

LED CONTROL V2

Ahmed Hesham – Sarah Mohamed
Automotive Embedded Bootcamp
6/20/23

1. Project Introduction.....	2
1.1. Project Components.....	2
1.2. Project Requirements.....	2
2. High Level Design.....	3
2.1. System Architecture	3
2.1.1. Layered Architecture	3
2.2. Modules Description.....	4
2.2.1. GPIO.....	4
2.2.2. SYSTICK	4
2.2.3. LED Module	4
2.2.4. BTN Module.....	4
2.2.5. Design.....	5
2.3. Drivers' Documentation (APIs).....	6
2.3.1 Definition.....	6
2.3.2. MCAL APIs	6
2.3.2.1. GPIO Driver.....	6
2.3.2.2. Systick Driver	8
2.3.3. HAL APIs	9
2.3.3.1. LED APIs.....	9
2.3.3.2. BTN APIs	9
2.3.4. APP APIs	10
3. Low Level Design	11
3.1. MCAL Layer	11
3.1.1. GPIO Module.....	11
3.1.1.a. PIN INDEX CHECK sub-process.....	11
3.1.1.1. gpio_pin_init.....	12
3.1.1.2. gpio_pin_write	15
3.1.1.3. gpio_pin_toggle.....	16
3.1.1.4. gpio_pin_read	17
3.1.1.5. gpio_pin_enable_notification	18
3.1.1.6. gpio_pin_set_callback.....	19
3.1.2. systick Module	20
3.1.1.1. Systick_init	20
3.1.1.2. Systick_disable	21
3.1.1.3. Systick_reload.....	22
3.1.1.4. systick_enable.....	23
3.1.1.5. SysTick_Handler	24
3.2. HAL Layer	25
3.2.1. LED Module	25
3.2.1.1. Led_init.....	25
3.2.1.2. Led_on	26
3.2.1.3. Led_off	27
3.2.2. BTN Module.....	28
3.2.2.1. Button_init	28
3.2.2.2. Button_get_state	28

LED CONTROL DESIGN v2

1. Project Introduction

The project to develop GPIO driver and SYSTICK driver, and use them to control RGB LED on the TivaC board based using the push button.

1.1. Project Components

Tiva C verification board using SW1 and rgb LED

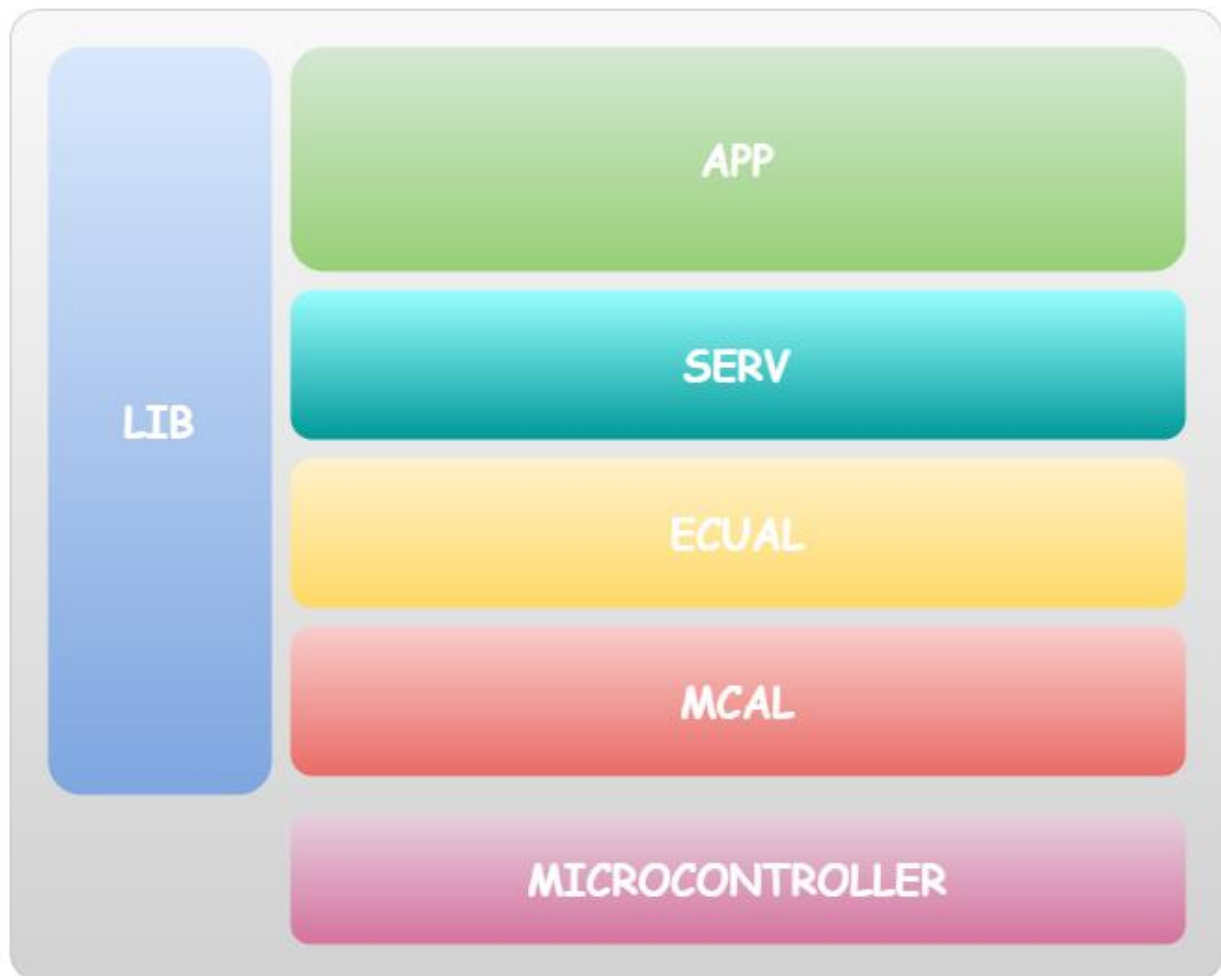
1.2. Project Requirements

1. The RGB LED is OFF initially
2. Pressing SW1:
 - 2.1. After the first press, the Red led is on for 1 sec only
 - 2.2. After the second press, the Green Led is on for 1 sec only
 - 2.3. After the third press, the Blue led is on for 1 sec only
 - 2.4. After the fourth press, all LEDs are on for 1 sec only
 - 2.5. After the fifth press, should disable all LEDs
 - 2.6. After the sixth press, repeat steps from 1 to 6

2. High Level Design

2.1. System Architecture

2.1.1. Layered Architecture



2.2. Modules Description

2.2.1. GPIO

The *GPIO* module reads input signals from the system's sensors (such as buttons) and drives output signals to the system's actuators (such as *LEDs*). It provides a set of APIs to configure the direction and mode of each pin (input/output, pull-up/down resistor), read the state of an input pin, and set the state of an output pin.

