M.Mowafey

Timer-Triggered Operating System

1. Project Introduction

This project involves designing a small operating system (OS) with a priority based preemptive scheduler based on time-triggered.

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.

Service Layer is the topmost layer of Basic Software Architecture. The service layer constitutes an operating system, which runs from the application layer to the microcontroller at the bottom.

2.1.2. Layered Architecture

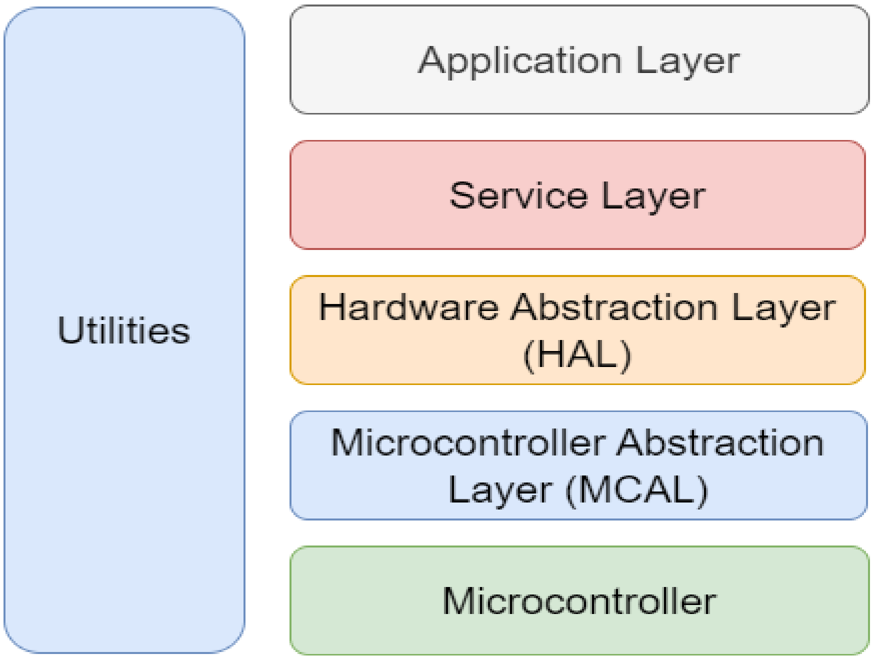


Figure 1. Layered Architecture Design

2.1.3. System Modules

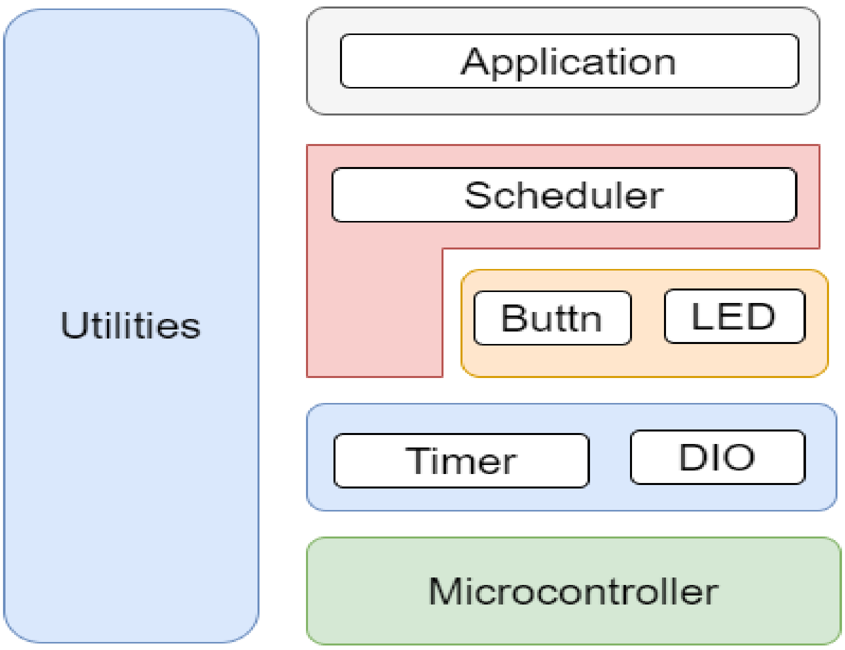


Figure 2. System Module Design

2.2. Modules Description

2.2.1. TIMER Module

The ***TIMER*** module is responsible for generating timing events that are used by other modules in the system. It provides a set of APIs to configure the timer clock source and prescaler, set the timer mode (count up/down), set the timer period, enable/disable timer interrupts, and define an ISR that will be executed when the timer event occurs.

2.2.2. DIO Module

The ***DIO***, or ***Digital Input/Output***, is a simple form of interface used in a wide range of systems to effectively relay digital signals from sensors, transducers and mechanical equipment to other electrical circuits and devices.
Sometimes referred to as General Purpose Input/Output (GPIO), DIO utilizes a logic signal to transfer information.

2.2.3. Button Module

The ***Button*** can be considered the simplest input peripheral that can be connected to a microcontroller. Because of that, usually, every embedded development board is equipped with a button marked as “User Button” and this means it is actually connected to a GPIO pin you can read via software.

2.2.4. Scheduler Module

A ***co-operative scheduler*** provides a simple, highly predictable environment. The scheduler is written entirely in ‘C’ and becomes part of the application: this tends to make the operation of the whole system more transparent and eases development, maintenance and porting to different environments. Memory overheads are seven bytes per task and CPU requirements (which vary with tick interval) are low.

2.3. Drivers’ Documentation (APIs)

2.3.1 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.

2.3.2. MCAL APIs

2.3.2.1. TIMER Driver

.| Initializes timer0 at normal mode
| This function initializes/selects the timer_0 normal mode for the
| timer, and enables the ISR for this timer.
| **Parameters**

```
|                   [in] en_a_interrputEnable value to set the interrupt
|                          bit for timer_0 in the TIMSK reg.
|                   [in] **u8_a_shutdownFlag double pointer, acts as a main switch for
|                             timer0 operations.
| Return
|     An EN_TIMER_ERROR_T value indicating the success or failure of
|                   the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
|                   otherwise)
|
EN_TIMER_ERROR_T                             TIMER_timer0NormalModeInit(EN_TIMER_INTERRPUT_T
en_a_interrputEnable, u8 ** u8_a_shutdownFlag);


| Creates a delay using timer_0 in overflow mode
| This function Creates the desired delay on timer_0 normal mode.
| Parameters
|                   [in] u16_a_interval value to set the desired delay.
| Return
|     An EN_TIMER_ERROR_T value indicating the success or failure of
|                   the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
|                   otherwise)
EN_TIMER_ERROR_T TIMER_delay_ms(u16 u16_a_interval);


| Start the timer by setting the desired prescaler.
|
| This function sets the prescaler for timer_0.
| Parameters
|                   [in] u16_a_prescaler value to set the desired prescaler.
| Return
|     An EN_TIMER_ERROR_T value indicating the success or failure of
|                   the operation
|          (TIMER_OK if the operation succeeded, TIMER_ERROR otherwise)
|
EN_TIMER_ERROR_T TIMER_timer0Start(u16 u16_a_prescaler);


| Stop the timer by setting the prescaler to be 000--> timer is stopped.
|
| This function clears the prescaler for timer_0.
|
| Return
|     void
|

void TIMER_timer0Stop(void);

| Initializes timer2 at normal mode
|
| This function initializes/selects the timer_2 normal mode for the
|  timer, and enables the ISR for this timer.
| Parameters
|                   [in] en_a_interrputEnable value to set
|          the interrupt bit for timer_2 in the TIMSK reg.
|
```

```
| Return
|     An EN_TIMER_ERROR_T value indicating the success or failure of
|                        the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
/                        otherwise)
|
EN_TIMER_ERROR_T TIMER_timer2NormalModeInit(EN_TIMER_INTERRPUT_T);


| Stop the timer by setting the prescaler to be 000--> timer is stopped.
|
| This function clears the prescaler for timer_2.
| Parameters
|                        [in] void.
| Return
|     void
|
void TIMER_timer2Stop(void);


| Start the timer by setting the desired prescaler.
|
| This function sets the prescaler for timer_2.
| Parameters
|                        [in] u16_a_prescaler value to set the desired prescaler.
| Return
|     An EN_TIMER_ERROR_T value indicating the success or failure of
|                        the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
|                        otherwise)
|
 EN_TIMER_ERROR_T TIMER_timer2Start(u16 u16_a_prescaler);


| Creates a timeout delay in msy using timer_2 in overflow mode
|
| This function Creates the desired delay on timer_2 normal mode.
| Parameters
|                        [in] u16_a_interval value to set the desired delay.
| Return
|     An EN_TIMER_ERROR_T value indicating the success or failure of
|                        the operation
|            (TIMER_OK if the operation succeeded, TIMER_ERROR otherwise)
|
EN_TIMER_ERROR_T TIMER_intDelay_ms(u16 u16_a_interval);


| Set callback function for timer overflow interrupt
|
| Parameters
|                        [in] void_a_pfOvfInterruptAction Pointer to the function to be
|                                  called on timer overflow interrupt
| Return
|     EN_TIMER_ERROR_T Returns TIMER_OK if callback function is set
|                             successfully, else returns TIMER_ERROR
|
EN_TIMER_ERROR_T TIMER_ovfSetCallback(void (*void_a_pfOvfInterruptAction)(void));
```

```
| Interrupt Service Routine for Timer2 Overflow.
|      This function is executed when Timer2 Overflows.
|      It increments u16_g_overflow2Ticks counter and checks whether
|      u16_g_overflow2Numbers is greater than u16_g_overflow2Ticks.
|      If true, it resets u16_g_overflow2Ticks and stops Timer2.
|      It then checks whether void_g_pfOvfInterruptAction is not null.
|      If true, it calls the function pointed to by
|     void_g_pfOvfInterruptAction.
|
| Return
|     void
|
ISR(TIMER2_ovfVect);
```

2.3.2.2. DIO Driver

```
| Initializing the desired pin as output/input
|Parameters
|                    [in] uint8_port     the desired port for initializing the pin.
|                    [in] uint8_pin      the desired pin inside the port..
|                    [in] uint8_mode   the desired mode [i/o].
| Return
|     An enu_dio_error_t value indicating the success or failure of
|                    the operation (DIO_OK if the operation succeeded, DIO_ERROR
|          otherwise)
|
enu_dio_error_t DIO_init(uint8_t uint8_port,uint8_t uint8_pin,uint8_t uint8_mode);
```

```
| write the desired digital logic on the pin
|Parameters
|                    [in] uint8_port     the desired port for initializing the pin.
|                    [in] uint8_pin      the desired pin inside the port..
|                    [in] uint8_val      apply the desired logic level..
| Return
|     An enu_dio_error_t value indicating the success or failure of
|                    the operation (DIO_OK if the operation succeeded, DIO_ERROR
|          otherwise)
|
enu_dio_error_t DIO_write(uint8_t uint8_led_port,uint8_t uint8_led_pin,uint8_t uint8_val);
```

```
| Read the applied digital logic on the pin
|Parameters
|                    [in] uint8_port     the desired port for initializing the pin.
|                    [in] uint8_pin      the desired pin inside the port.
| Return
|     An enu_dio_error_t value indicating the success or failure of
|                    the operation (DIO_OK if the operation succeeded, DIO_ERROR
|          otherwise)
enu_dio_error_t DIO_read(uint8_t uint8_led_port, uint8_t uint8_led_pin);
```

