Simple OS scheduler design

Prepared by:

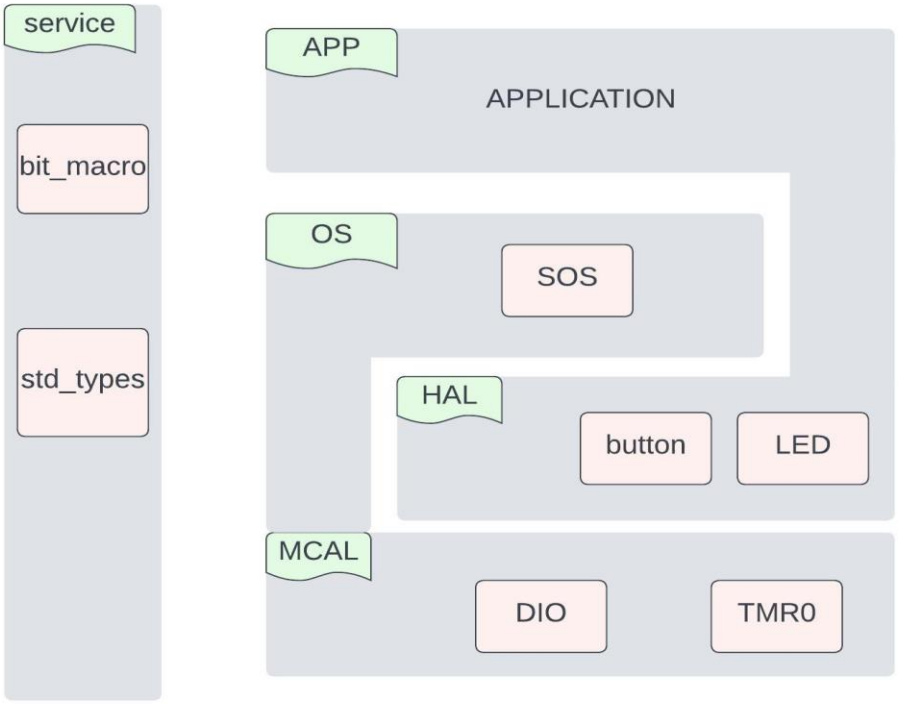
Mustafa Mohammed Abdou

Table of content:

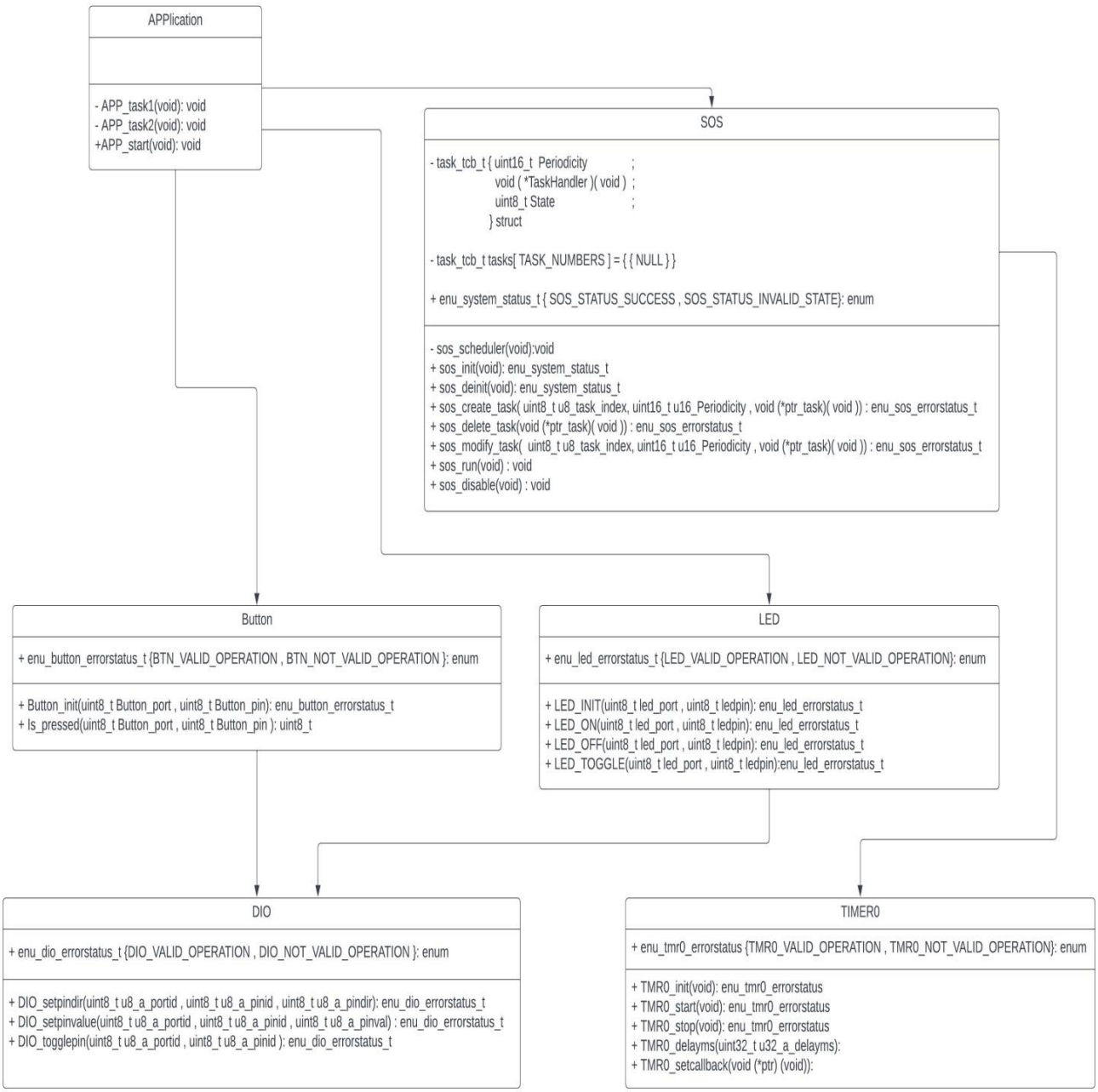
1 – Description

Scheduler is one of the core components inside the OS it's mainly responsible for detecting which task is ready to acquire the resources and start execution, based on an algorithm the scheduler follows some criteria according to pre-emption the scheduler maybe developed as pre-emptive or non pre-emptive also according to priority it can be priority based scheduler or round robin.
In this document we design a simple scheduler based on timer and developed as non pre-emptive but based on priority which means if two tasks intersects the task with higher priority will be executed first.