2.2.2. SYSTICK

Cortex-M4 includes an integrated system timer, SysTick, which provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism. The counter can be used in several different ways.

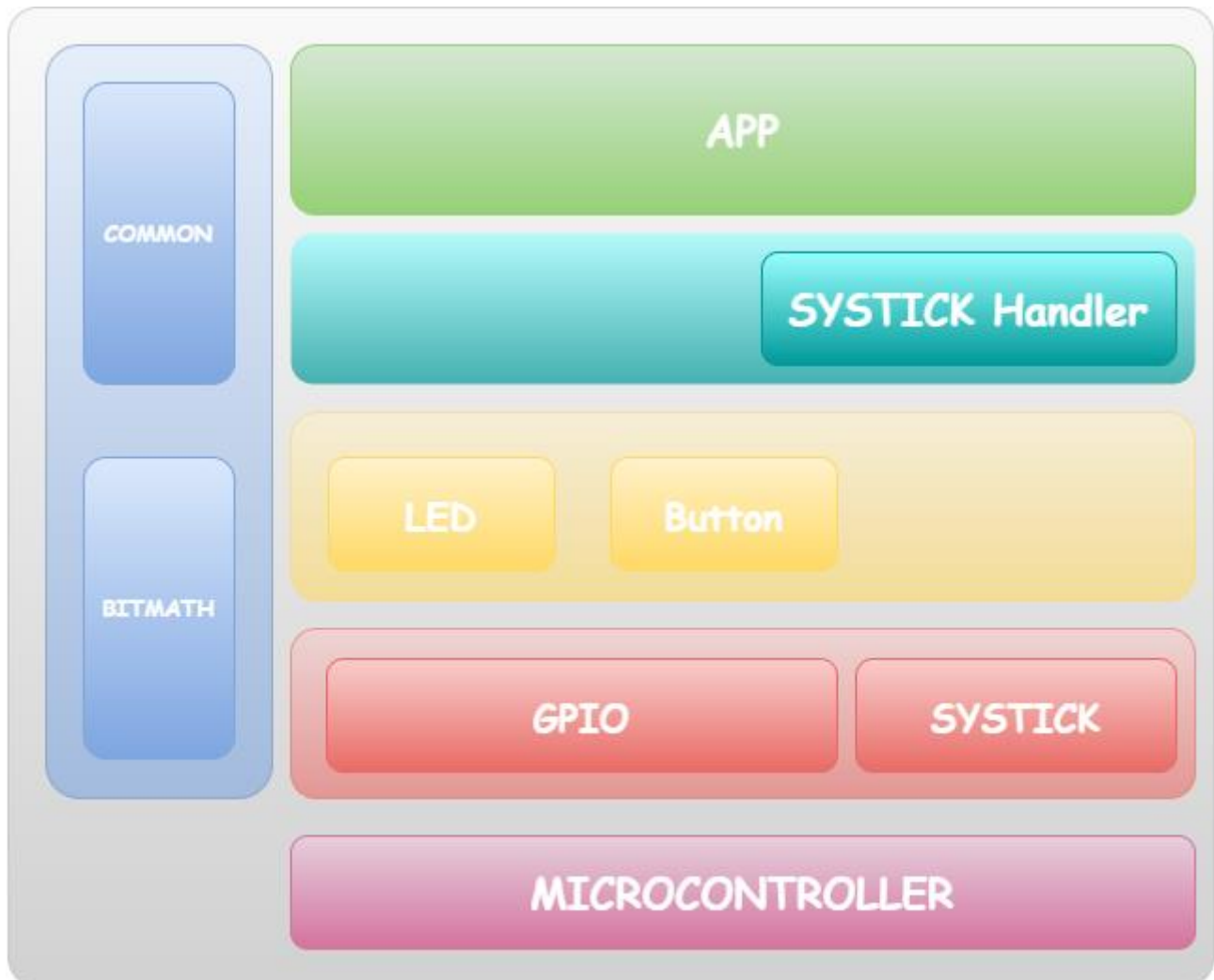
2.2.3. LED Module

The LED module is responsible for initializing the LEDs and turning them on/off or toggling them.

2.2.4. BTN Module

The *BTN* (Button) module is responsible for reading the state of the system's buttons. It provides a set of APIs to enable/disable button interrupts, set the button trigger edge (rising/falling/both), and define an ISR that will be executed when a button press is detected.

2.2.5. Design



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. GPIO Driver

```

| Brief
|           This function is used to initialize a specific pin
| Parameters
|           [in]      ptr_str_gpio_config: Ptr to the gpio instance structure
|           [out]     none
| Return
|           ERROR_OK : In case of successeion
|           GPIO_INVALID_PIN_INDEX : In case of wrong pin index
|           GPIO_INVALID_PIN_MODE : In case of wrong mode choosen
|           GPIO_INVALID_PIN_DIRECTION : In case of wrong direction choosen
|           GPIO_INVALID_OP_CURRENT :In case of wrong current choosen
|           GPIO_INVALID_INTERNAL_ATTACH : In case of wrong internal attach
|
enu_error_status_t_ gpio_pin_init(str_gpio_config_t* ptr_str_gpio_config)

| Brief
|           This function is used to Write a specific output on a pin
| Parameters
|           [in]      uint8_pin_index : Pin index used
|           [in]      enu_pin_level : Output level to be written
|           [out]     none
| Return
|           ERROR_OK:In case of successeion
|           GPIO_INVALID_PIN_INDEX:In case of wrong pin index
|           GPIO_INVALID_PIN_LEVEL:In case of wrong output level choosen
|
enu_error_status_t_ gpio_pin_write(uint8_t_ uint8_pin_index, enu_gpio_pin_level_t_ enu_pin_level)

