Design document

project title: Design a Small OS

Represented by: Hazem Ashraf

Table of contents

Project introduction

- **Description**

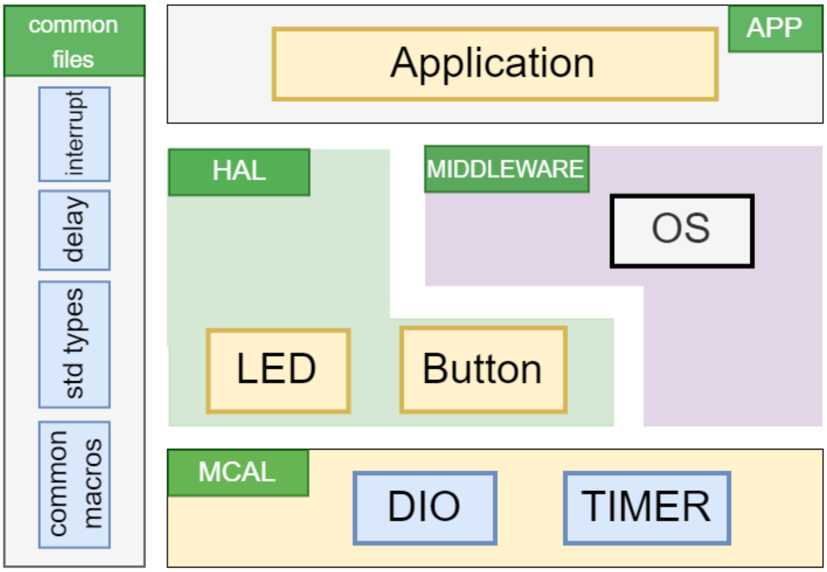
In this Document we discuss the design of a small OS with a priority based on Non preemptive scheduler based on time triggered
Used to executing multiple tasks at different time intervals
By design simple scheduler
a scheduler can be viewed as a simple operating system that allows tasks to be called 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 requires only one function to be altered

The co-operative scheduler operations

- Tasks are scheduled to run at specific times (either on a periodic or one-shot basis)
- When a task is scheduled to run it is added to the waiting list
- When the CPU is free, the next waiting task (if any) is executed
- The task runs to completion, then returns control to the scheduler

High Level Design

- Layered architecture



- **Layer Architecture Description**

  - **Application Layer**

refers to a software layer used for system- and application-specific purposes that is decoupled from the underlying hardware. The application code meets product-specific features and requirements.

  - **Middleware Layer**

refers to the software layer that contains software dependent upon the lower lying hardware drivers but does not directly contain application code. Application code is usually dependent upon the software contained within this middle layer of software. A specific application may have multiple middleware components, such as an RTOS, TCP/IP stack, file system

  - **Hardware abstraction layer (HAL)**

refers to a firmware layer that replaces hardware-level accesses with higher-level function calls.

  - **MCAL**

refers to the software layer that contains low-level, microcontroller-specific software. The driver layer forms the basis from which higher-level software interacts with and controls the microcontroller.

  - **Common Files**

Refers to the layer that contain system utilities and any software that could be used with any layer

High Level Design

- **Modules Description**

  - **APP Layer**

    Contain the implementation of OS calling and the task that will scheduled

  - **OS Layer**

    Contain the files of the OS which responsible for task creation and management

  - **HAL modules**

BUTTON

Used to configure button pin as input and it is used for start and stop OS

LED

Used to configure LED pin as output and it is used to control led toggle

  - **MCAL modules**

GPIO

Used to configure pins directions and read the pin if it is direction is input and write high / low if it is directions is output. Using GPIO for initialize BUTTON ,LED

Timer_1

Use the Timer_1 with the OS to enable schedule the tasks and configure tick time

TIMER_0

Use Timer_0 with to create time delays in the system

High Level Design

- **Common files**

interrupt

Use the configure external interrupts

TIMER_0

Use Timer_0 with to create time delays in the system

- **SOS APIs**

1- sos_init 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
Return	SOS_STATUS_SUCCESS: in case of successful operation SOS_STATUS_INVALID: in case of the SOS is already initialized

2- sos_deinit function will reset the SOS database to invalid values

Function Name	**sos_deinit**
Syntax	**enu_system_status_t** sos_deinit (void);
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in):	None
Parameters (out):	None
Return	SOS_STATUS_SUCCESS: in case of successful operation SOS_STATUS_INVALID: in case of the SOS is already De-initialized or was not initialized previously