2.3.3. HAL APIs

2.3.3.1 LED Driver

| Initializing the desired led_pin as output
|**Parameters**
| [in] uint8_t_led_port the desired port for LED.
| [in] uint8_t_led_pin the desired pin inside the port..
| **Return**
| An enu_led_error_t value indicating the success or failure of
| the operation *(LED_OK if the operation succeeded, LED_ERROR*
| *otherwise)*
|

enu_led_error_t LED_init(uint8_t uint8_led_port, uint8_t uint8_led_pin);

| Turn the LED on
|**Parameters**
| [in] uint8_t_led_port the desired port for LED.
| [in] uint8_t_led_pin the desired pin inside the port..
| **Return**
| An enu_led_error_t value indicating the success or failure of
| the operation *(LED_OK if the operation succeeded, LED_ERROR*
| *otherwise)*
|

enu_led_error_t LED_on(uint8_t uint8_led_port, uint8_t uint8_led_pin);

| Turn the LED off
|**Parameters**
| [in] uint8_t_led_port the desired port for LED.
| [in] uint8_t_led_pin the desired pin inside the port..
| **Return**
| An enu_led_error_t value indicating the success or failure of
| the operation *(LED_OK if the operation succeeded, LED_ERROR*
| *otherwise)*
|

enu_led_error_t LED_off(uint8_t uint8_led_port, uint8_t uint8_led_pin);

| Toggle the LED
|**Parameters**
| [in] uint8_t_led_port the desired port for LED.
| [in] uint8_t_led_pin the desired pin inside the port..
| **Return**
| An enu_led_error_t value indicating the success or failure of
| the operation *(LED_OK if the operation succeeded, LED_ERROR*
| *otherwise)*
|

enu_led_error_t LED_toggle(uint8_t uint8_led_port, uint8_t uint8_led_pin);

2.3.3.2 Button Driver

```
| Initializing the desired pin as input
|Parameters
|                  [in] uint8_port     the desired port for initializing the pin.
|                  [in] uint8_pin      the desired pin inside the port..
| Return
|    An enu_buttn_error_t value indicating the success or failure of
|                  the operation (BUTTN_OK ) if the operation succeeded, BUTTN_ERROR
|         otherwise)
|
enu_buttn_error_t  BUTTN_init(uint8_t uint8_port, uint8_t  uint8_pin);
```

```
| Read the button status
|Parameters
|                  [in] uint8_port     the desired port for initializing the pin.
|                  [in] uint8_pin      the desired pin inside the port.
| Return
|    An enu_buttn_error_t value indicating the success or failure of
|                  the operation (BUTTN_OK  if the operation succeeded, BUTTN_ERROR
|         otherwise)
enu_buttn_error_t BUTTN_status(uint8_t uint8_led_port, uint8_t uint8_led_pin);
```

2.3.4. Service Layer APIs

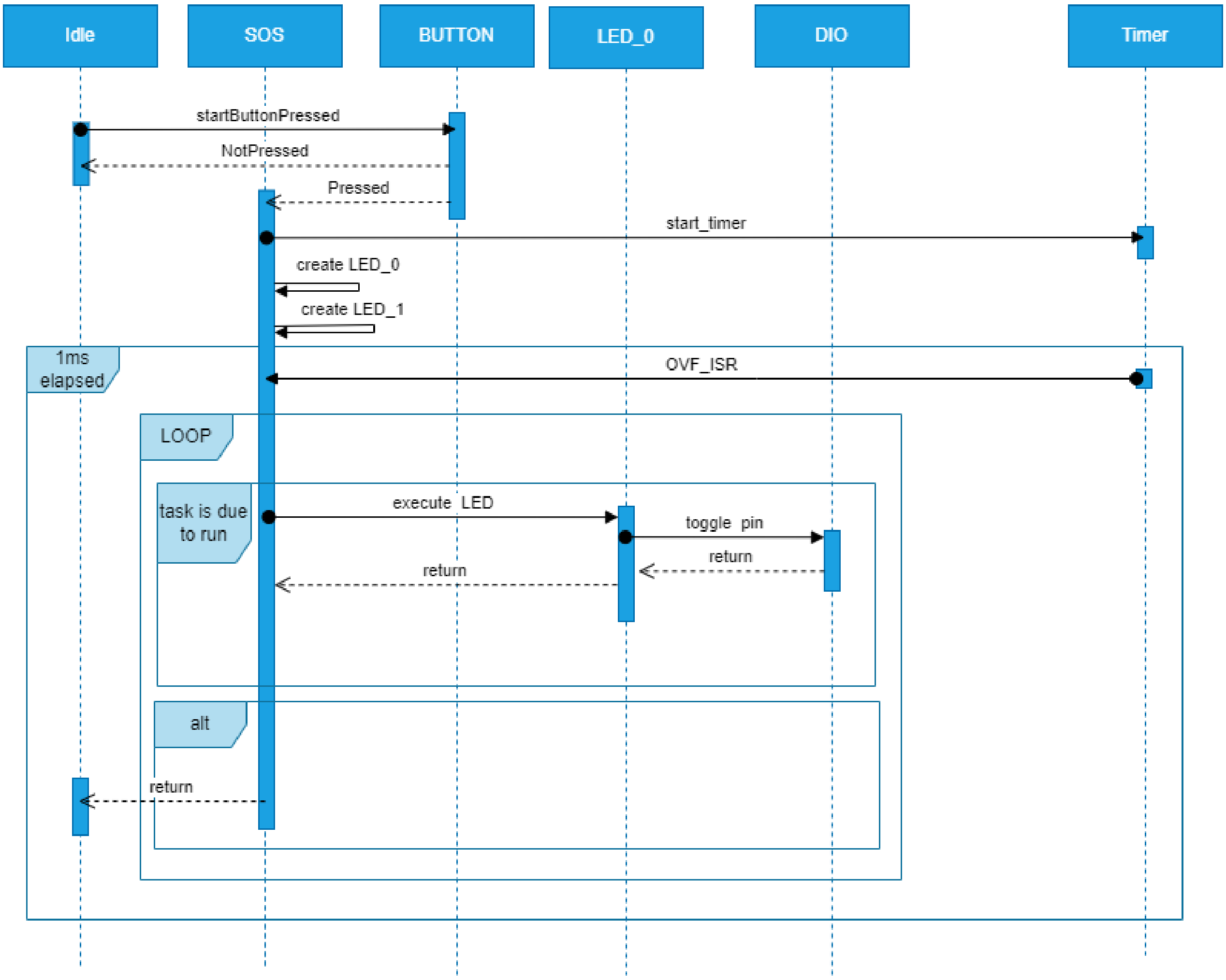
2.3.4.1. Scheduler APIs

```
| When function is due to run This function will run it
| it must be called repeatedly from the main loop
| Parameters
| -                 void
| Return
|            enu_system_status_t. system operation status.
enu_system_status_t SOS_dispatcher (void);
```

```
| Create a task to be executed at regular intervals
| | Parameters
|                  [in] *pTask : pointer to task.
|                  [in] delay : interval before task is executed.
|                  [in] period : periodic time to call this task again.
|                  [in] priority: task priority when intersection is exist.
| Return
|            enu_system_status_t. system operation status.
enu_system_status_t SOS_create_task (void (void * pTask ), const uint32_t uint32_l_delay,
```

```
const uint32_t   uint32_l_period , const uint32_t   uint32_l_priority );


| Remove task from the scheduler
| | Parameters
|                       [in] uint32_l_taskIndex: task index provided by SOS_add_task()
|
| Return
|                       enu_system_status_t. system operation status
enu_system_status_t SOS_delete_Task (const uint32_t uint32_l_taskIndex);


| Enter idle mode
| | Parameters
|                       [in] void.
|
| Return
|                       enu_system_status_t system operation status.
enu_system_status_t SOS_goToSleep (void);


|Deleting all tasks from  the OS
|  Parameters
|                       [in] void.
|
| Return
|                       enu_system_status_t system operation status.
enu_system_status_t SOS_deinit (void);

|Run the SOS by enable the global interrupt bit
|  Parameters
|                       [in] void.
|
| Return
|                       enu_system_status_t system operation status.
enu_system_status_t SOS_run (void);

| Modify the task
|  Parameters
|                       [in] uint32_l_taskIndex: task index provided by SOS_add_task().
|
| Return
|                       enu_system_status_t system operation status.
enu_system_status_t SOS_modify_task (const uint32_t uint32_l_taskIndex);
```