```

```

| Brief

```

```

|                                     This function is used to toggle output level on a pin
| Parameters
|     [in]      uint8_pin_index:Pin Index used
|     [out]     none
|
| Return
|
|     ERROR_OK:In case of successeion
|     GPIO_INVALID_PIN_INDEX:In case of wrong pin index
|
enu_error_status_t_ gpio_pin_toggle(uint8_t_ uint8_pin_index)

| Brief
|                                     This function is used to get pin's input value
| Parameters
|     [in]      uint8_pin_index:Pin index used
|     [out]     uint8_pin_state:Pin level state
| Return
|
|     ERROR_OK:In case of successeion
|     GPIO_INVALID_PIN_INDEX:In case of wrong pin index
|
enu_error_status_t_ gpio_pin_read(uint8_t_ uint8_pin_index, uint8_t_* uint8_pin_state)

| Brief
|                                     This function is used to enable pin's interrupt
| Parameters
|     [in]      uint8_pin_index:Pin index used
|     [out]     none
| Return
|
|     ERROR_OK:In case of successeion
|     GPIO_INVALID_PIN_INDEX:In case of wrong pin index
|
enu_error_status_t_ gpio_pin_enable_notification(uint8_t_ uint8_pin_index)

| Brief
|                                     This function is used to set callback function for a specific pin's interrupt handler
| Parameters
|     [in]      uint8_pin_index:Pin index used
|               ptr_callback:pointer to a callback function
|     [out]     none
| Return
|
|     ERROR_OK:In case of successeion
|     GPIO_INVALID_PIN_INDEX:In case of wrong pin index
|     PASSING_NULL_PTR:In case of passing null pointer
|
enu_error_status_t_ gpio_pin_set_callback(uint8_t_ uint8_pin_index, ptr_gpio_callback_t_
ptr_callback)

```


2.3.2.2. SysTick Driver

```

| Brief
|     This function is used to initialize system tick and load its counter
| Parameters
|     [in]   uint32_delay_time_ms: desired time to interrupt after it
|     [out]  none
| Return
|     ERROR_OK           :      In case of success
|     SYSTICK_WRONG_VALUE :      In case of uint32_delay_time_ms = 0
|
enu_error_status_t_ systick_init(uint32_t_ uint32_delay_time_ms);

| Brief
|     This function is used to reload systick
| Parameters
|     [in]   uint32_delay_time_ms: desired time to interrupt after it
|     [out]  none
| Return
|     ERROR_OK           :      In case of success
|     SYSTICK_WRONG_VALUE :      In case of uint32_delay_time_ms = 0
|
enu_error_status_t_ systick_reload(uint32_t_ uint32_delay_time_ms);

| Brief
|     This function is used to disable systick
| Parameters
|     [in]   none
|     [out]  none
| Return
|     ERROR_OK           :      In case of success
|
enu_error_status_t_ systick_disable(void);

| Brief
|     This function is used to enable systick
| Parameters
|     [in]   none
|     [out]  none
| Return
|     ERROR_OK           :      In case of success
|
enu_error_status_t_ systick_enable(void);

| Brief
|     This function is used to set callback function for systick interrupt
| Parameters
|     [in]   a_void_ptr      :      ptr to callback function
|     [out]  none
| Return
|     ERROR_OK           :      In case of success
|     PASSING_NULL_PTR   :      In case of passing null pointer
|
enu_error_status_t_ systick_set_callback(void (*a_void_ptr) (void));

```

2.3.3. HAL APIs

2.3.3.1. LED APIs

```
|      @brief          This function is initialize pin
|      @param [in]      none
|      @return          ERROR_OK      :      In case of succeesion
|      @return          LED_NOK       :      In case of wrong pin index
enu_error_status_t_ led_init(void);

|      @brief          This function is turn on led
|      @param [in]      uint8_ledpin_index      :      Pin index used
|      @return          ERROR_OK      :      In case of succeesion
|      @return          LED_NOK       :      In case of wrong pin index
enu_error_status_t_ led_on(uint8_t_ uint8_ledpin_index );

|      @brief          This function is turn off led
|      @param [in]      uint8_ledpin_index:      Pin index used
|      @return          ERROR_OK      :In case of succeesion
|      @return          LED_NOK       : In case of wrong pin index
enu_error_status_t_ led_off(uint8_t_ uint8_ledpin_index );
```

2.3.3.2. BTN APIs

```
|      @brief          This function is used for initializing button
|      @param [in]      ptr_func          :      pointer to callback
|      @return          ERROR_OK          :      In case of succeesion
|      @return          PASSING_NULL_PTR   :      In case of passing null pointer
|      @return          BUTTON_NOK        :      In case of wrong pin index
enu_error_status_t_ button_init(void(*ptr_func)(void));

|      @brief          This function is used for getting button state
|      @param [in]      buttonpin          :      Pin index used
|      @param [out]     ptr_uint8_button_state      :      returns button state
|      @return          ERROR_OK          :In case of succeesion
|      @return          BUTTON_NOK        :In case of wrong pin index
enu_error_status_t_ button_get_state(uint8_t_ uint8_button_pin, uint8_t_* ptr_uint8_button_state);
```

