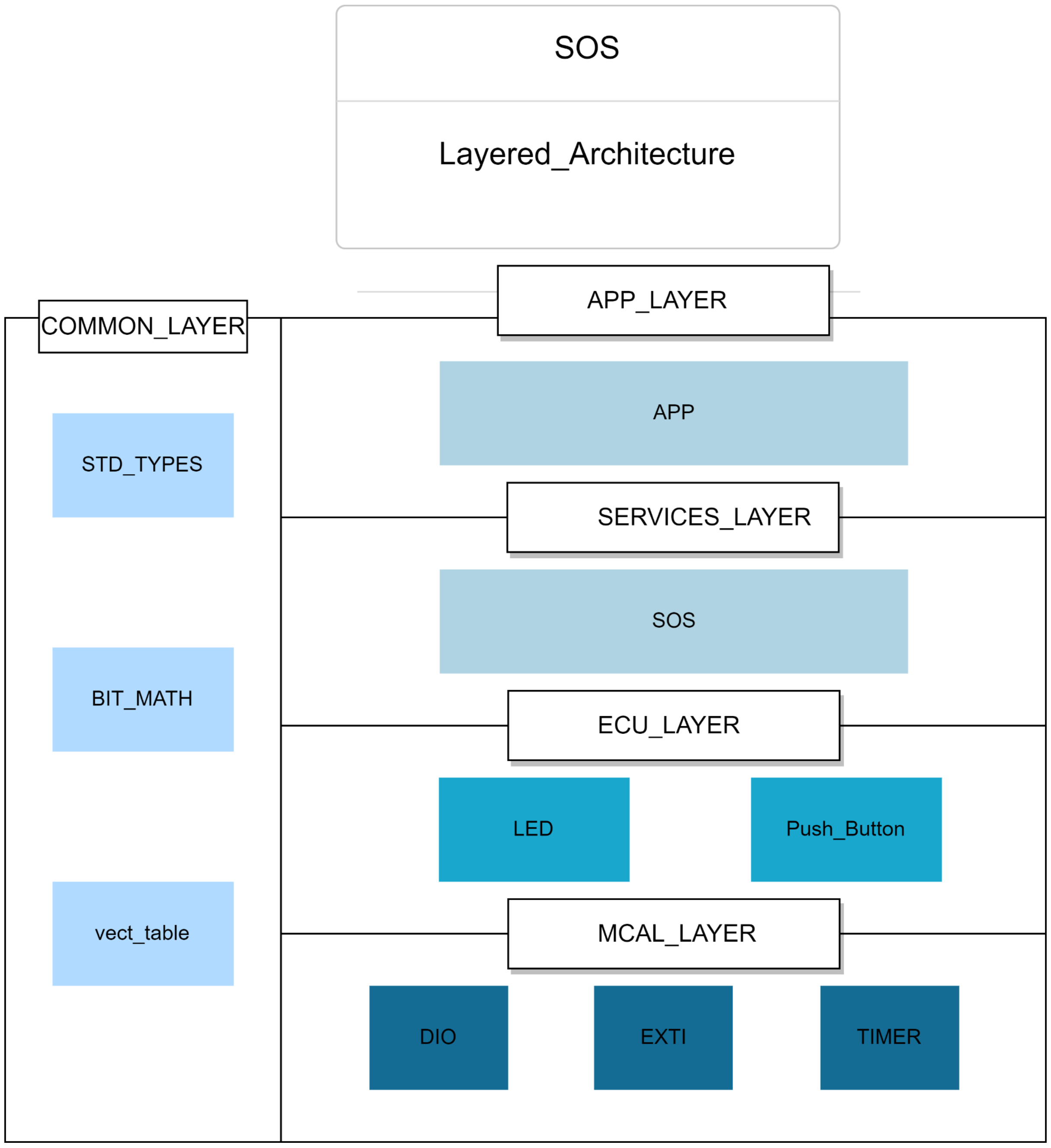
Function Name	sos_run
Syntax	void sos_run(void);
Sync / Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (out)	None
Parameters (in , out)	None
Return	None

8. sos_disable API this API will stop the scheduler

Function Name	sos_disable
Syntax	enu_system_status_t sos_disable(void);
Sync / Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (out)	None
Parameters (in , out)	None
Return	1 - SOS_STATUS_DISABLED_OK: In case of successful operation 2 - SOS_STATUS_DISABLED_NOK : In case of the sos Not Disabled

2 - High Level Design

2.1 - Layered Architecture



2.2 - Modules Description

2.2.1 - DIO Module

A DIO (Digital Input/Output) module is a hardware component that provides digital input and output capabilities to a system. It can be used to interface with digital sensors, switches, and actuators, and typically includes features such as interrupt capability, programmable resistors, overvoltage/current protection, and isolation. Additionally, some DIO modules may include advanced features such as counter/timer functionality or PWM.