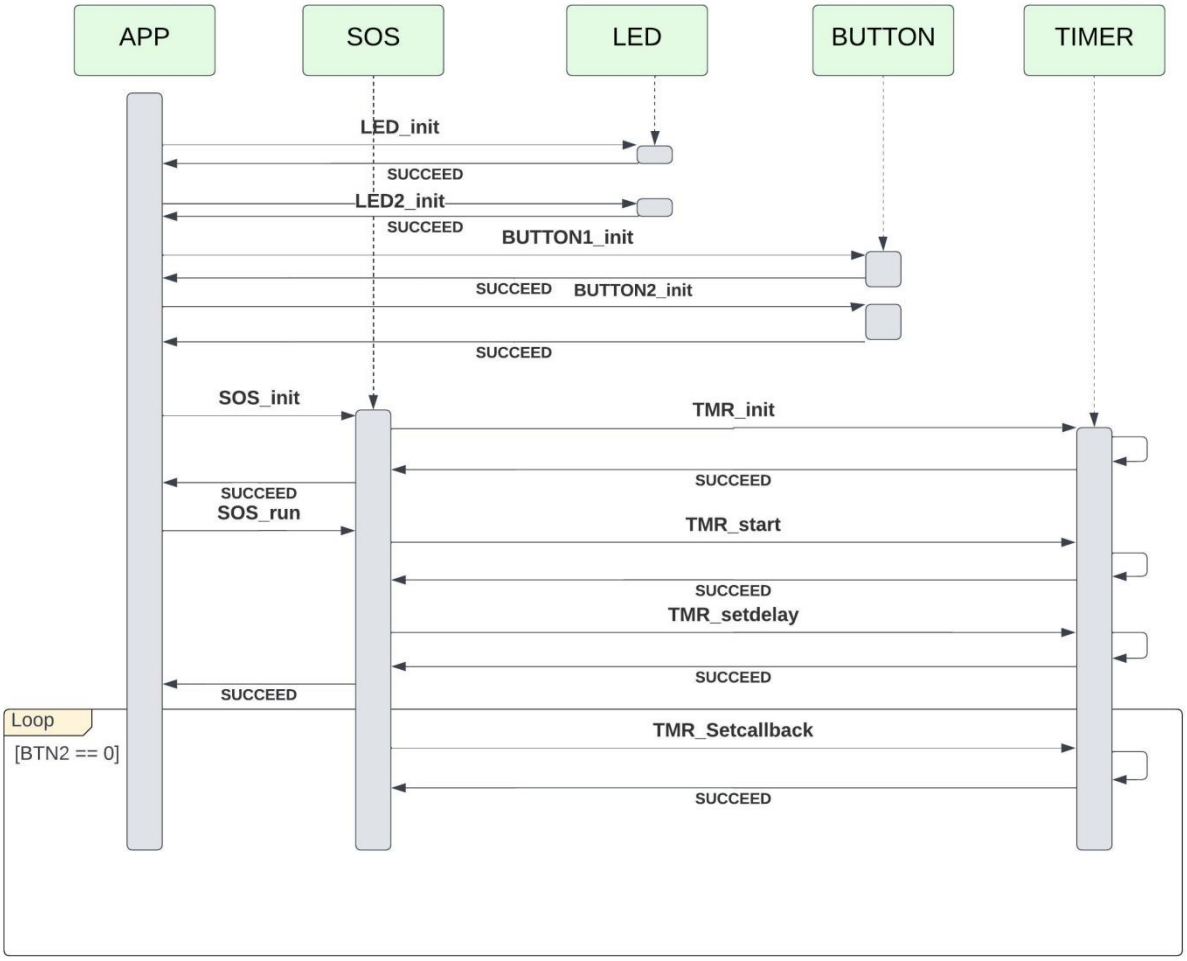
2 – Layered architecture



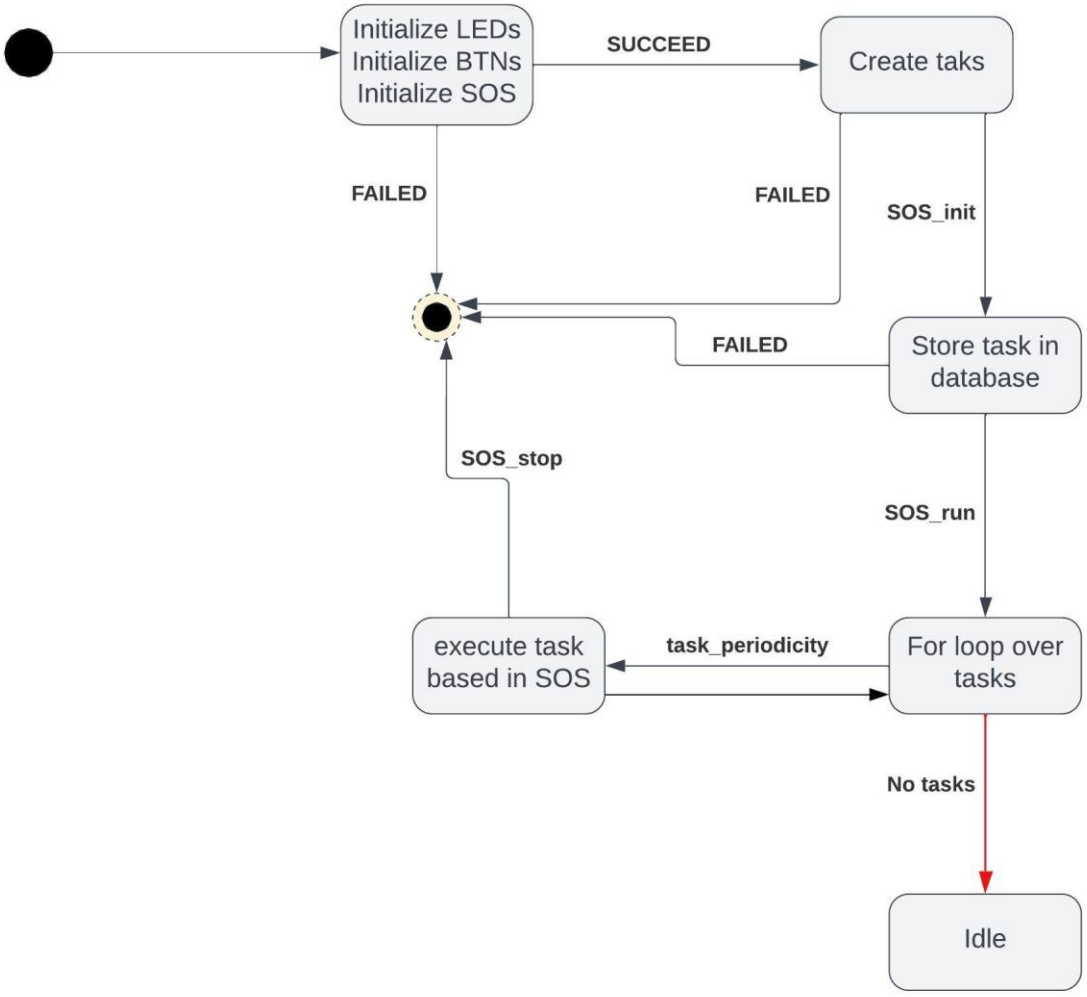
3- SOS class diagram



4- Application sequence diagram



5- SOS state machine



4.1 MCAL APIs

4.1.1 DIO APIs

```
/*******************************************************************/
/* DESCRIBTION  : FUNCTION TO SET THE DIRECTION OF SPECIFIC PIN   */
/* INPUT   : PORT , PINID , DIRECTION                              */
/* RETURNS : PinDirection_t                                        */
/*******************************************************************/
PinDirection_t DIO_setpindir(uint8_t u8_a_portid , uint8_t u8_a_pinid , uint8_t u8_a_pindir);

/*******************************************************************/
/* DESCRIBTION  : FUNCTION TO SET THE DIRECTION OF SPECIFIC PORT   */
/* INPUT   : PORT , DIRECTION                                      */
/* RETURNS : PinDirection_t                                        */
/*******************************************************************/
PinDirection_t DIO_setportdir(uint8_t u8_a_portid , uint8_t u8_a_portdir);


/*******************************************************************/
/* DESCRIBTION  : FUNCTION TO SET THE VALUE OF SPECIFIC PIN       */
/* INPUT   : PORT , PINID , DIRECTION                              */
/* RETURNS : PinValue_t                                            */
/*******************************************************************/
PinValue_t DIO_setpinvalue(uint8_t u8_a_portid , uint8_t u8_a_pinid , uint8_t u8_a_pinval);

/*******************************************************************/
/* DESCRIBTION  : FUNCTION TO SET THE VALUE OF SPECIFIC PORT       */
/* INPUT   : PORT , PINID , DIRECTION                              */
/* RETURNS : PinValue_t                                            */
/*******************************************************************/
PinValue_t DIO_setportvalue(uint8_t u8_a_portid , uint8_t u8_a_portval);


/*******************************************************************/
/* DESCRIBTION  : FUNCTION TO GET THE VALUE OF SPECIFIC PIN       */
/* INPUT   : PORTID , PINID , POINTER TO SET THE VALUE IN IT      */
/* RETURNS : PinRead_t                                             */
/*******************************************************************/
PinRead_t DIO_readpin(uint8_t u8_a_portid , uint8_t u8_a_pinid , uint8_t* u8_a_val);