2.3.5. SOS Sequence Diagram

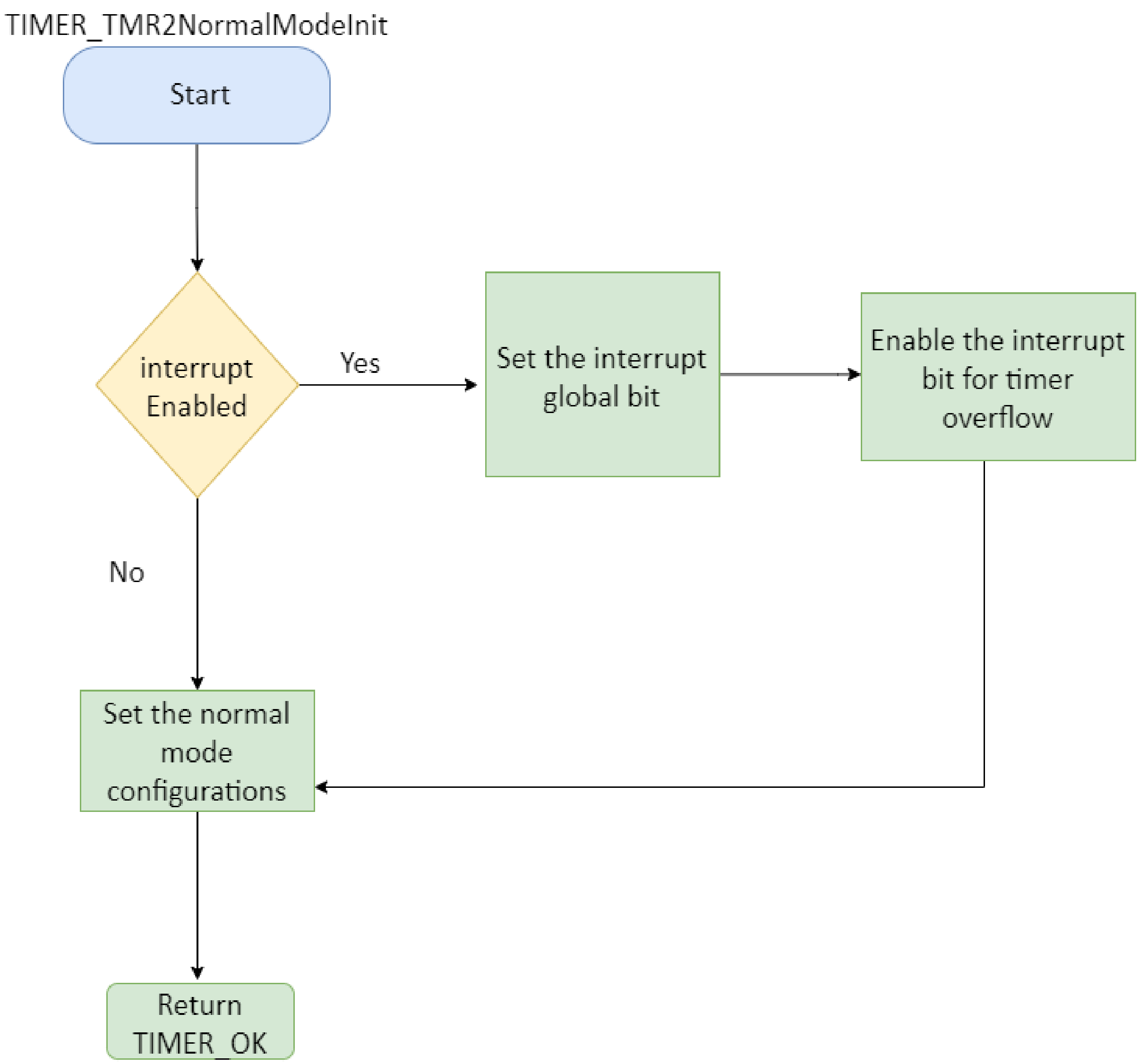


3.Low Level Design

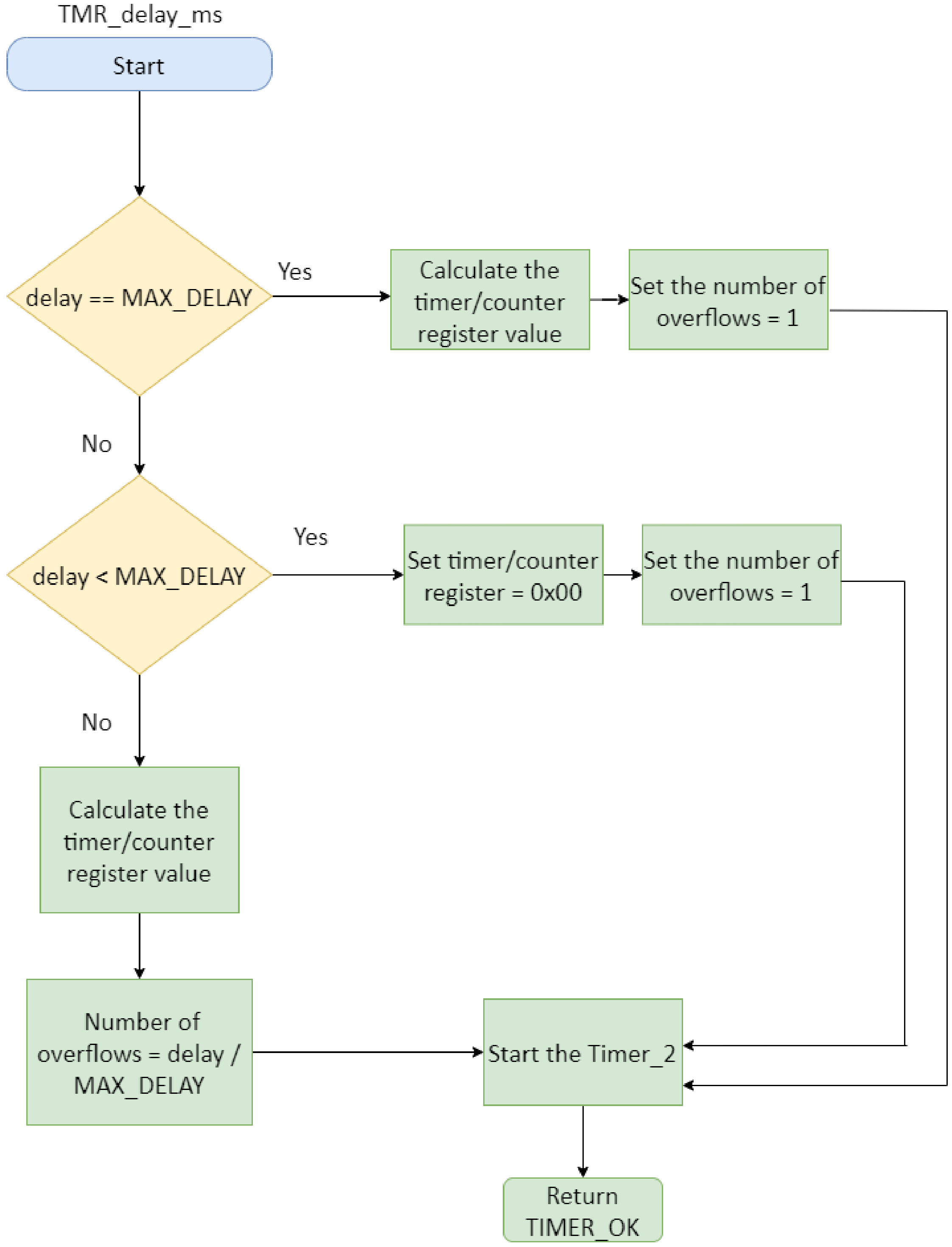
3.1. MCAL Flowcharts

3.1.1 TIMER Module

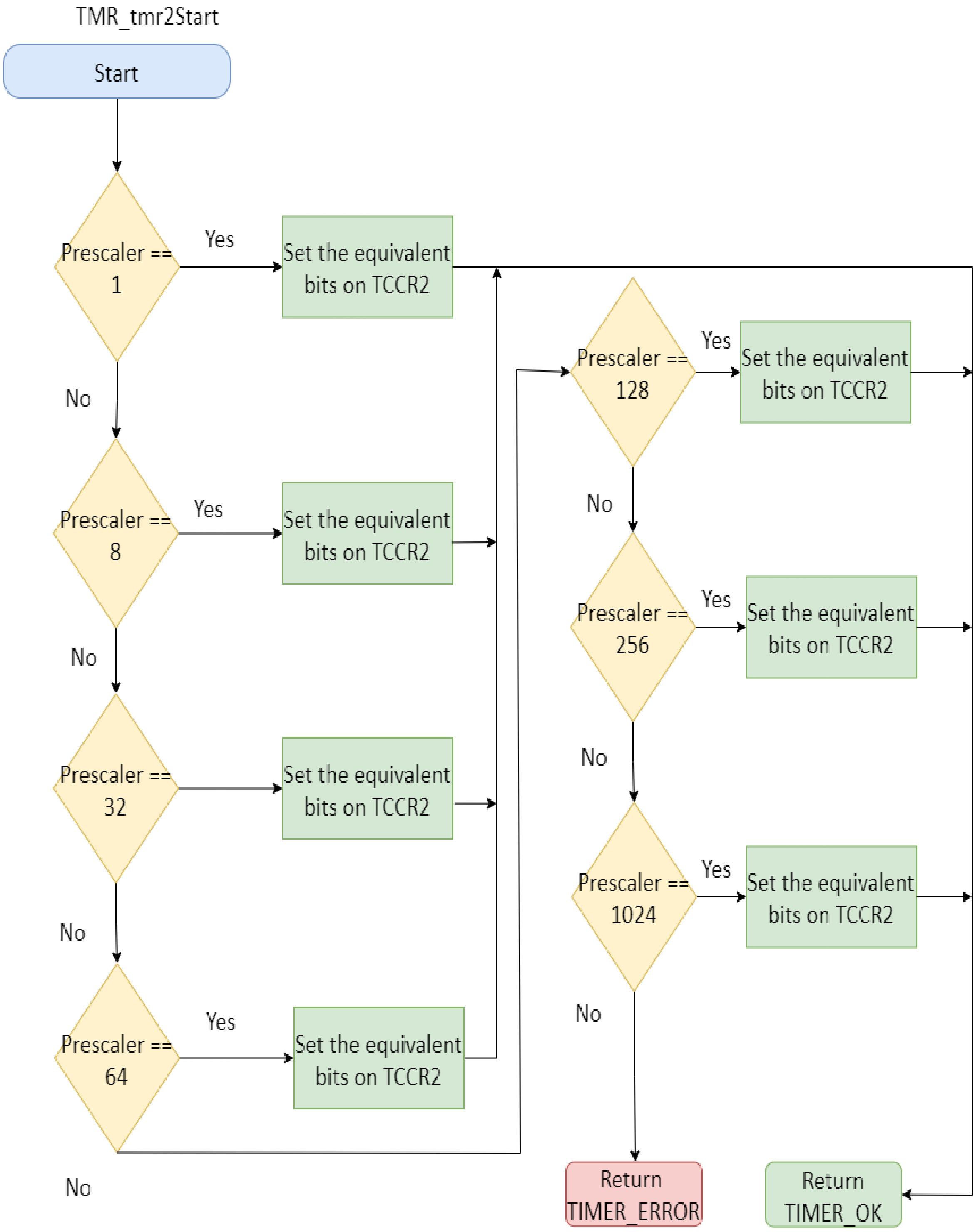
3.1.1.1. TMR_tmr2NormalModeInit



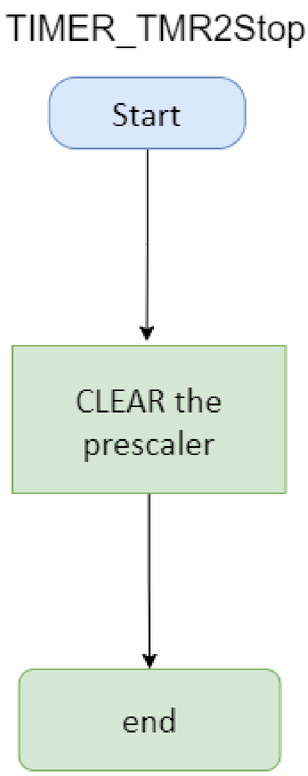