2.2.2 - Timer Module

Timers in microcontrollers are hardware components that provide timing and counting capabilities to the system. They can be used for a variety of applications such as generating PWM signals, measuring frequency and pulse width, and input capture. Microcontroller timers can be synchronized and used together to perform more complex timing and counting functions.They often include interrupts and flags that can be used to trigger events.Timers are a powerful tool for precise timing and counting in embedded systems design and development.

2.2.3 - EXTERNAL_INTERRUPT Module

An external interrupt is a signal from outside the ATmega32 that makes it run a special code. The ATmega32 has three pins for external interrupts: INT0 (PD2), INT1 (PD3), and INT2 (PB2). To use an external interrupt, two bits must be turned on: one in EIMSK and one in SREG. The external interrupt can happen when the pin is low, high, or changes, the ATmega32 runs the ISR for that interrupt.

2.2.4 - LED Module

A LED (Light Emitting Diode) module is a hardware component that provides visual output to a system. It can be controlled by software running on a microcontroller and typically includes features such as brightness control, colour selection, and blinking patterns. The LED module is useful for providing status indicators, displaying data, or as a user interface, and can be interfaced with the microcontroller through different protocols such as I2C, SPI, or digital I/O.

2.2.5 - PUSH_BUTTON Module

A push button is a simple switch mechanism that controls some aspect of a machine or a process. It is usually made of plastic or metal and has a flat or shaped surface that can be easily pressed or pushed. A push button can be either momentary or latching, meaning that it returns to its original state when released or stays in the pushed state until pressed again. Push buttons are used for various purposes, such as turning on or off devices, performing calculations and controlling games.