- **SOS APIs**

3- sos_create_task function will create new task and add it to the SOS database

Function Name	**sos_create_task**
Syntax	**enu_system_status_t** sos_create_task (ptr_task_t ptr_task ,uint16 delay, uint16 period, uint16 priority, uint8* task_id);
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in):	**ptr_task** : pointer to the function you wish to schedule **Delay** :the delay (in **ticks**) before task is first executed. If set to 0, the task is executed Immediately **Period:** the interval (in **ticks**) between repeated executions of the task. If set to 0, the task is executed only once **Priority:** the task priority if set to 0, the task will be the highest Priority **Task_id**: the task_id is reference to the task index in the database
Parameters (out):	None
Return	SOS_STATUS_SUCCESS: in case of successful operation SOS_STATUS_INVALID: in case of the OS database is full

4- sos_delete_task function will delete existing task from SOS database

Function Name	**sos_delete_task**
Syntax	**enu_system_status_t** sos_delete_task (uint8 task_id);
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in):	**task_id**: the index of the task in OS database
Parameters (out):	None
Return	SOS_STATUS_SUCCESS: in case of successful operation SOS_STATUS_INVALID: in case of the task is not found

- **SOS APIs**

5- sos_modify_task function will modify existing task parameters in the SOS database

Function Name	**sos_modify_task**
Syntax	**enu_system_status_t sos_modify_task** (ptr_task_t ptr_task ,uint16 delay, uint16 period, uint16 priority);
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in):	**ptr_task** : pointer to the function you wish to modify **Delay** :the delay (in **ticks**) before task is first executed. If set to 0, the task is executed Immediately **Period:** the interval (in **ticks**) between repeated executions of the task. If set to 0, the task is executed only once **Priority:** the task priority if set to 0, the task will be the highest Priority
Parameters (out):	None
Return	SOS_STATUS_SUCCESS: in case of successful operation SOS_STATUS_INVALID: in case of the task is not found

6- sos_run function will run the scheduler

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

- **SOS APIs**

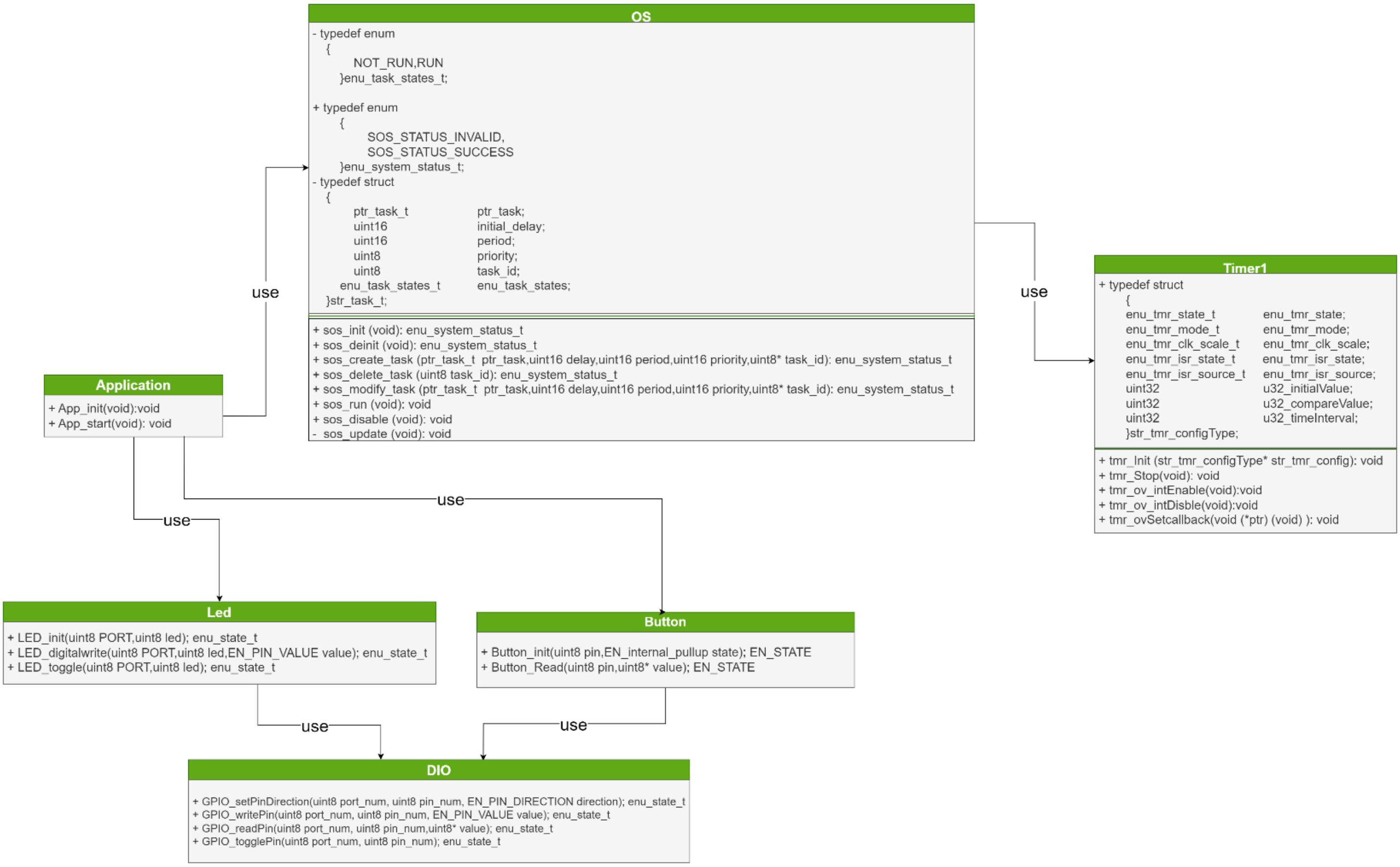
7- sos_disable function will stop the scheduler