3.1.1.2.TMR_delay_ms



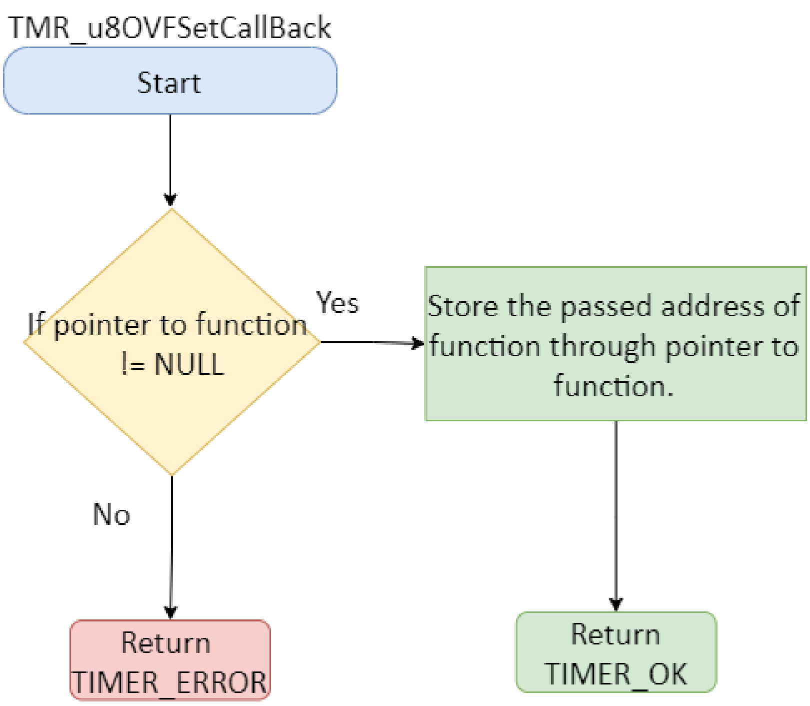
3.1.1.3. TMR_tmr2Start



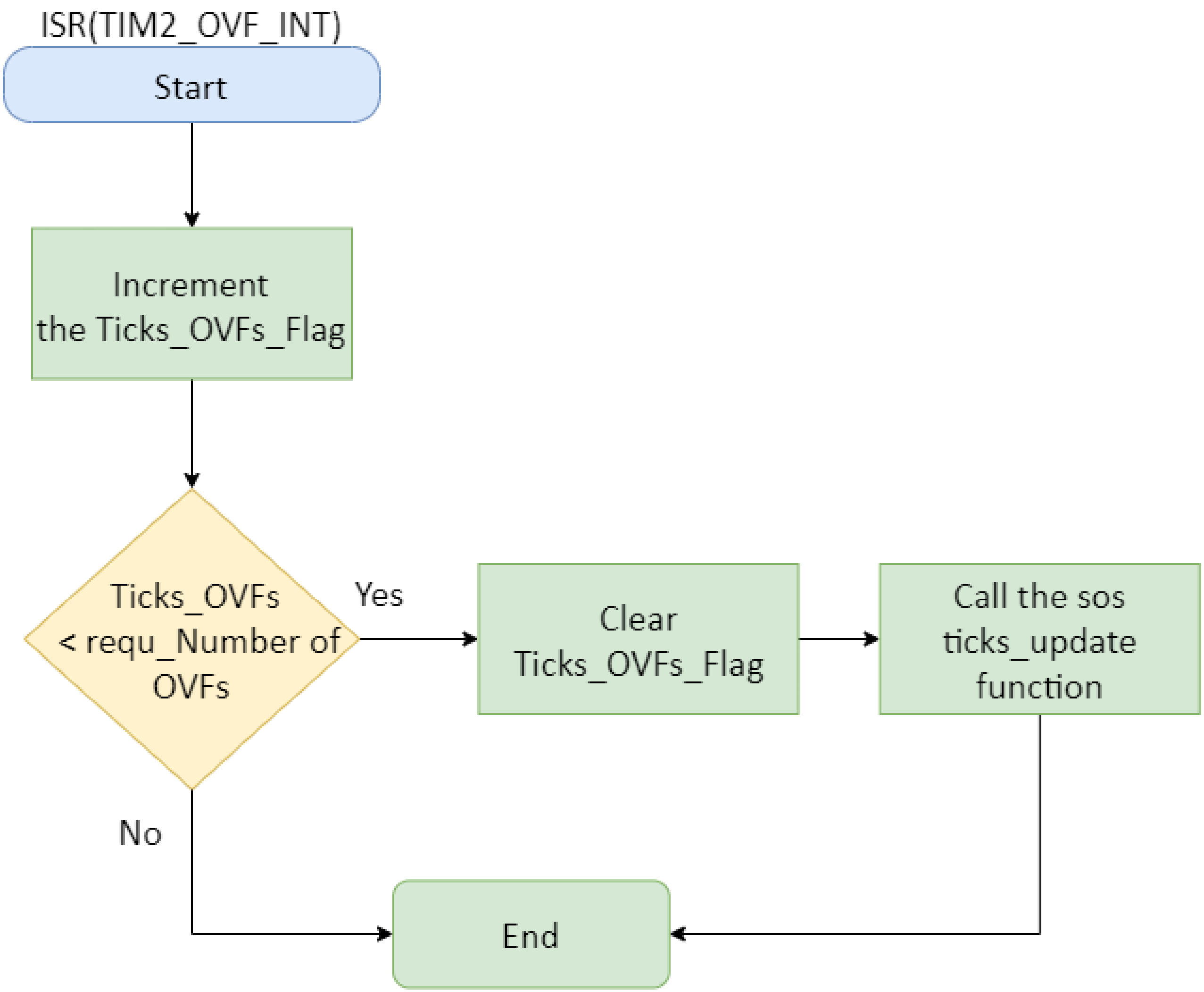
3.1.1.4. TMR_tmr2Stop



3.1.1.5. TMR_u8OVFSetCallback

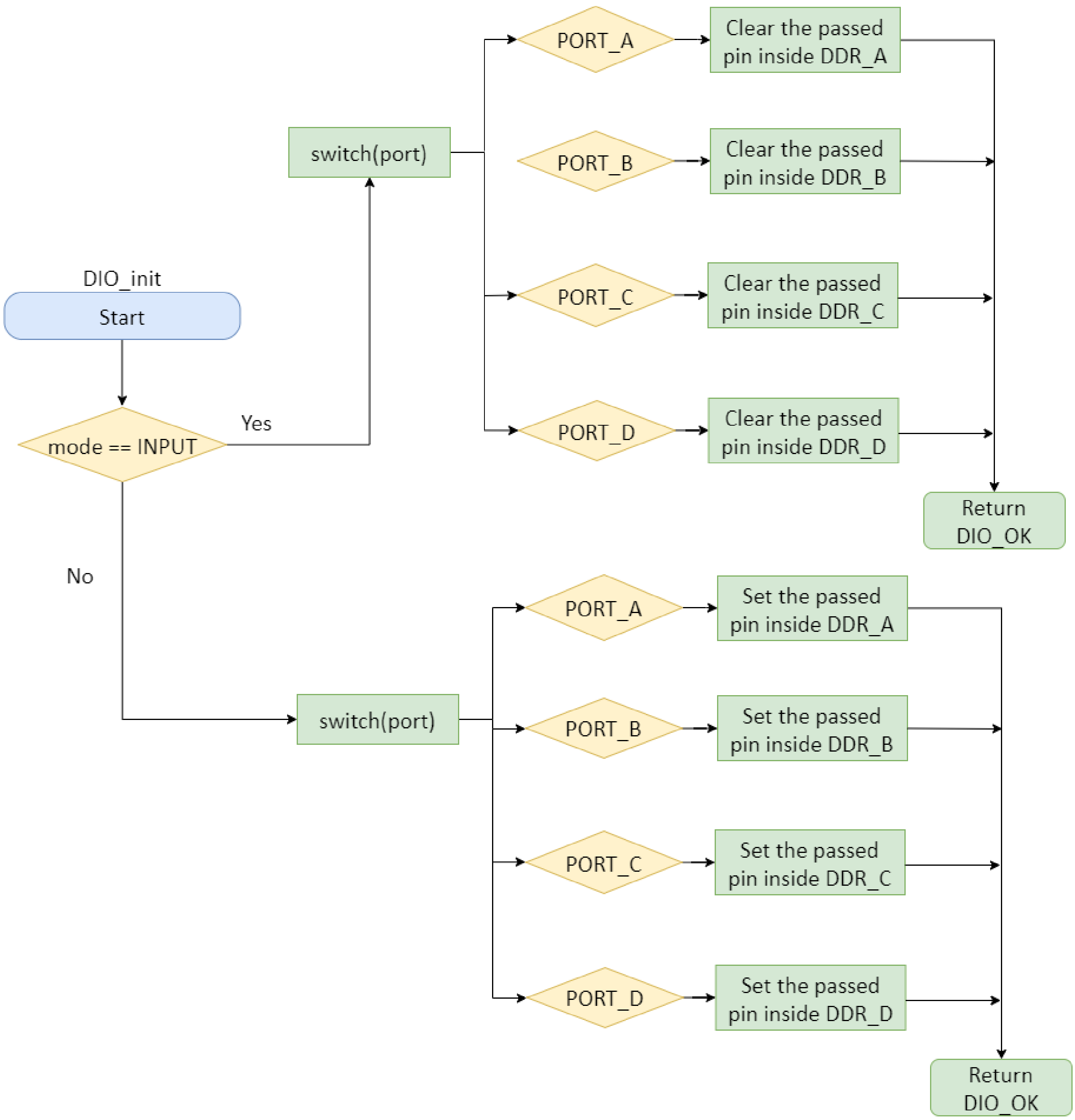


3.1.1.6. ISR(TIM2_OVF_INT)

