

design & Implement   
A Small OS Design

Youssef Ahmed Abbas Mohamed



Sprints

Contents

[Project Introduction: 1](#_Toc136426801)

[High Level Design: 1](#_Toc136426802)

[Layered architecture: 1](#_Toc136426803)

[Module Description 2](#_Toc136426804)

[Driver Documentations 3](#_Toc136426805)

[LED: 3](#_Toc136426806)

[SOS: 4](#_Toc136426807)

[DIO: 5](#_Toc136426808)

[UML: 8](#_Toc136426809)

[State Machine: 8](#_Toc136426810)

[Sequence Diagram 9](#_Toc136426811)

[Low Level Design: 11](#_Toc136426812)

[Flowchart 11](#_Toc136426813)

# Project Introduction:

The SOS system is a small operating system designed for time-triggered task scheduling on microcontrollers. It provides a priority-based preemptive scheduler that allows the execution of tasks at specified intervals. The system supports task creation, modification, and deletion, as well as the ability to start and stop task execution dynamically.

The SOS system consists of the following main components:

* Main Program: The entry point of the system, responsible for initializing the SOS, configuring peripherals, and managing other program tasks.
* SOS Module: The core module that handles task management, scheduling, and execution.
* Task Functions: User-defined functions that perform specific actions at specified intervals.
* Interrupt Service Routines (ISRs): Interrupt handlers that respond to button presses and control the execution of the SOS.

The system follows a time-triggered approach, where tasks are executed periodically based on their assigned intervals. The SOS module manages the task list, handles task execution, and ensures proper prioritization of tasks.

# High Level Design:

## Layered architecture:

1. Application
2. Manager
3. ECUAL
4. MCAL
5. Microcontroller

## Module Description

1. Application
2. Manager
   1. SOS
3. ECUAL
   1. LED
   2. Button
4. MCAL
   1. Dio
   2. Timer
   3. EINT
5. Microcontroller



## Driver Documentations

### LED:

The module contains functions for initializing the LED, turning it on and off, and toggling its state.

To use this module, the **dio\_interface.h** header file must be included. Additionally, the **str\_dio\_t** structure is used to configure the underlying digital input/output (DIO) pins associated with the LED.

**Dependencies**

* **dio\_interface.h**: This header file defines the functions and data structures related to digital input/output (DIO) operations.

**Data Types**

**enm\_led\_status\_t**

This enumerated type defines the possible states of the LED. It has the following values:

* **LED\_ON**: Represents the LED being turned on (value: 1).
* **LED\_OFF**: Represents the LED being turned off (value: 0).

**str\_led\_t**

This structure represents the LED and its associated properties. It contains the following members:

* **str\_dio**: An instance of the **str\_dio\_t** structure that configures the DIO pins associated with the LED.
* **enm\_led\_status**: The current status of the LED, which can be either **LED\_ON** or **LED\_OFF**.

**Functions**

**void LED\_init(str\_led\_t\* led)**

This function initializes the LED by configuring the DIO pins and setting the initial LED status.

* **led**: A pointer to the **str\_led\_t** structure representing the LED to be initialized.

**void LED\_on(str\_led\_t\* led)**

This function turns the LED on by setting the appropriate DIO pin(s) to the active state.

* **led**: A pointer to the **str\_led\_t** structure representing the LED to be turned on.

**void LED\_off(str\_led\_t\* led)**

This function turns the LED off by setting the appropriate DIO pin(s) to the inactive state.

* **led**: A pointer to the **str\_led\_t** structure representing the LED to be turned off.

**void LED\_toggle(str\_led\_t\* led)**

This function toggles the state of the LED. If the LED is currently on, it will be turned off, and vice versa.

* **led**: A pointer to the **str\_led\_t** structure representing the LED to be toggled.

### SOS:

**sos\_init** function:

* + Initialize the SOS database.
  + Reset any necessary variables or hardware to their default state.
  + Configure the timer for time-triggered scheduling.
  + Enable global interrupts.

**sos\_deinit** function:

* + Disable the SOS.
  + Reset any variables or hardware to their default state.
  + Clear the task list and task count.

**sos\_create\_task** function:

* + Check if the maximum number of tasks has been reached. If yes, return an error.
  + Create a new task with the provided interval, task function, and priority.
  + Add the task to the task list.
  + Increment the task count.
  + Return success.

**sos\_delete\_task** function:

* + Iterate through the task list to find the task with the specified task function.
  + If the task is found, remove it from the task list and adjust the task count.
  + Return success.
  + If the task is not found, return an error.

**sos\_modify\_task** function:

* + Iterate through the task list to find the task with the specified task function.
  + If the task is found, update its interval and priority with the provided values.
  + Return success.
  + If the task is not found, return an error.

**sos\_run** function:

* + Check if the SOS is already running. If yes, return.
  + Set the SOS running flag to true.
  + Enter a loop to execute tasks.
  + Iterate through the task list based on task priority.
  + For each task, increment its counter.
  + If the counter exceeds the task's interval, reset the counter and execute the task's function.
  + Repeat the loop until the SOS running flag is false.

**sos\_disable** function:

* + Set the SOS running flag to false.