Function Name	**sos_disable**
Syntax	**Void sos_disable** (void);
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in):	None
Parameters (out):	None
Return	None

8- sos_update function will update the task to be run

Function Name	**sos_update**
Syntax	static void **sos_update (void)**;
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters (in):	None
Parameters (out):	None
Return	None

Class Diagram



OS

```
- typedef enum
   {
        NOT_RUN,RUN
     }enu_task_states_t;

+ typedef enum
     {
          SOS_STATUS_INVALID,
          SOS_STATUS_SUCCESS
     }enu_system_status_t;
- typedef struct
   {
        ptr_task_t          ptr_task;
        uint16              initial_delay;
        uint16              period;
        uint8               priority;
        uint8               task_id;
     enu_task_states_t      enu_task_states;
   }str_task_t;
```

```
+ sos_init (void): enu_system_status_t
+ sos_deinit (void): enu_system_status_t
+ sos_create_task (ptr_task_t  ptr_task,uint16 delay,uint16 period,uint16 priority,uint8* task_id): enu_system_status_t
+ sos_delete_task (uint8 task_id): enu_system_status_t
+ sos_modify_task (ptr_task_t  ptr_task,uint16 delay,uint16 period,uint16 priority,uint8* task_id): enu_system_status_t
+ sos_run (void): void
+ sos_disable (void): void
-  sos_update (void): void
```

Timer1

```
+ typedef struct
      {
      enu_tmr_state_t          enu_tmr_state;
      enu_tmr_mode_t           enu_tmr_mode;
      enu_tmr_clk_scale_t      enu_tmr_clk_scale;
      enu_tmr_isr_state_t      enu_tmr_isr_state;
      enu_tmr_isr_source_t     enu_tmr_isr_source;
      uint32                   u32_initialValue;
      uint32                   u32_compareValue;
      uint32                   u32_timeInterval;
      }str_tmr_configType;
```

```
+ tmr_Init (str_tmr_configType* str_tmr_config): void
+ tmr_Stop(void): void
+ tmr_ov_intEnable(void):void
+ tmr_ov_intDisble(void):void
+ tmr_ovSetcallback(void (*ptr) (void) ): void
```

Application

```
+ App_init(void):void
+ App_start(void): void
```

Led

```
+ LED_init(uint8 PORT,uint8 led); enu_state_t
+ LED_digitalwrite(uint8 PORT,uint8 led,EN_PIN_VALUE value); enu_state_t
+ LED_toggle(uint8 PORT,uint8 led); enu_state_t
```