2.3.4. APP APIs

```

| Brief
|           This function is used to initialize drivers used
| Parameters
|   [in]      none
|   [out]     none
|
| Return
|           none
|
void app_init(void)

| Brief
|           This function is the called back when button pressed
| Parameters
|   [in]      none
|   [out]     none
|
| Return
|           none
|
static void button_task(void)

| Brief
|           This function is the called to perform LED task
| Parameters
|   [in]      none
|   [out]     none
|
| Return
|           none
|
static void app_led_task(void)

| Brief
|           This function is the called back when system tick interrupts
| Parameters
|   [in]      none
|   [out]     none
|
| Return
|           none
|
static void app_systick_task(void)

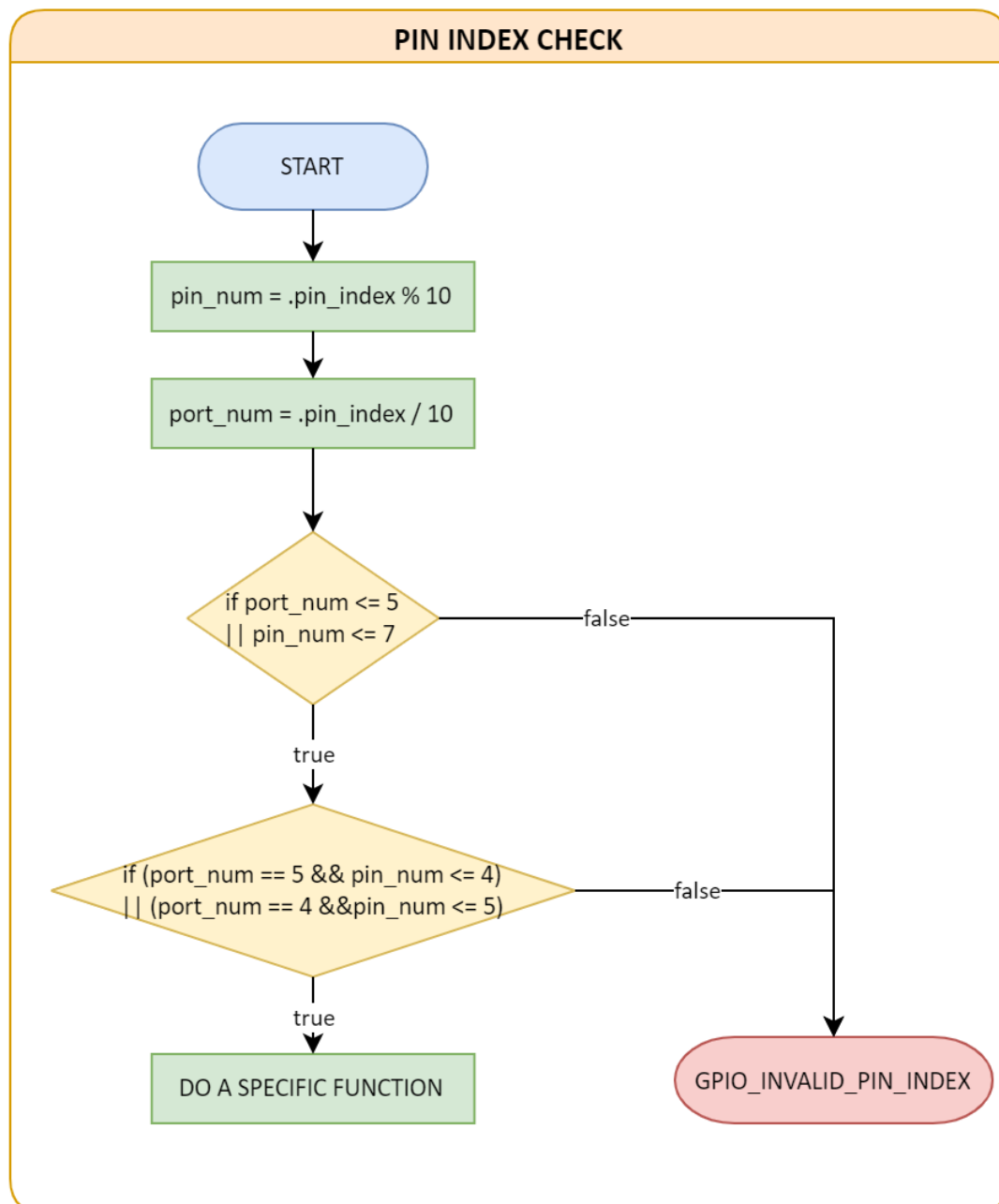
```