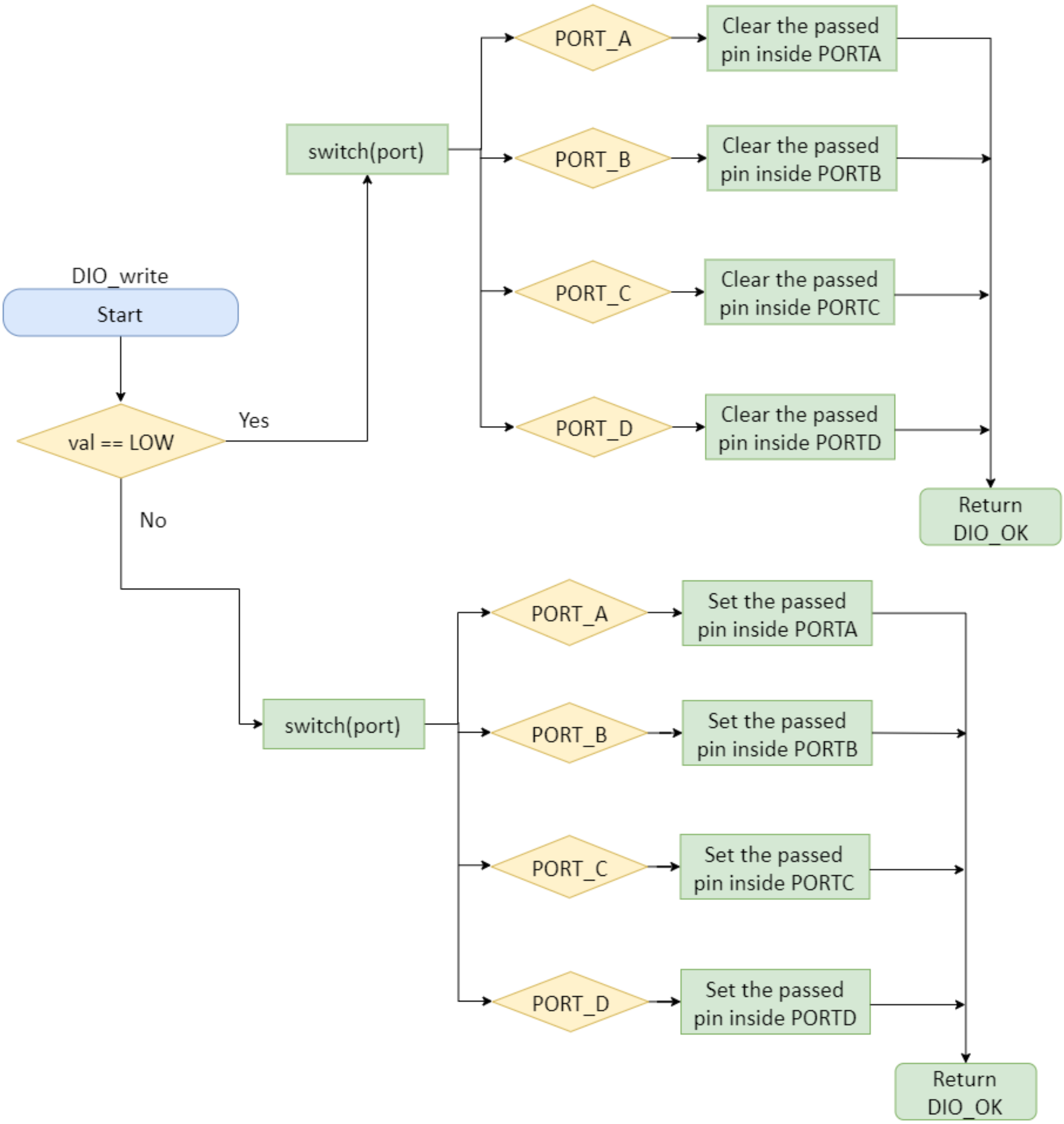


3.1.2 DIO Module

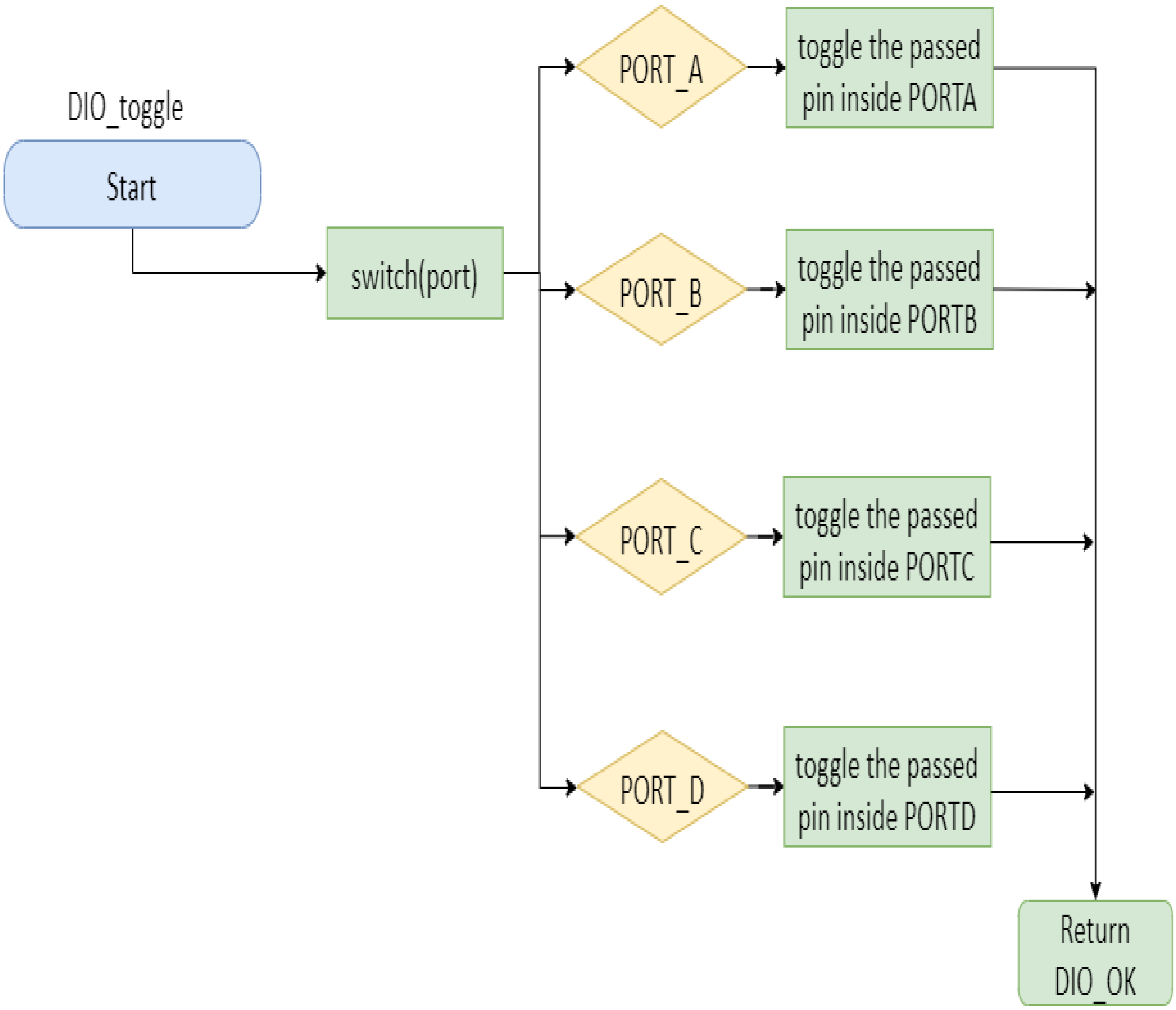
3.1.2.1 DIO_init



3.1.2.2 DIO_write



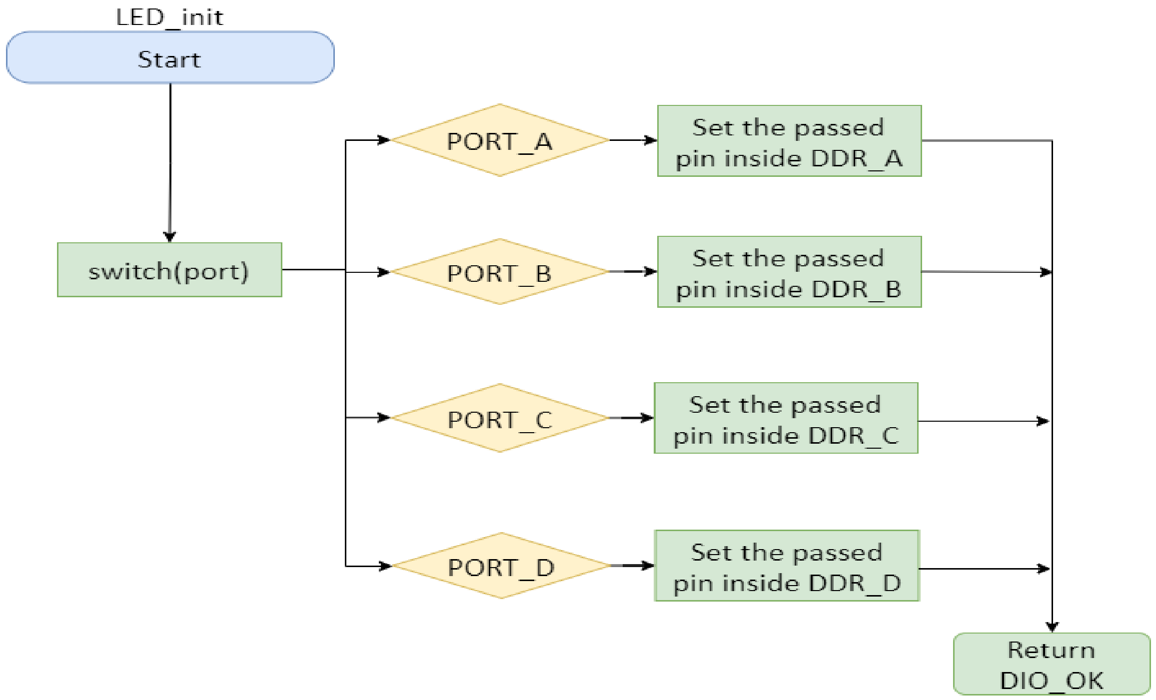
3.1.2.3 DIO_toggle



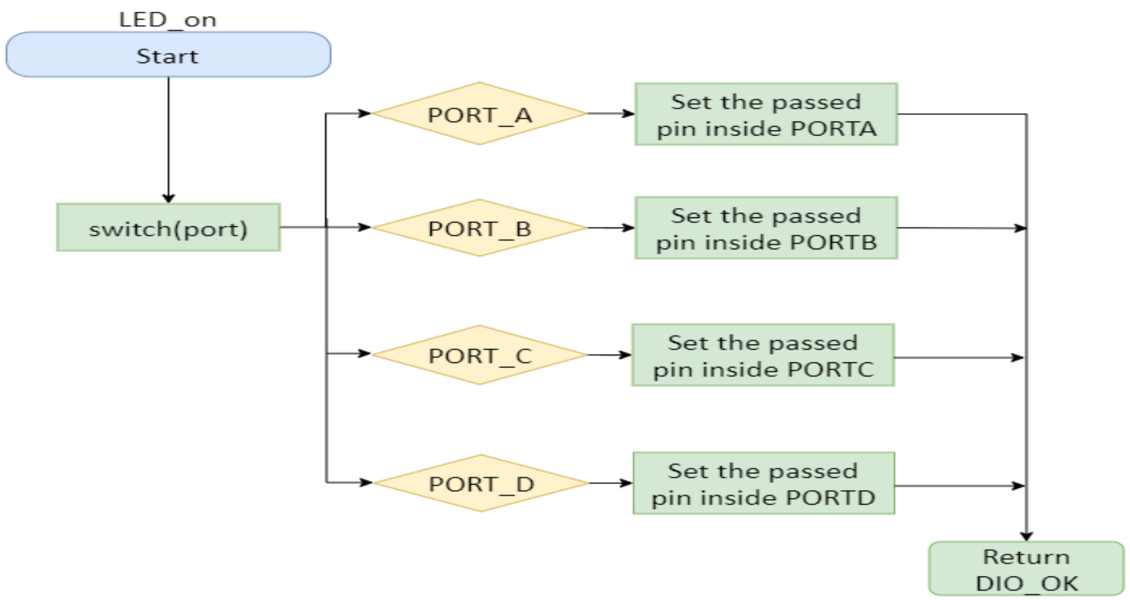
3.2. HAL Flowcharts

3.2.1 LED Module