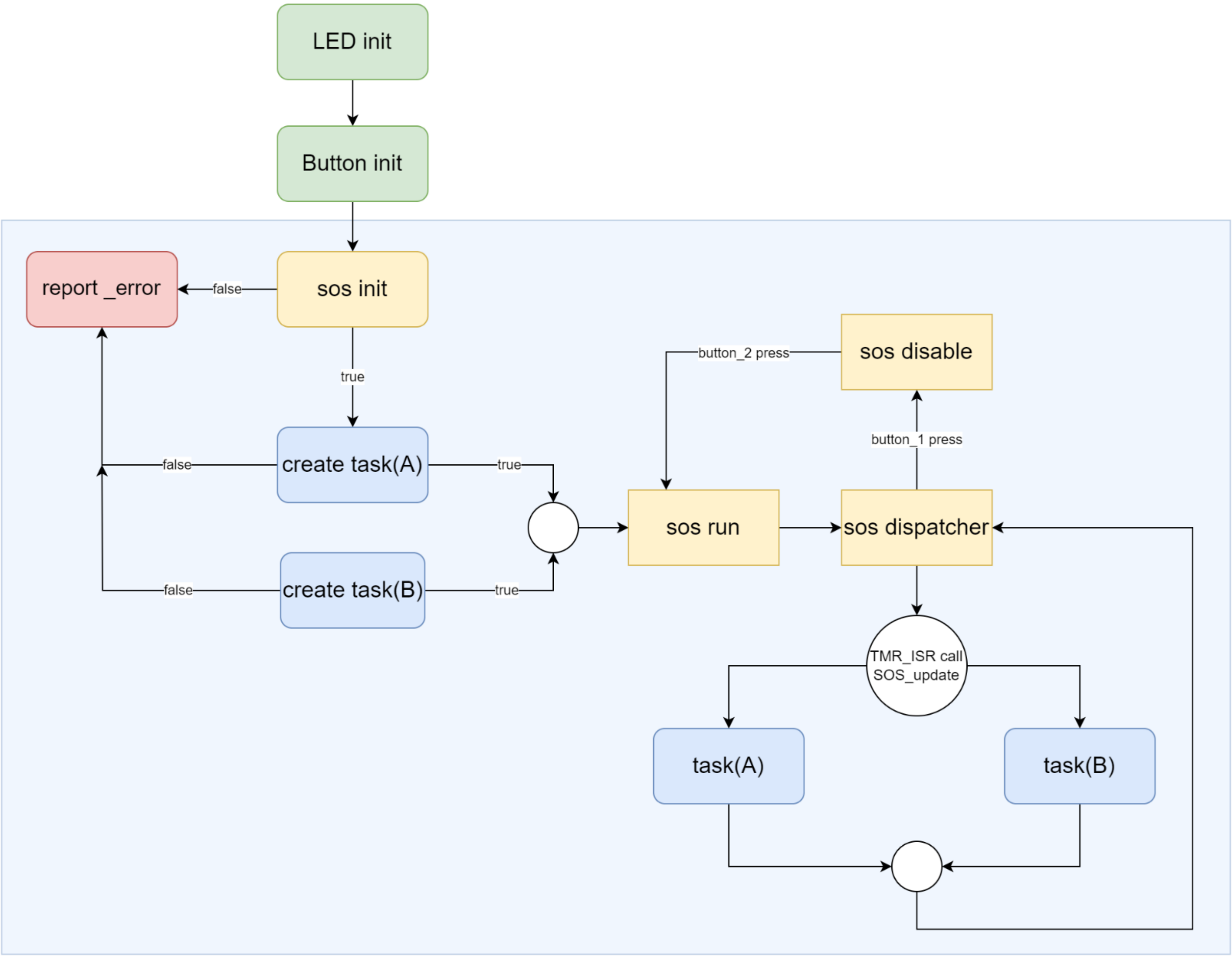
Button

```
+ Button_init(uint8 pin,EN_internal_pullup state); EN_STATE
+ Button_Read(uint8 pin,uint8* value); EN_STATE
```