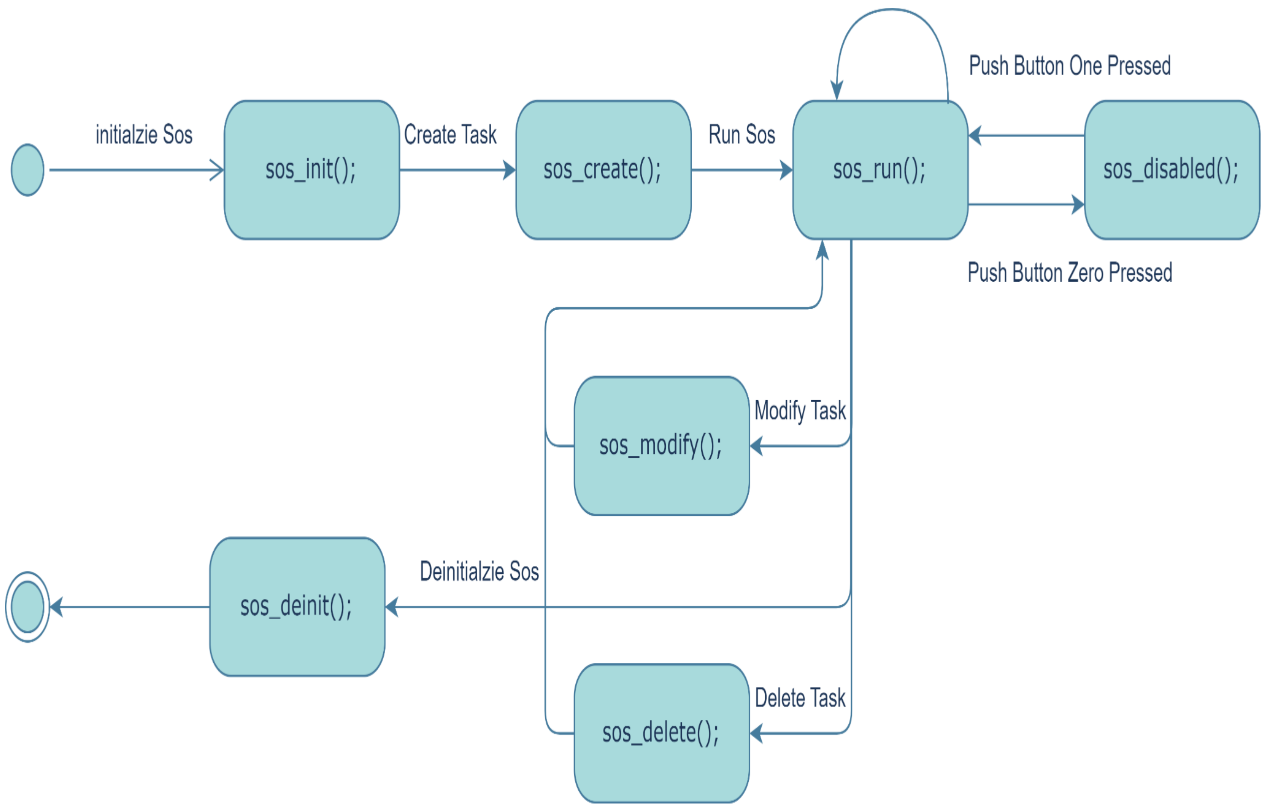
2.2.6 - SOS Module

A small operating system module for an embedded system is a lightweight and efficient software component that manages hardware resources, executes tasks, and provides a platform for application development. It includes a kernel, device drivers, libraries, and APIs, and is designed to meet the specific requirements of the embedded system, such as limited resources and real-time constraints. Its design is optimized for the hardware platform and provides a foundation for managing hardware resources and executing tasks.

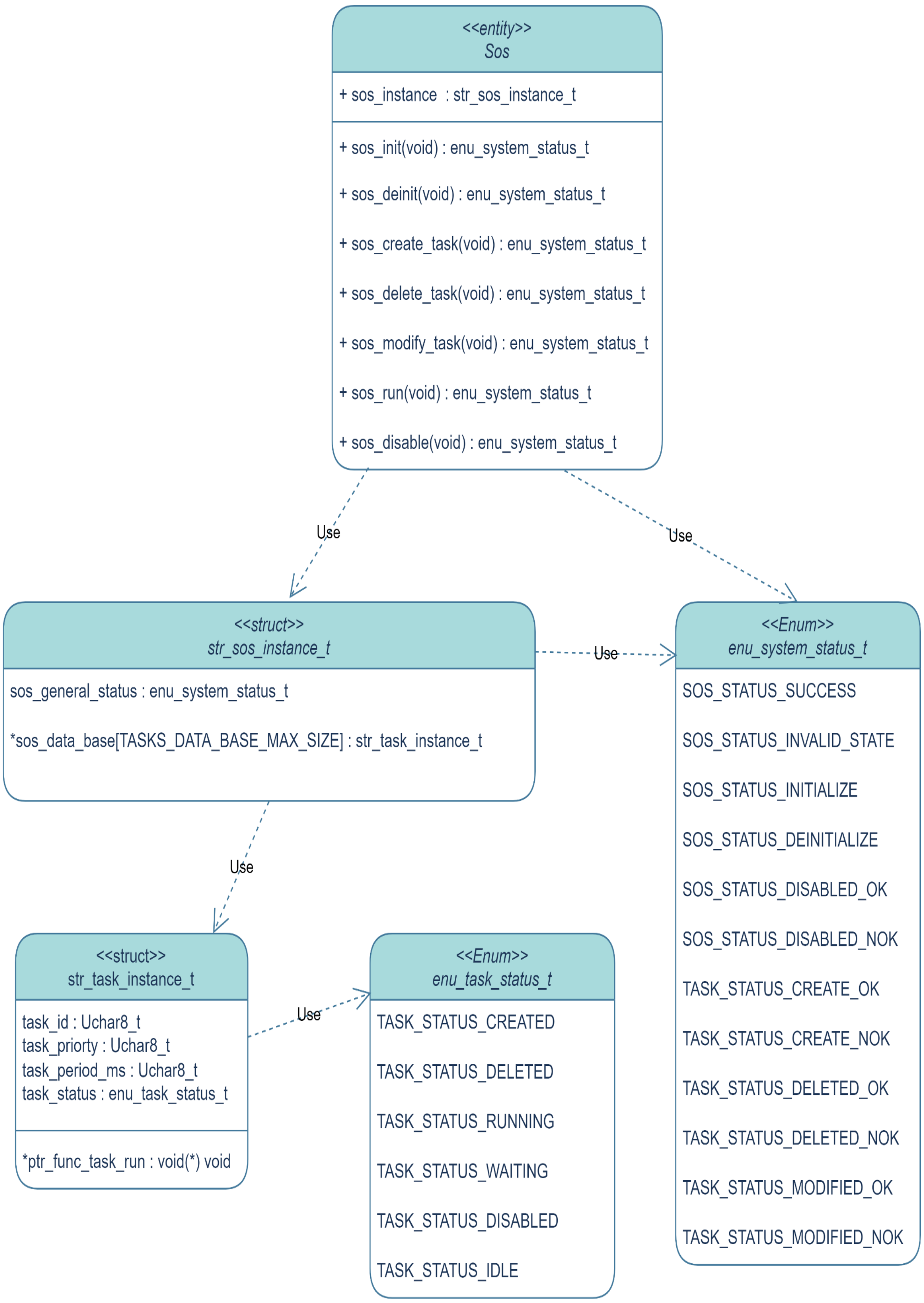
2.2.7 - APP Module

An app module can use ECU driver modules to handle the flow of a specific application within a system. The ECU driver module provides low-level access to the hardware components of the system, such as sensors and actuators. The app module uses these driver modules to interact with the system and perform its specific tasks, resulting in more efficient development and better code organization.