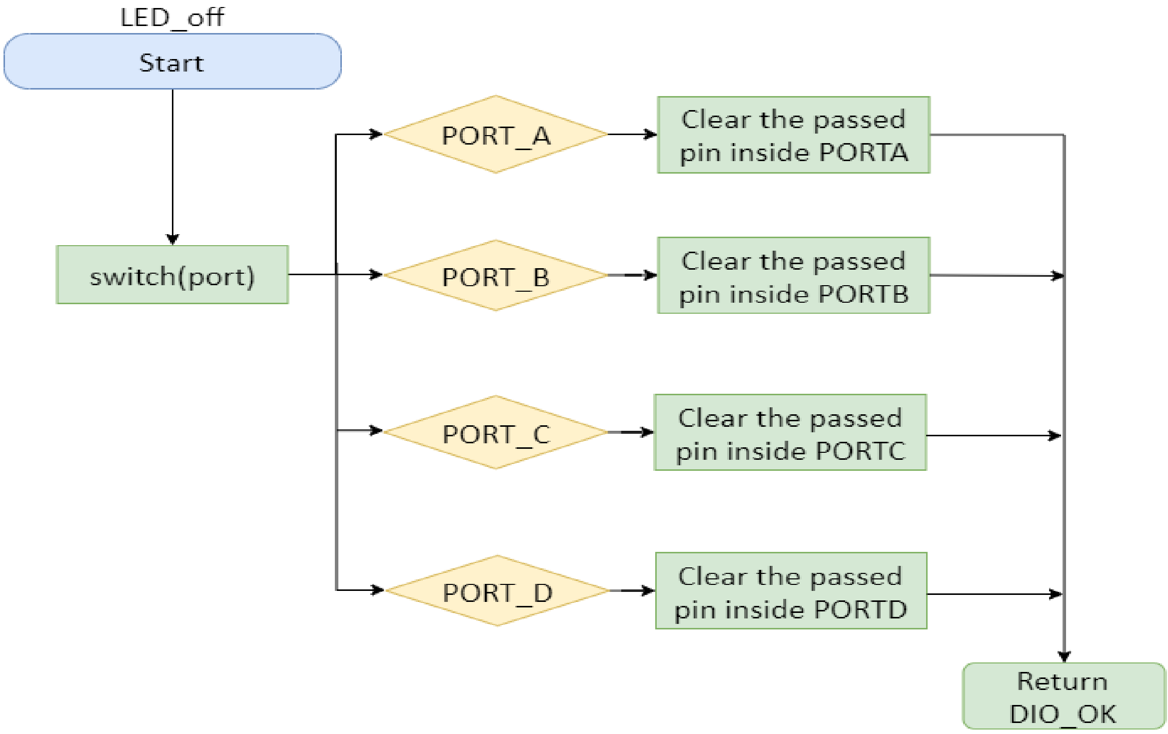
3.2.1.1 LED_init



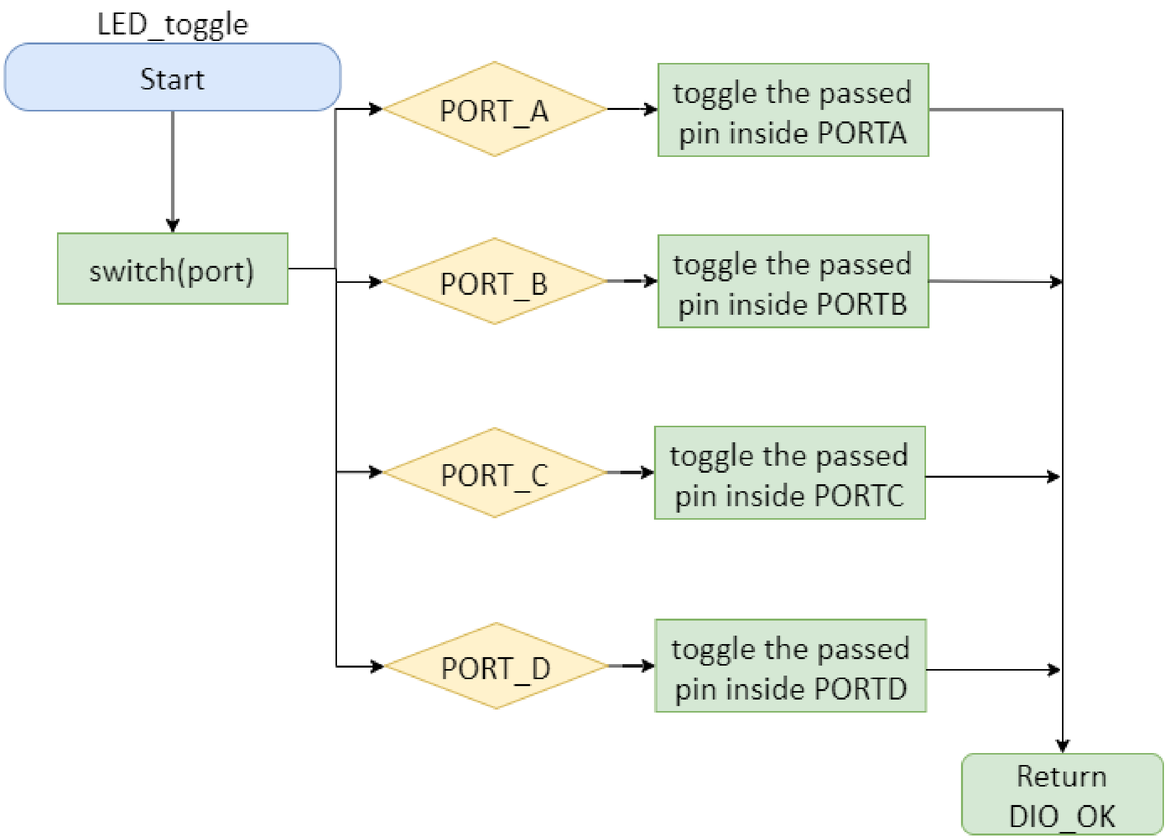
3.2.1.2 LED_on



3.2.1.3 LED_off

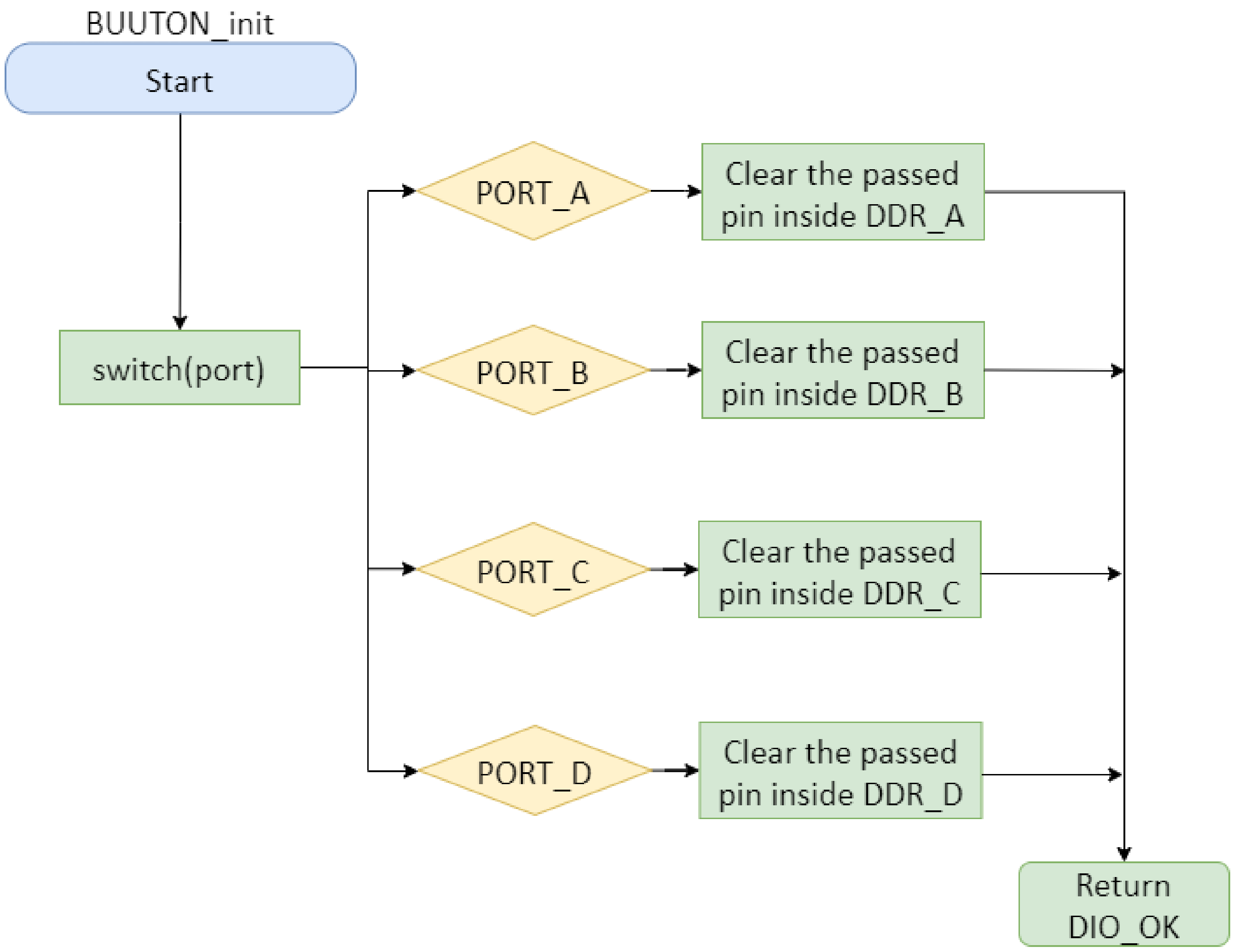


3.2.1.4 LED_toggle

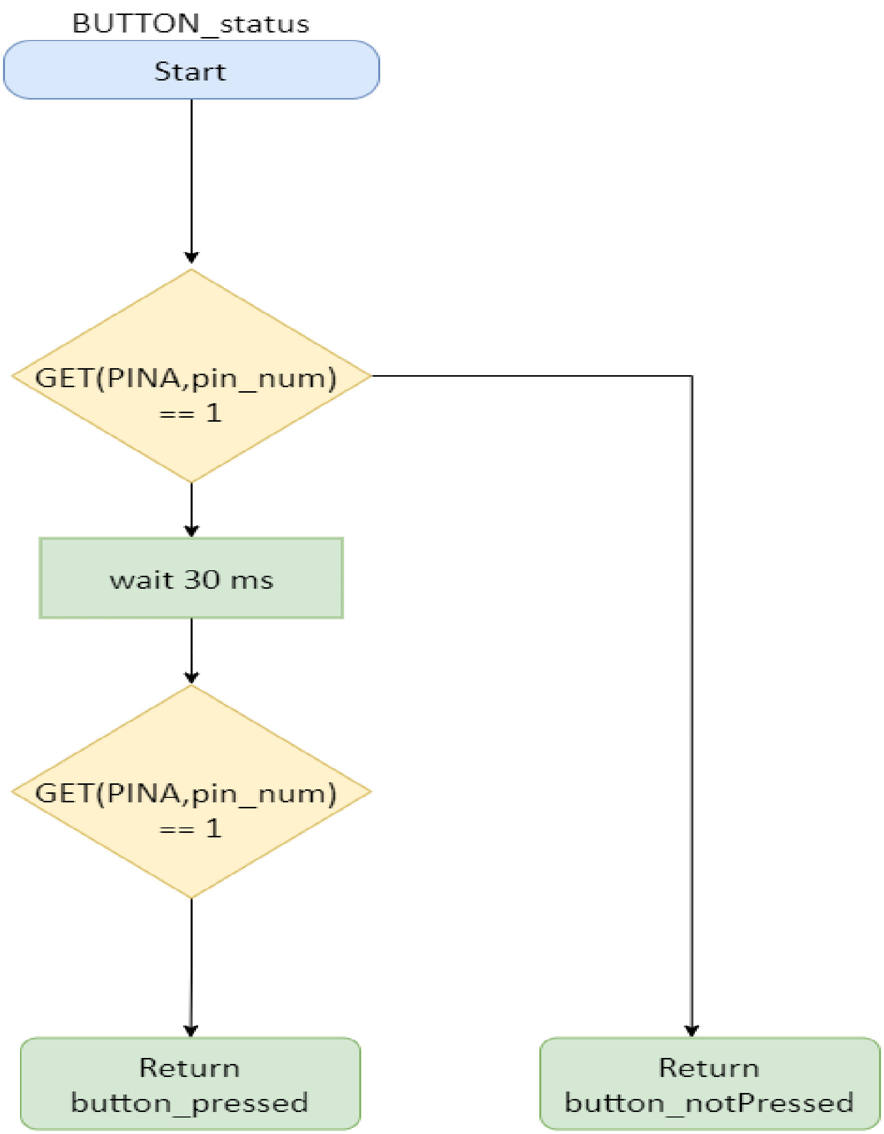


3.2.2 BUTTON Module