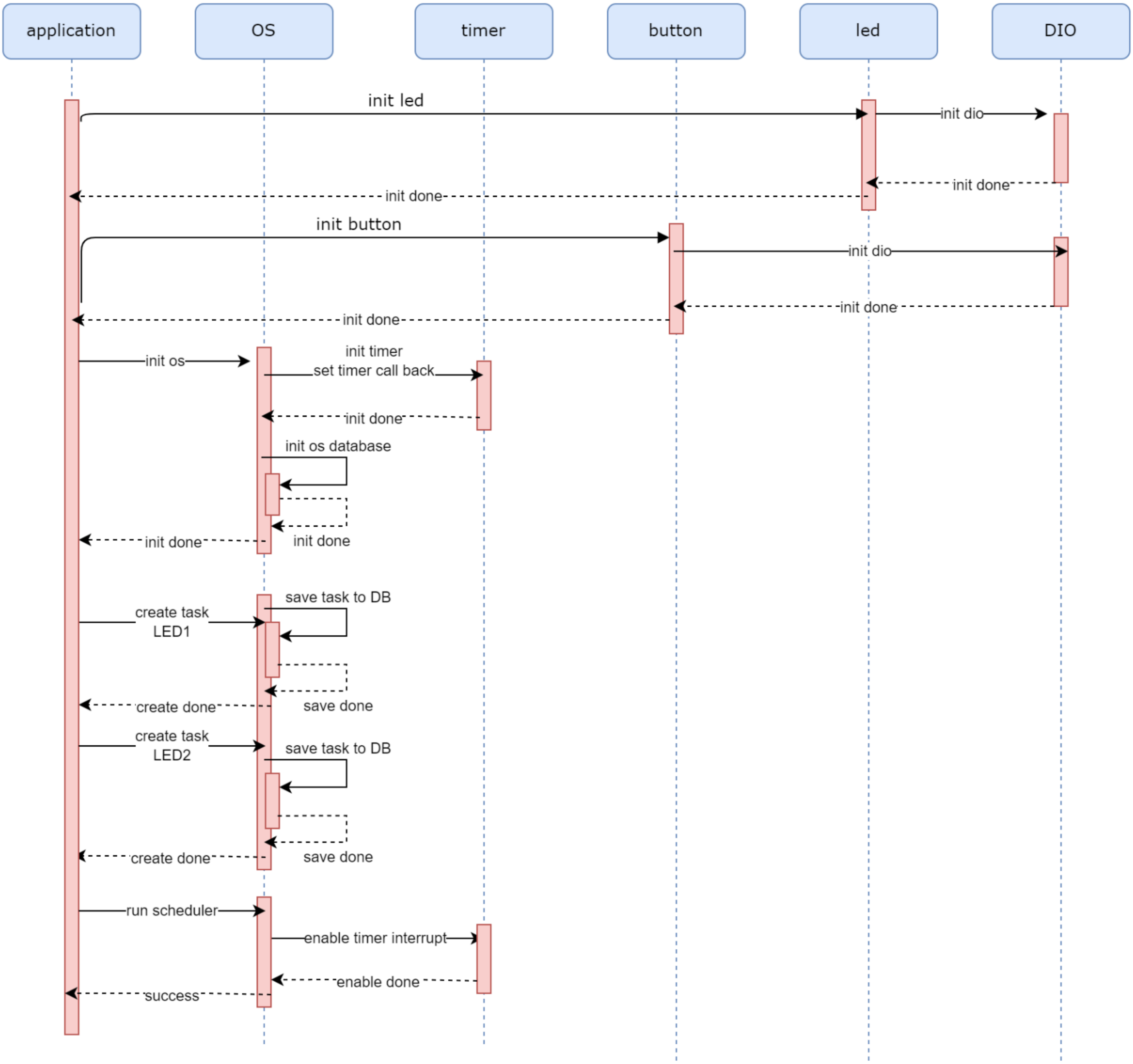
DIO

```
+ GPIO_setPinDirection(uint8 port_num, uint8 pin_num, EN_PIN_DIRECTION direction); enu_state_t
+ GPIO_writePin(uint8 port_num, uint8 pin_num, EN_PIN_VALUE value); enu_state_t
+ GPIO_readPin(uint8 port_num, uint8 pin_num,uint8* value); enu_state_t
+ GPIO_togglePin(uint8 port_num, uint8 pin_num); enu_state_t
```

State Machine



Sequence diagram



SOS interface

OS header file

```
#define SCH_MAX_TASK(10)
#define TICK_TIME(1)

typedef enum{
SOS_STATUS_INVALID,
SOS_STATUS_SUCCESS
}enu_system_status_t;

/*initialize SOS database*/
enu_system_status_t sos_init (void);
/*reset the SOS database to invalid values*/
enu_system_status_t sos_deinit (void);
/*create new task and add it to the SOS database*/
enu_system_status_t sos_create_task (ptr_task_t  ptr_task,
uint16 delay,uint16 period,uint16 priority,uint8* task_id);
/*delete existing task from SOS database*/
enu_system_status_t sos_delete_task (uint8 task_id);
/*modify existing task parameters in the SOS database*/
enu_system_status_t sos_modify_task (ptr_task_t  ptr_task,
uint16 delay,uint16 period,uint16 priority);
/*run the scheduler*/
void sos_run (void);
/*stop the OS*/
void sos_disable (void);
```

OS source file

```
typedef enum{
NOT_RUN,RUN
}enu_task_states_t;

typedef struct
{
ptr_task_t         ptr_task;
Uint16             initial_delay;
Uint16             period;
Uint8              priority;
Uint8              task_id;
enu_task_states_t  enu_task_states;
}str_task_t;

/*OS database*/
str_task_t arr_str_task[SCH_MAX_TASK];
```