3. Low Level Design

3.1. MCAL Layer

3.1.1. GPIO Module

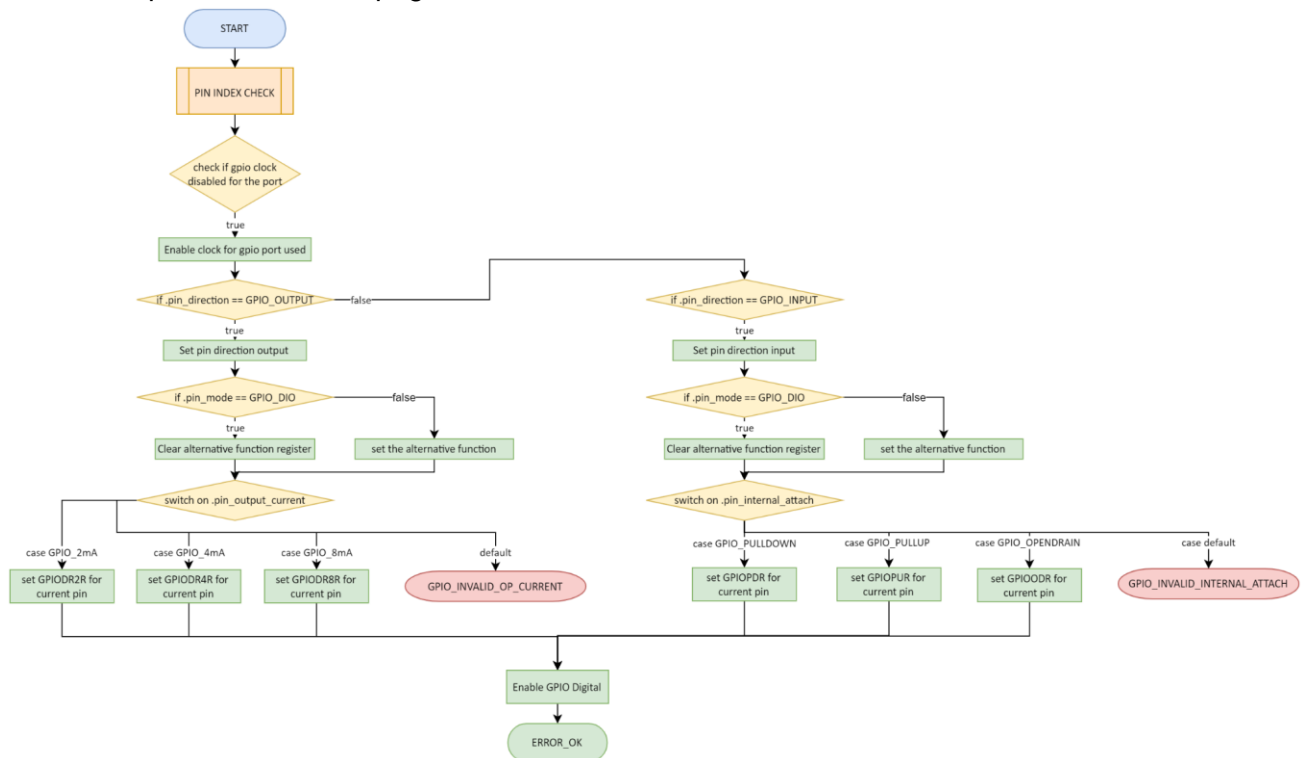
3.1.1.a. PIN INDEX CHECK sub-process



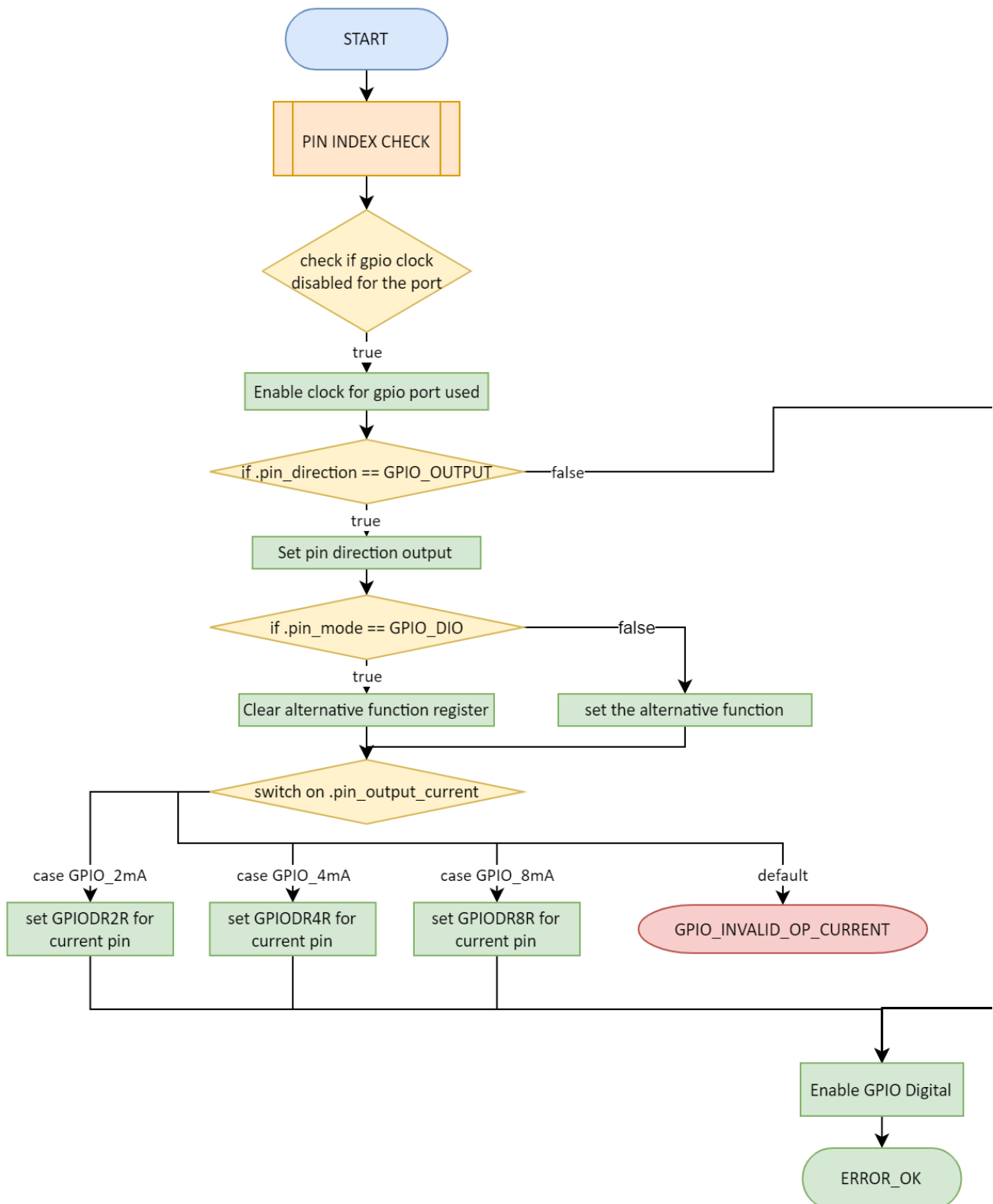
3.1.1.1. gpio_pin_init