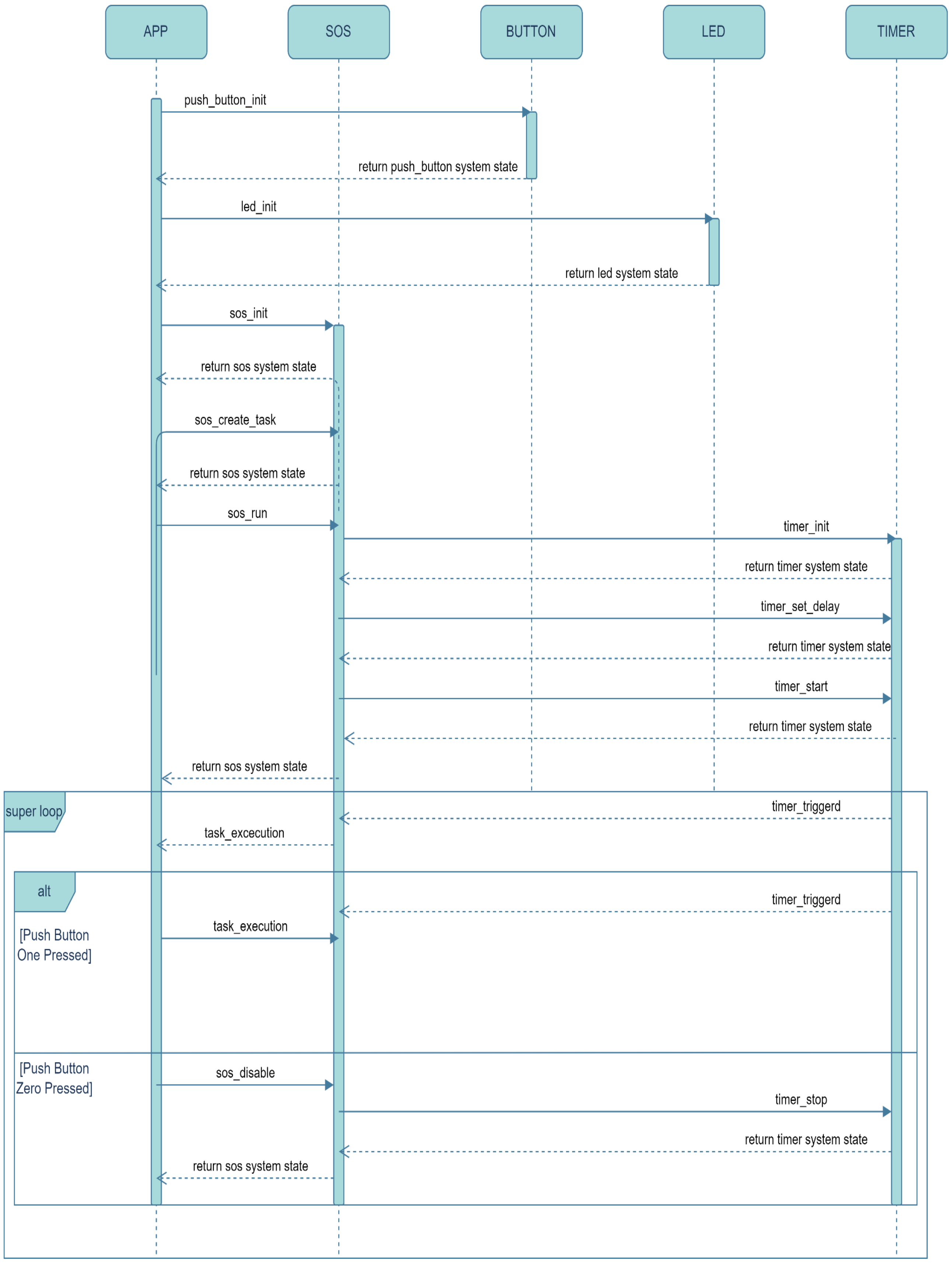
2.3 - SOS State Machine Diagram



2.4 - SOS Class Diagram



2.5 - SOS Sequence Diagram



3 - Link For Diagrams With High Quality