3.2.2.1 BUTT_init



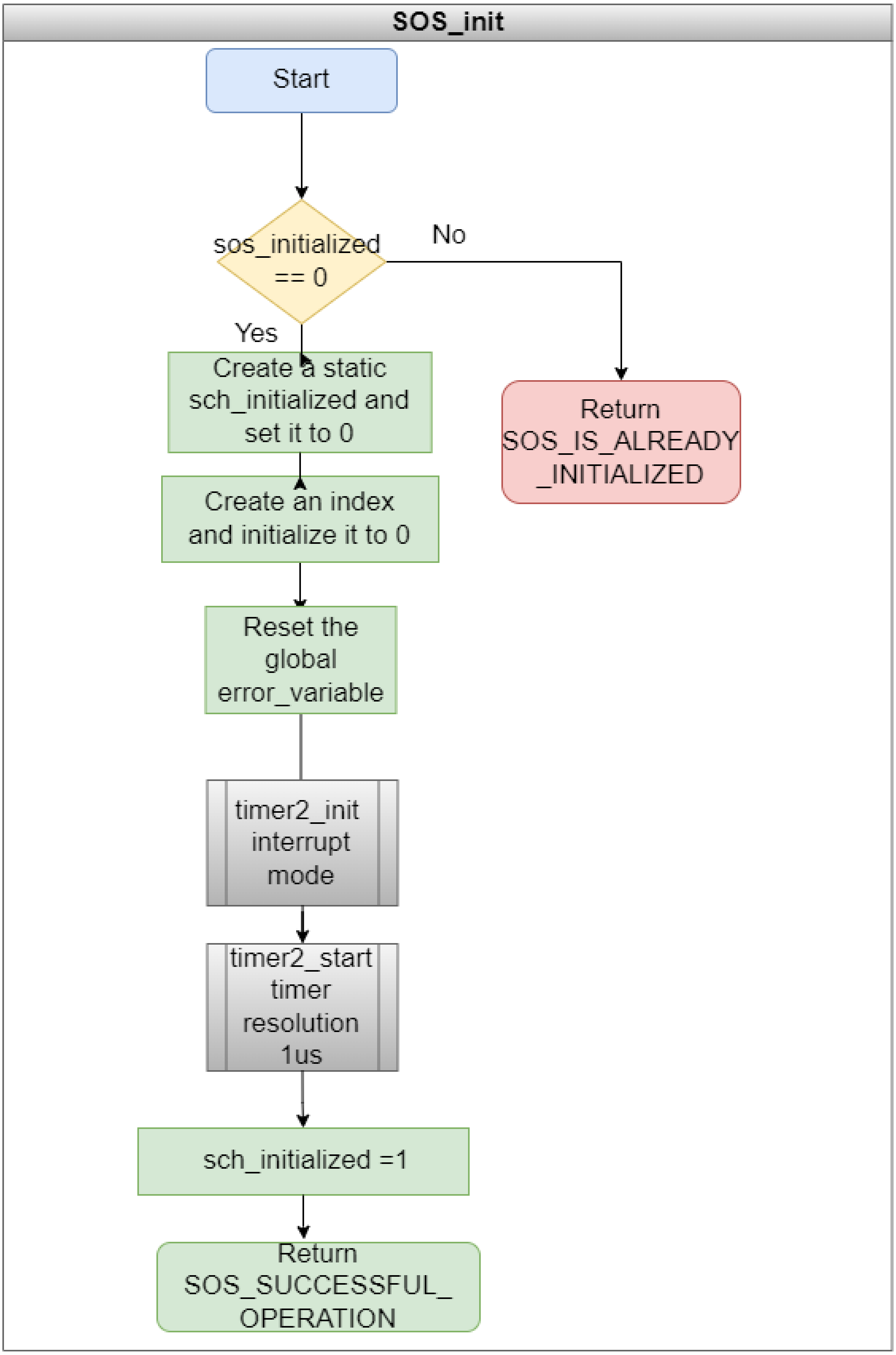
3.2.2.2 BUTT_status



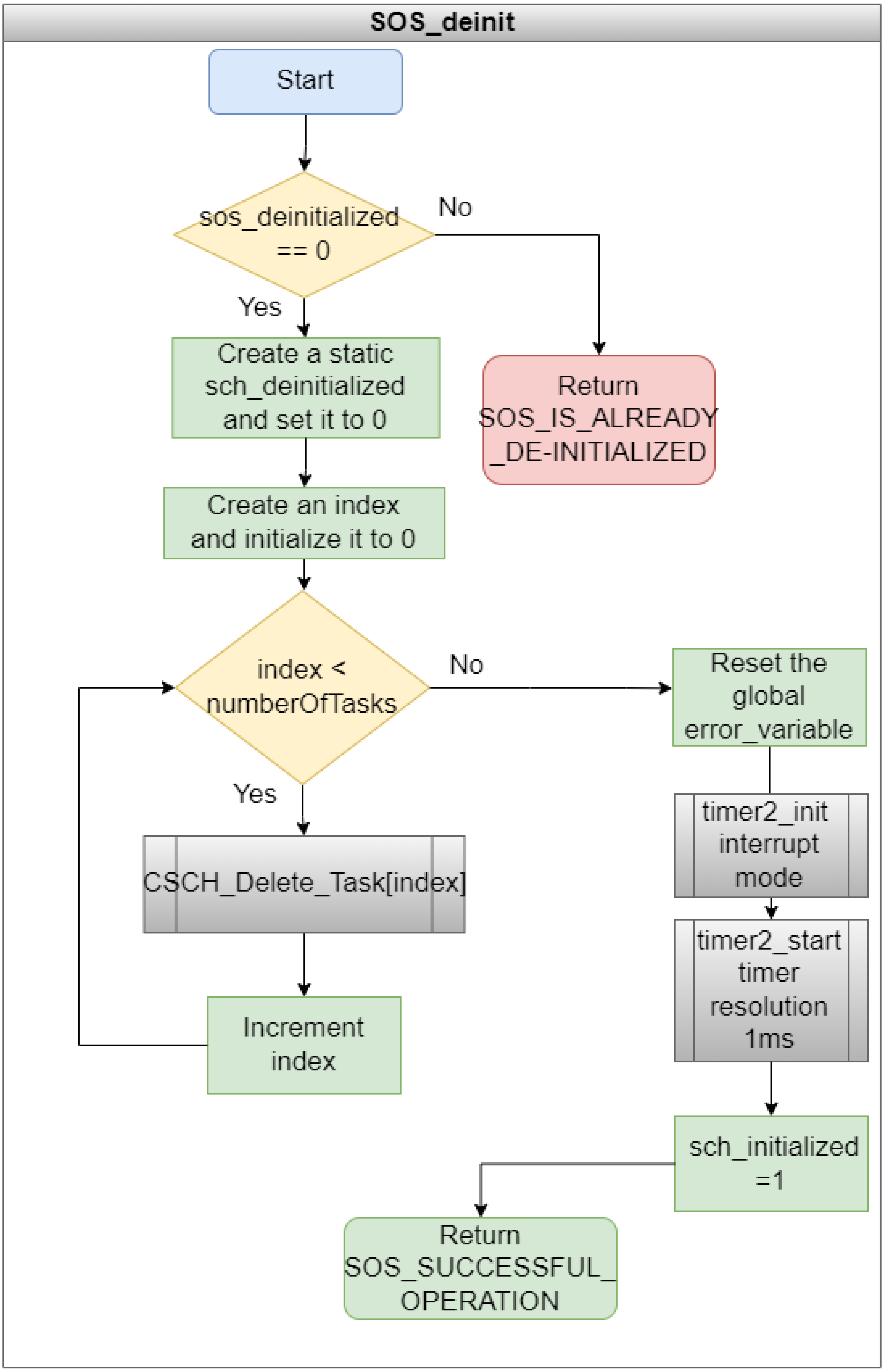
3.3. Service Layer Flowcharts

3.3.1 SOS Module

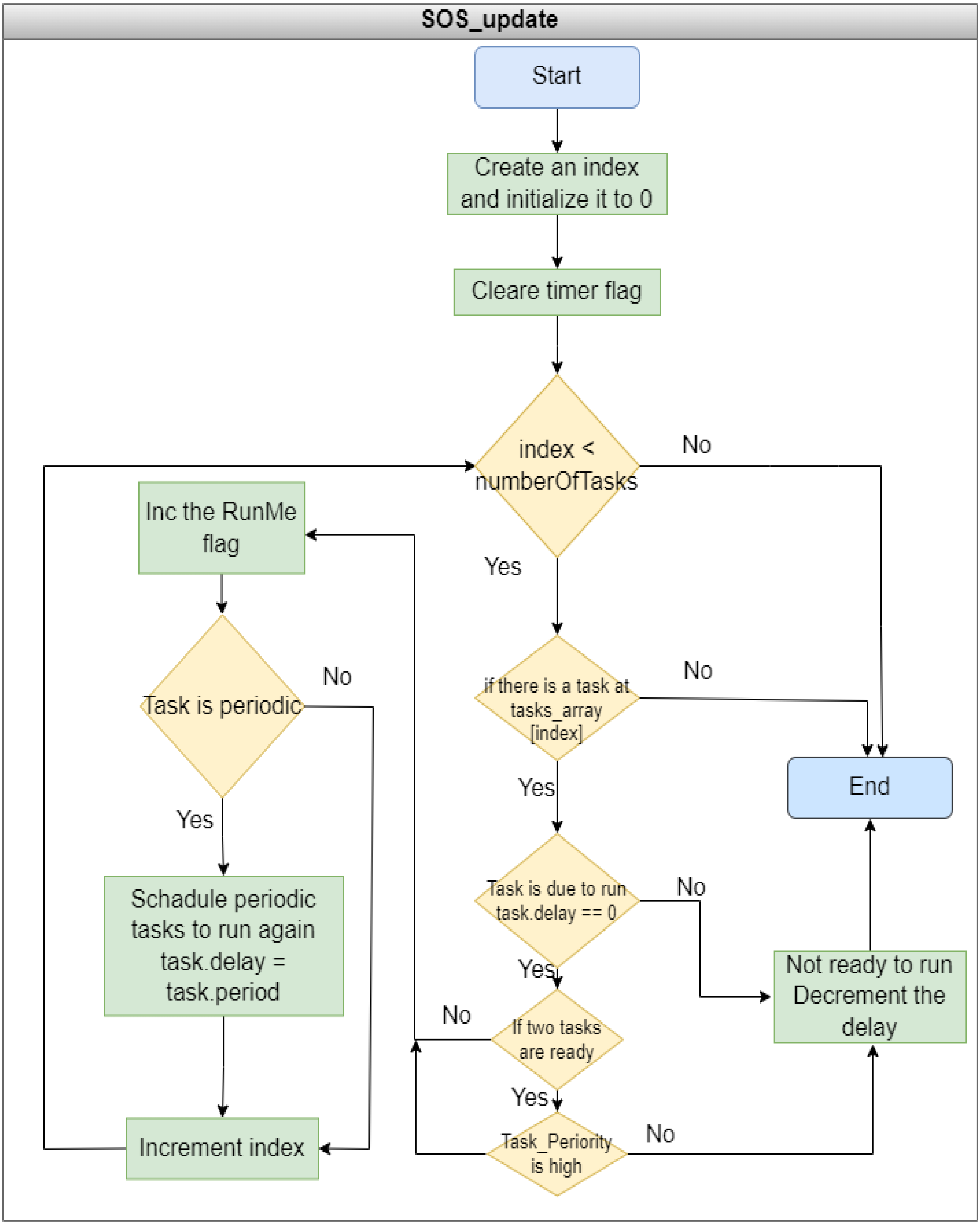
3.3.1.1 SOS_init



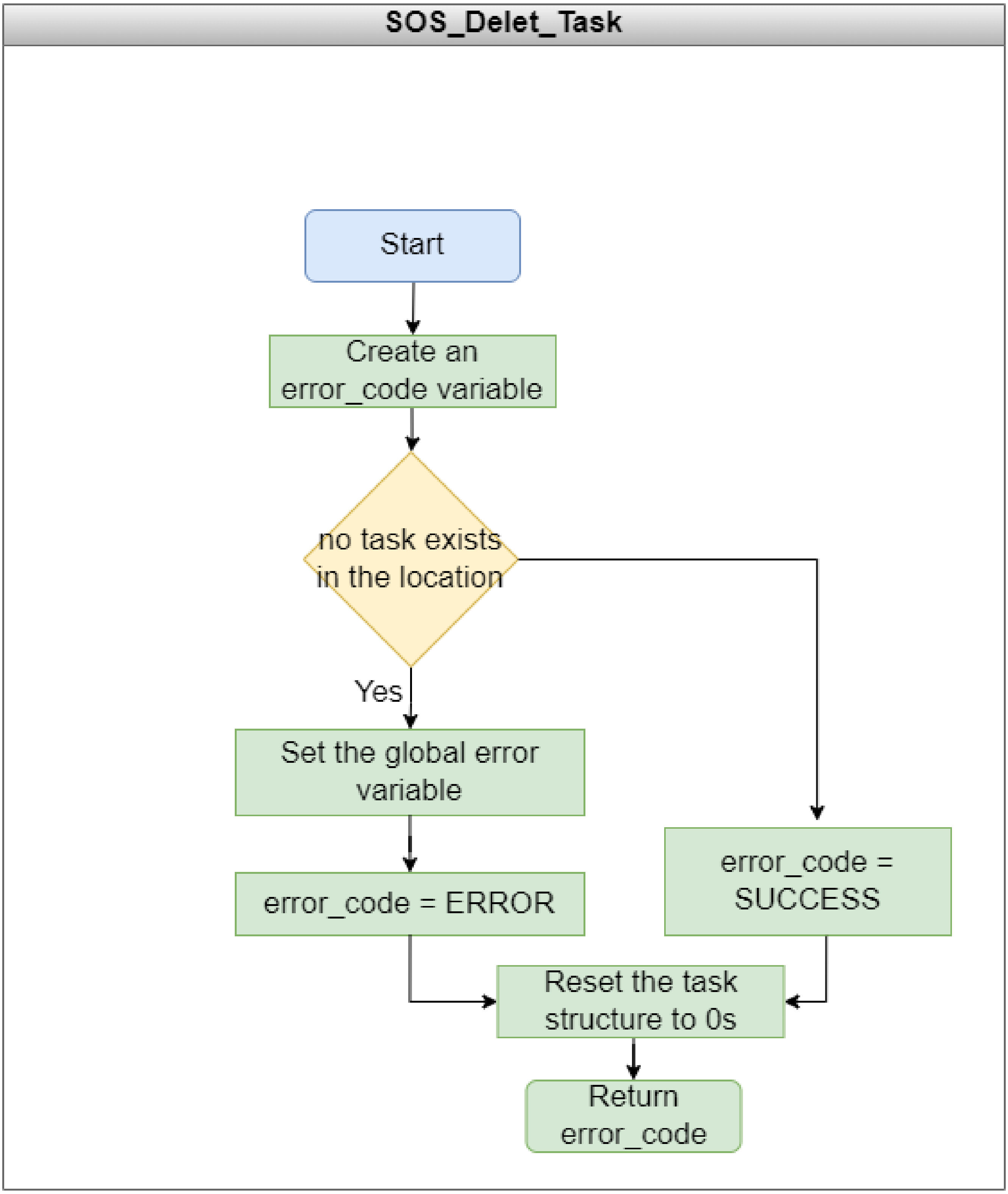