Full diagram

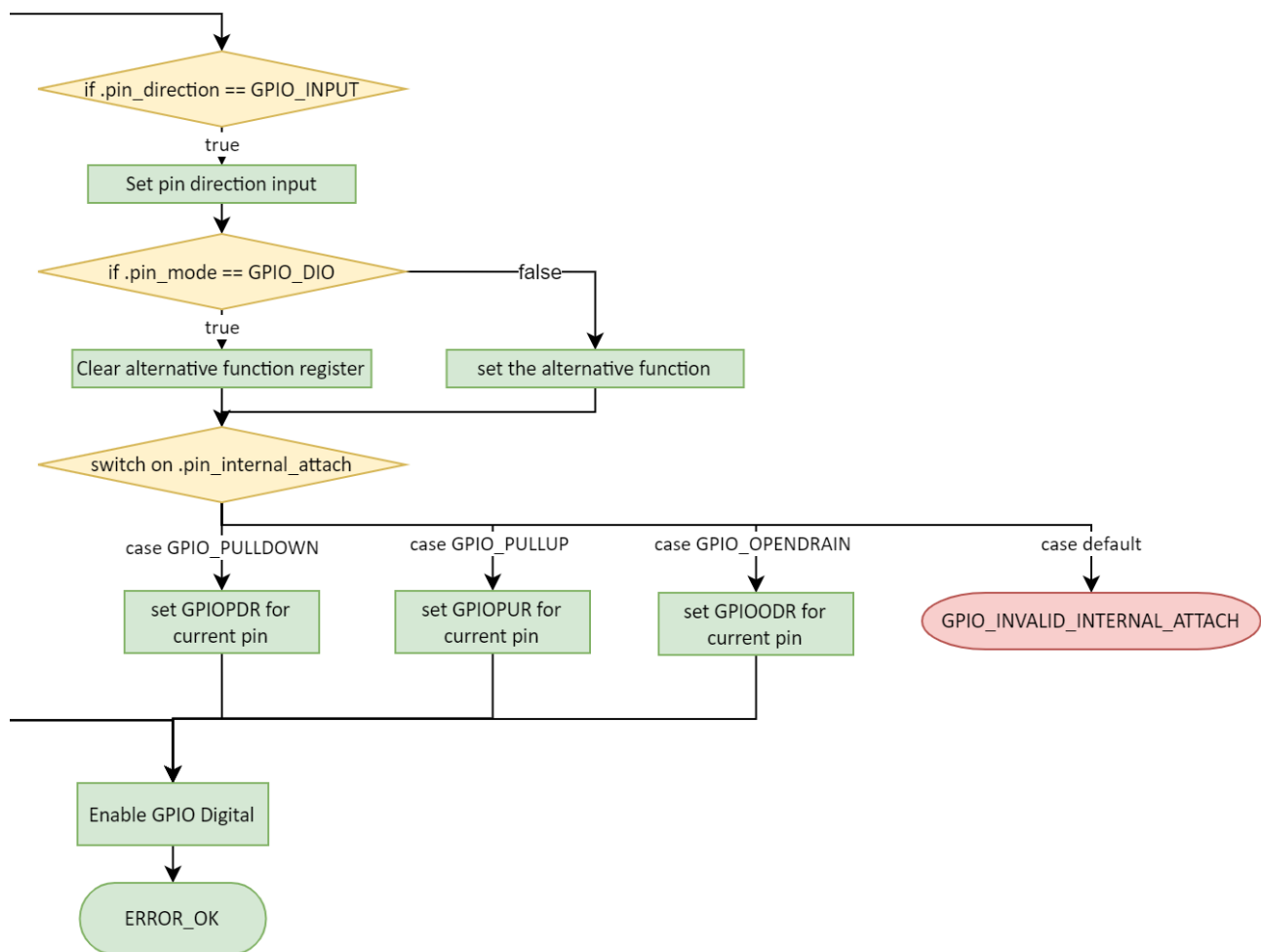
Part 1 and part 2 in the next pages



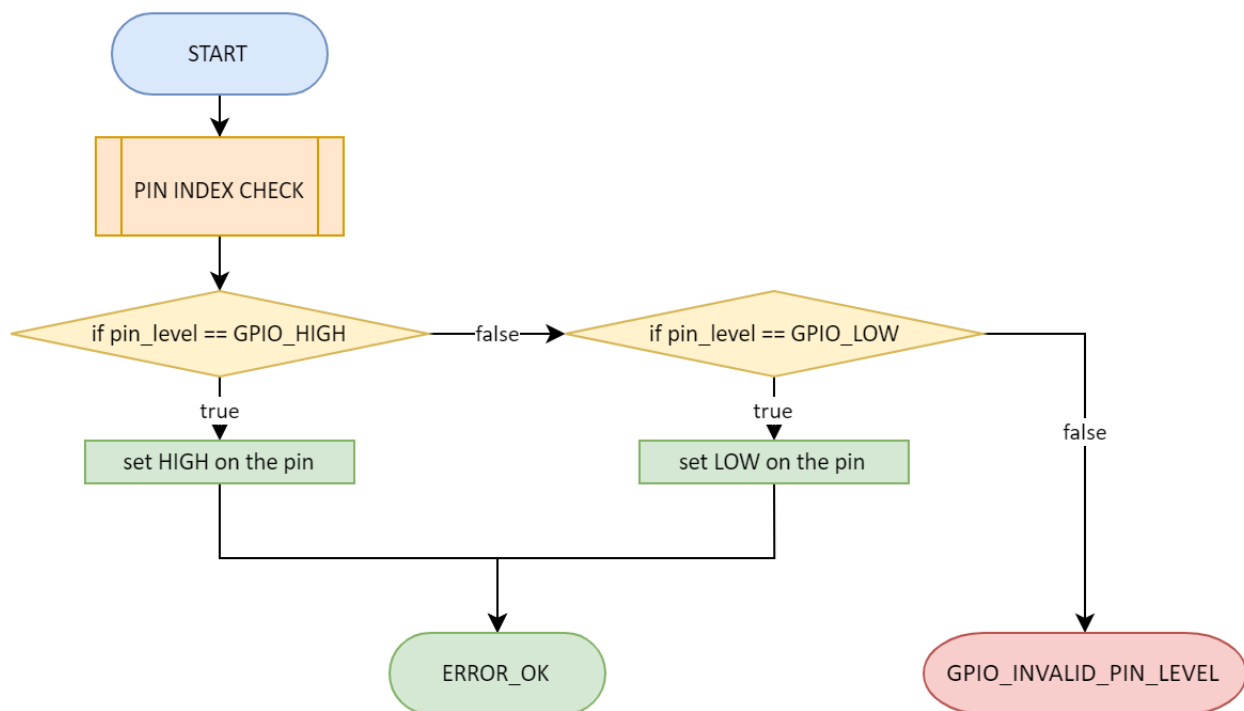
Part 1



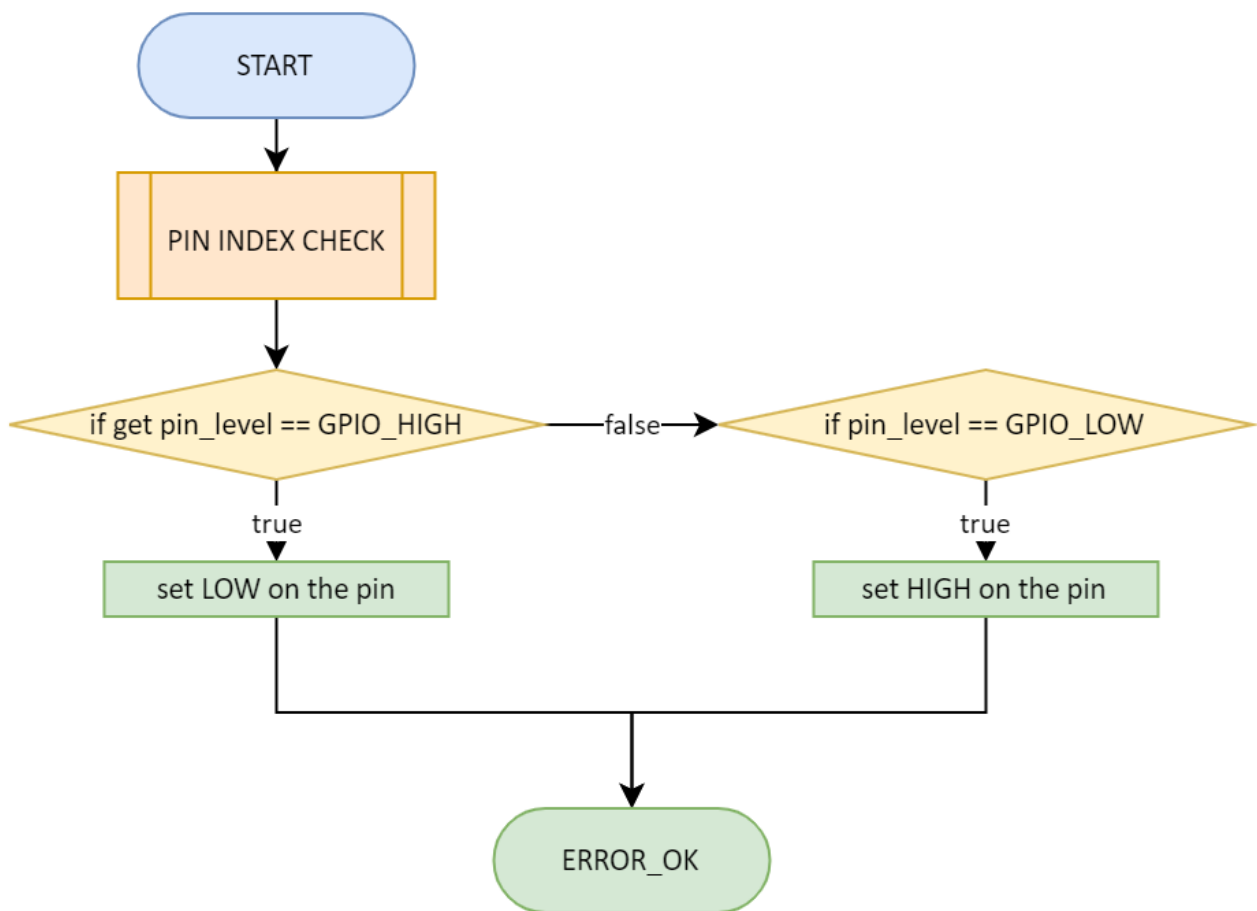