/*******************************************************************/
/* DESCRIBTION  : FUNCTION TO TOGGLE SPECIFIC PIN                 */
/* INPUT   : PORTID , PINID                                        */
/* RETURNS : PinRead_t                                             */
/*******************************************************************/
PinRead_t DIO_togglepin(uint8_t u8_a_portid , uint8_t u8_a_pinid );
```

4.1.2 Timer0 APIs

```c
typedef enum { VALID_INIT  , NOT_VALID_INIT} TMR0_init_error ;
/*************************************************************/
/** FUNCTION TO INITIALIZE TMR0 WITH SOME CONFIGURATIONS     */
/** ARGUMENTS  : VOID                                        */
/** RETURNS    : TMR0_init                                   */
/*************************************************************/
TMR0_init_error TMR0_init(void);


typedef enum {VALID_START , NOT_VALID_START } TMR0_start_error;
/*****************************************************************************/
/** FUNCTION TO LET TIMER 0 START WORK BY ASSIGN PRESCALLER OR CLOCK SOURCE   */
/** ARGUMENTS  : VOID                                                        */
/** RETURNS    : TMR0_start                                                  */
/*****************************************************************************/
TMR0_start_error TMR0_start(void);



typedef enum {VALID_STOP , NOT_VALID_STOP } TMR0_stop_error;
/*****************************************************************************/
/** FUNCTION TO STOP TIMER 0                                                 */
/** ARGUMENTS  : VOID                                                        */
/** RETURNS    : TMR0_stop                                                   */
/*****************************************************************************/
TMR0_stop_error TMR0_stop(void);


typedef enum {VALID_DELAY , NOT_VALID_DELAY } TMR0_delay_error ;
/*****************************************************************************/
/** FUNCTION TO SET DELAY USING TIMER 0                                      */
/** ARGUMENTS  : TAKES DELAY IN ms                                           */
/** RETURNS    : TMR0_delay                                                  */
/*****************************************************************************/
TMR0_delay_error TMR0_delayms(uint32_t u32_a_delayms);
```