### DIO:

**Documentation: DIO (Digital Input/Output) Interface**

**Overview**

DIO (Digital Input/Output) interface, which facilitates controlling and reading digital signals on specific pins and ports. The module includes functions for initializing pins, writing values to pins and ports, reading values from pins and ports, and toggling pin states.

To use this module, the **std\_types.h** and **dio\_private.h** header files must be included. The module defines enums for ports, pin values, pin directions, and DIO errors. It also includes a structure **str\_dio\_t** for representing a DIO pin.

**Dependencies**

* **std\_types.h**: This header file provides standard types used throughout the module.
* **dio\_private.h**: This header file provides private definitions and declarations for the DIO module.

**Enums**

**enm\_dio\_port\_t**

This enumerated type defines the available ports for DIO pins. It has the following values:

* **PORT\_A**: Represents Port A.
* **PORT\_B**: Represents Port B.
* **PORT\_C**: Represents Port C.
* **PORT\_D**: Represents Port D.

**enm\_dio\_value\_t**

This enumerated type defines the possible values for a DIO pin. It has the following values:

* **DIO\_LOW**: Represents a low logic level (value: 0).
* **DIO\_HIGH**: Represents a high logic level.

**enm\_dio\_dir\_t**

This enumerated type defines the possible directions for a DIO pin. It has the following values:

* **DIO\_IN**: Represents the input direction (value: 0).
* **DIO\_OUT**: Represents the output direction.

**enm\_dio\_error\_t**

This enumerated type defines the possible errors that can occur during DIO operations. It has the following values:

* **DIO\_FAIL**: Represents a failure or error (value: 0).
* **DIO\_SUCCESS**: Represents a successful operation.

**Structures**

**str\_dio\_t**

This structure represents a DIO pin and its associated properties. It contains the following members:

* **port**: The port to which the pin belongs (**enm\_dio\_port\_t**).
* **pin**: The number of the pin within the port.

**Functions**

**enm\_dio\_error\_t dio\_init(str\_dio\_t dio\_pin, enm\_dio\_dir\_t dir)**

This function initializes a DIO pin with the specified direction.

* **dio\_pin**: The **str\_dio\_t** structure representing the DIO pin to be initialized.
* **dir**: The desired direction for the pin (**DIO\_IN** or **DIO\_OUT**).

**enm\_dio\_error\_t dio\_write\_pin(str\_dio\_t dio\_pin, enm\_dio\_value\_t value)**

This function writes a value to a DIO pin.

* **dio\_pin**: The **str\_dio\_t** structure representing the DIO pin to be written to.
* **value**: The value to be written to the pin (**DIO\_LOW** or **DIO\_HIGH**).

**enm\_dio\_error\_t dio\_toggle(str\_dio\_t dio\_pin)**

This function toggles the state of a DIO pin. If the pin is currently high, it will be set to low, and vice versa.

* **dio\_pin**: The **str\_dio\_t** structure representing the DIO pin to be toggled.

**enm\_dio\_error\_t dio\_read\_pin(str\_dio\_t dio\_pin, uint8 \*value)**

This function reads the value of a DIO pin and stores it in the provided variable.

* **dio\_pin**: The **str\_dio\_t** structure representing the DIO pin to be read.
* **value**: A pointer to a variable where the pin value will be stored (**DIO\_LOW** or **DIO\_HIGH**).

**enm\_dio\_error\_t dio\_write\_port(enm\_dio\_port\_t port, enm\_dio\_value\_t value)**

This function writes a value to the specified DIO port. The value will be applied to all pins of the port.

* **port**: The port to which the value will be written (**PORT\_A**, **PORT\_B**, **PORT\_C**, or **PORT\_D**).
* **value**: The value to be written to the port (**DIO\_LOW** or **DIO\_HIGH**).

**enm\_dio\_error\_t dio\_read\_port(enm\_dio\_port\_t port, uint8 \*data)**

This function reads the value of a DIO port and stores it in the provided variable. The value represents the combined state of all pins in the port.

* **port**: The port to be read (**PORT\_A**, **PORT\_B**, **PORT\_C**, or **PORT\_D**).
* **data**: A pointer to a variable where the port value will be stored.

## UML:

### State Machine:



The state machine for the application consists of three states:

State: IDLE

Description: The system is idle and waiting for a button press.

Events:

- PBUTTON0\_PRESSED: Transition to the STOPPED state.

- PBUTTON1\_PRESSED: Transition to the RUNNING state.

State: RUNNING

Description: The system is running and executing tasks.

Events:

- PBUTTON0\_PRESSED: Transition to the STOPPED state.

- PBUTTON1\_PRESSED: No action.

State: STOPPED

Description: The system is stopped and not executing tasks.

Events:

- PBUTTON0\_PRESSED: No action.

- PBUTTON1\_PRESSED: Transition to the RUNNING state.

### Sequence Diagram

A picture containing text, screenshot, diagram, parallel

Description automatically generated

# Low Level Design:

## Flowchart



Figure 1 Main Flow Chart

A picture containing text, diagram, screenshot, sketch

Description automatically generated

*A picture containing text, screenshot, diagram, font

Description automatically generated*

*A screenshot of a diagram

Description automatically generated with low confidence*

*A picture containing text, diagram, font, screenshot

Description automatically generated*

*A diagram of a task

Description automatically generated with low confidence*

A picture containing diagram, sketch, drawing, origami

Description automatically generated