Part 2



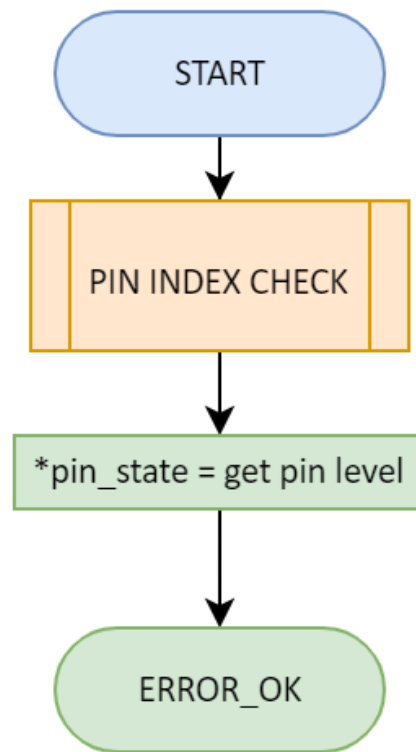
3.1.1.2. gpio_pin_write



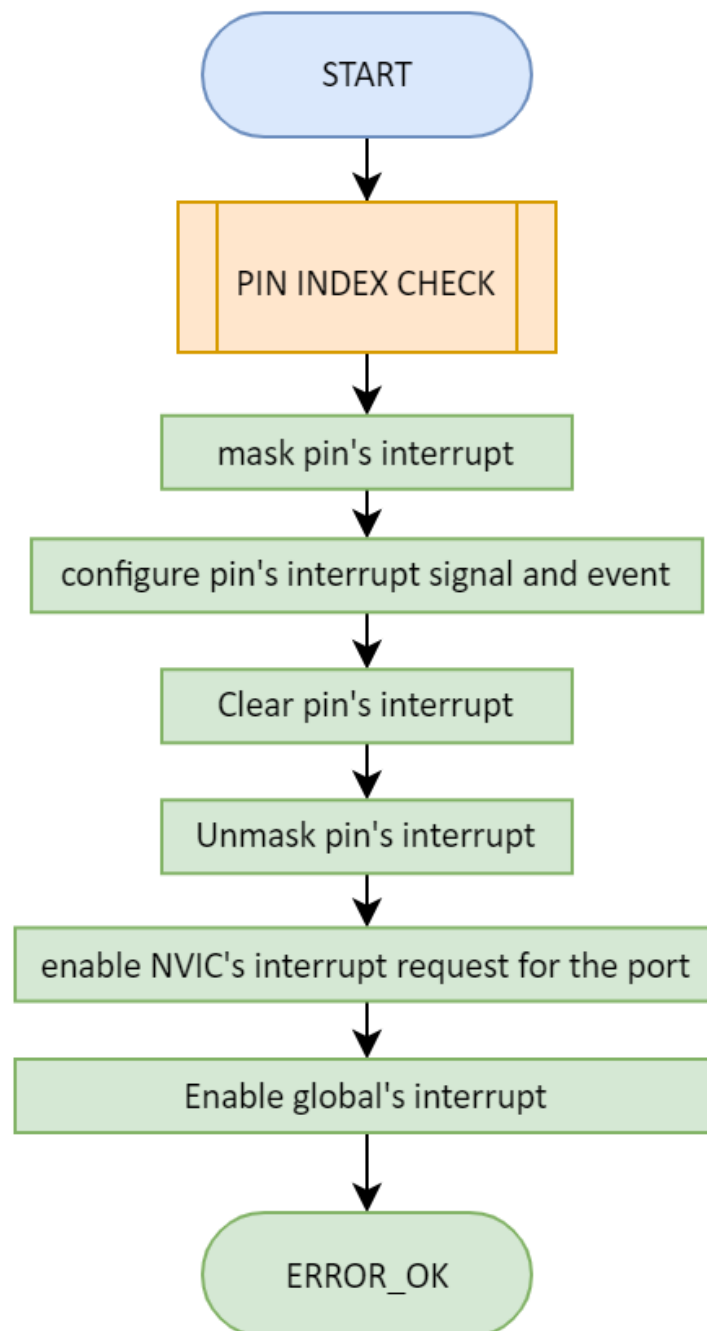
3.1.1.3. gpio_pin_toggle



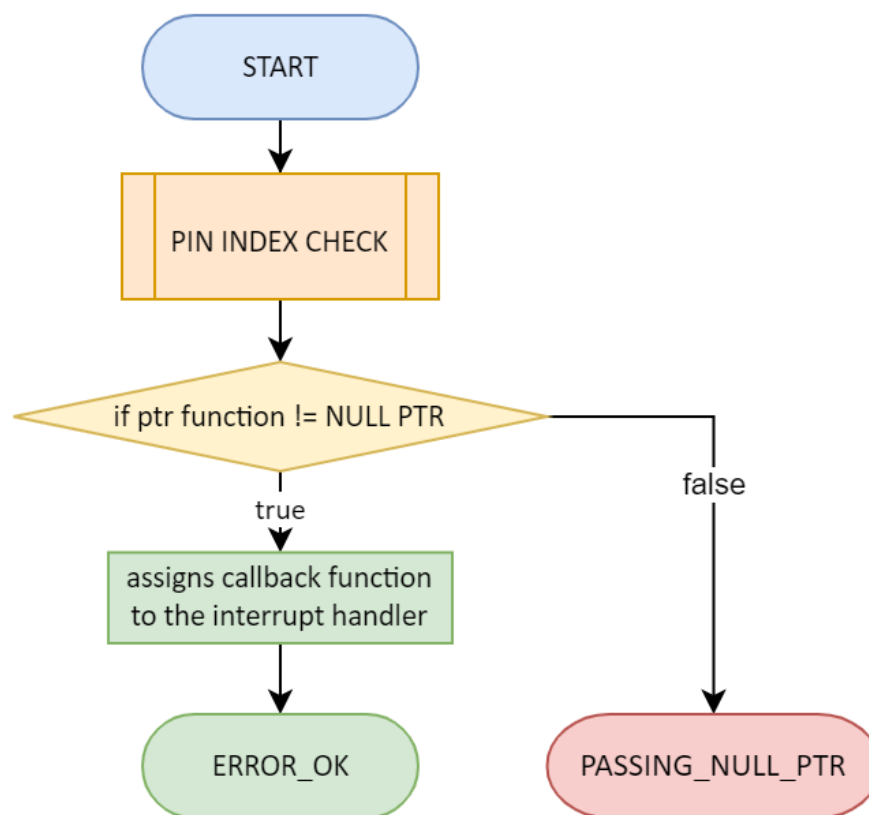