3.3.1.2 SOS_deinit



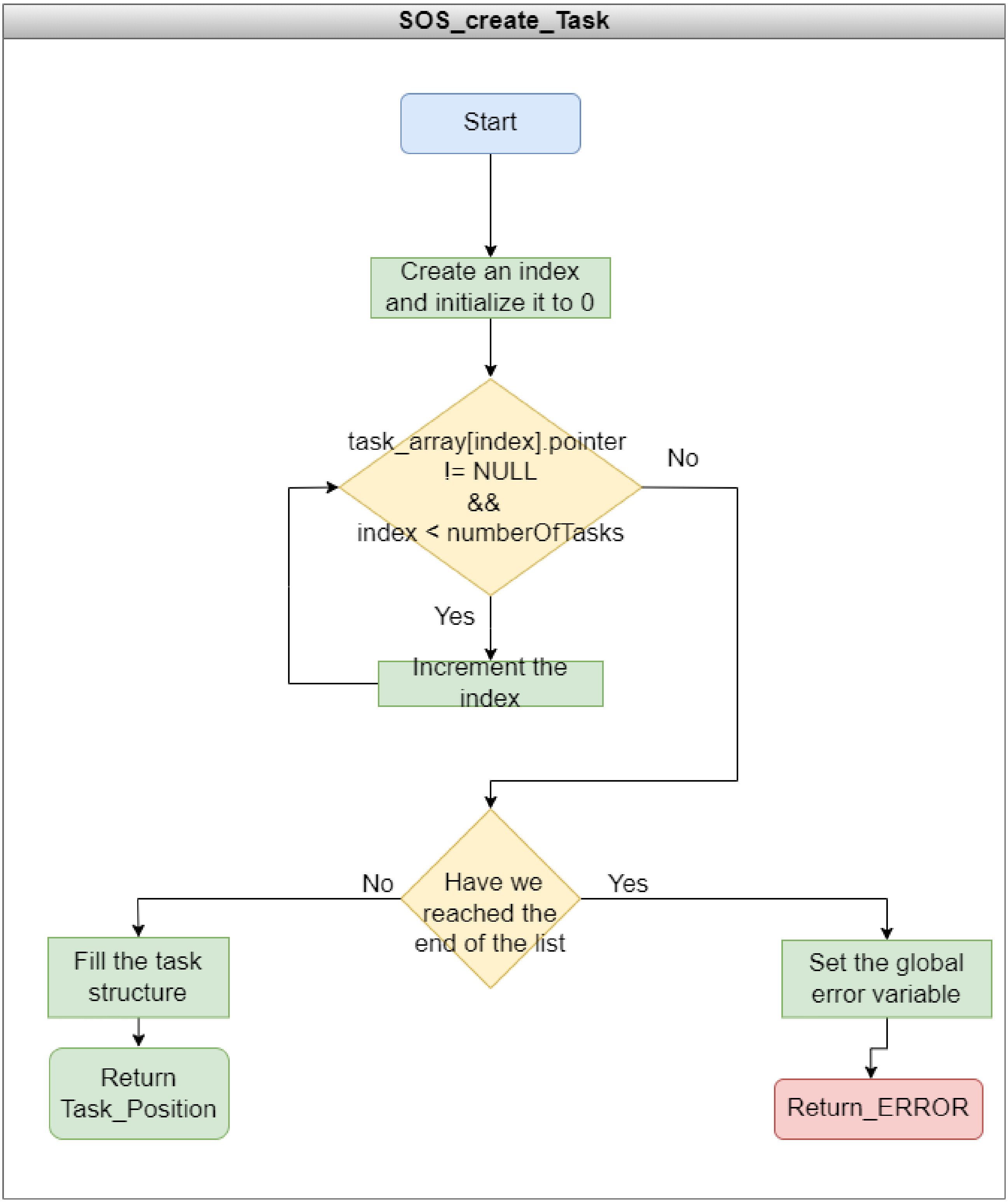
3.3.1.3 SOS_update



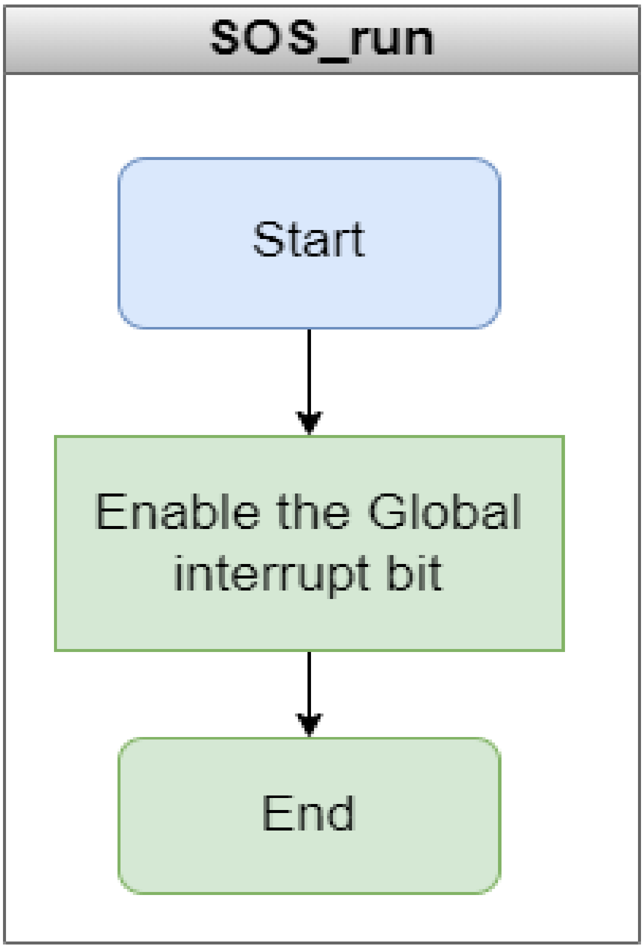
3.3.1.4 SOS_delete_task



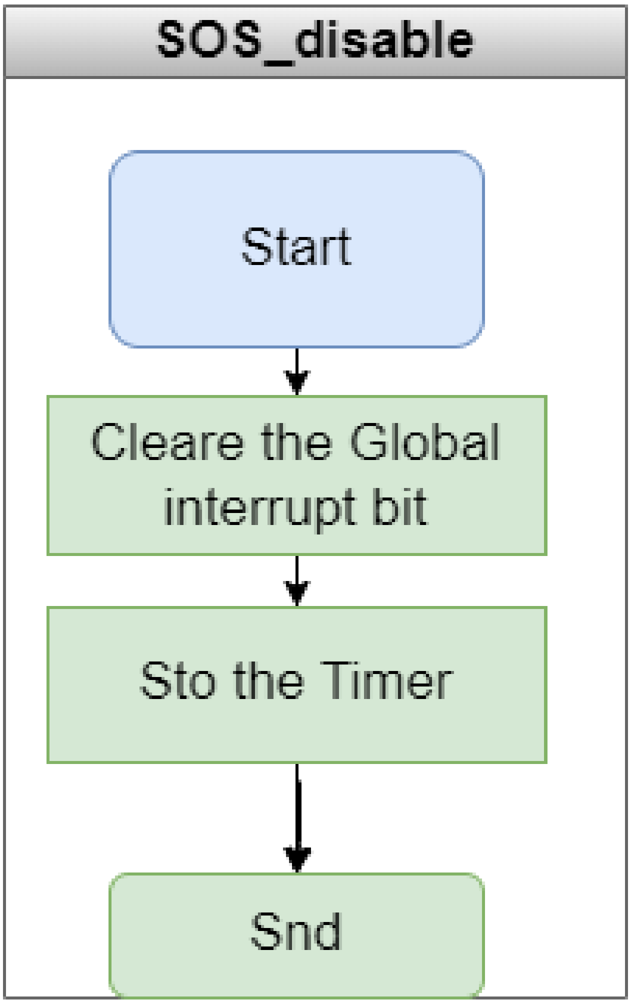
3.3.1.5 SOS_create_task



3.3.1.6 SOS_run



3.3.1.7 SOS_disable



3.3.1.7 SOS_dispatcher