4.1.3 Interrupt APIs

```
/*******************************************************/
/** FUNCTION TO SET THE GLOBAL INTERRUPT ENABLE FLAG    */
/** ARGUMENTS   : VOID                                  */
/** RETURNS     : VOID                                  */
/*******************************************************/
void SET_GLOBALINTERRUPT(void);

/*******************************************************/
/** FUNCTION TO INITIALIZE INT0                         */
/** ARGUMENTS   : VOID                                  */
/** RETURNS     : VOID                                  */
/*******************************************************/
void INT0_init(void);

/*******************************************************/
/** FUNCTION TO INITIALIZE INT1                         */
/** ARGUMENTS   : VOID                                  */
/** RETURNS     : VOID                                  */
/*******************************************************/
void INT1_init(void);

/*******************************************************/
/** FUNCTION TO INITIALIZE INT2                         */
/** ARGUMENTS   : VOID                                  */
/** RETURNS     : VOID                                  */
/*******************************************************/
void INT2_init(void);
```

4.2 HAL APIs

4.2.1 Button APIs

```
/*******************************************************************/
/* FUNCTION TO INITIALIZE THE BUTTON                               */
/* ARGUMENTS : VOID                                                */
/* RETURN    : VOID                                                */
/*******************************************************************/
void Button_init(void);


/*******************************************************************/
/* FUNCTION TO CHECK THE BUTTON STATUS PRESSED OR NOT              */
/* ARGUMENTS : TAKES THE BUTTON PIN                                */
/* RETURN    : RETURNS BUTTON_t type                               */
/*******************************************************************/
en_a_button_t Is_pressed(uint8_t Button_port , uint8_t Button_pin , uint8_t * value);
```

4.2.2 LED APIs

```
/*************************************************/
/** FUNCTION TO INITIALIZE A PIN                **/
/** INPUT : LED PORT , LED PIN                  **/
/** RETURNS : VOID                              **/
/*************************************************/
void LED_INIT(uint8_t led_port , uint8_t ledpin);


/*************************************************/
/** FUNCTION TO SET A LED AS ON                 **/
/** INPUT : LED PORT , LED PIN                  **/
/** RETURNS : VOID                              **/
/*************************************************/
void LED_ON(uint8_t led_port , uint8_t ledpin);

/*************************************************/
/** FUNCTION TO SET A LED AS OFF                **/
/** INPUT : LED PORT , LED PIN                  **/
/** RETURNS : VOID                              **/
/*************************************************/
void LED_OFF(uint8_t led_port , uint8_t ledpin);
```

4.3 SOS APIs

```
void sos_init(void);

void sos_deinit(void);

void sos_create_task(uint8_t task_id , uint16_t u16_Periodicity , void (*ptr_Task)( void ));

void sos_delete_task(void (*ptr_Task)( void ));

void sos_modify_task(uint8_t task_id , uint16_t u16_Periodicity , void (*ptr_Task)( void ));

void sos_run(void);

void sos_disable(void);
```

5-Drivers Description

5.1 DIO Driver

Configuration: Consist of 4 API's
Location: MCAL
Function: used to set pin direction (input or output), pin value (high or low) or read a value from a pin or toggle a pin

5.2 Timer Driver

Configuration: Consist of 5 API's
Location: MCAL
Function: used to set a time delay

5.3 Interrupt Driver

Configuration: Consist of 4 API's
Location: MCAL
Function: used to initialize external interrupts

5.4 Button Driver

Configuration: Consist of 2 API's
Location: HAL
Function: used to initialize a button to specific port and check if this bitton is pressed or not (return button status)

5.4 SOS Driver

Configuration: Consist of 7 API's
Location: Service layer
Function: handles all functionalities related to OS such as creating task , delete task , run the OS and disable OS.