3.1.1.4. gpio_pin_read



3.1.1.5. gpio_pin_enable_notification

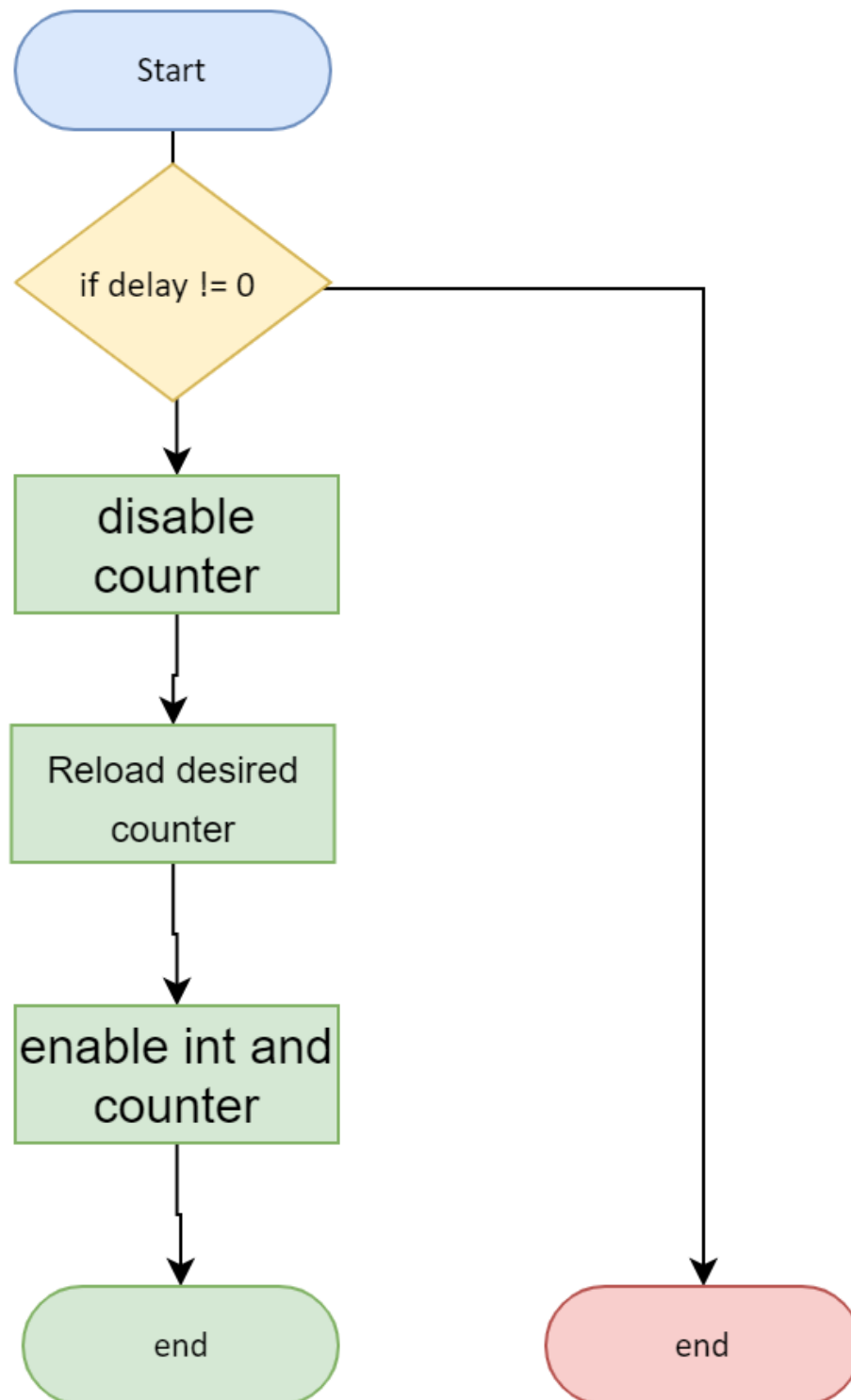


3.1.1.6. gpio_pin_set_callback

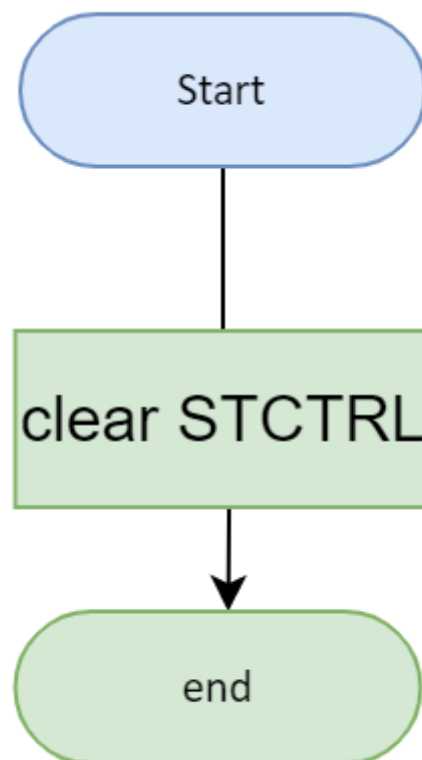


3.1.2. systick Module

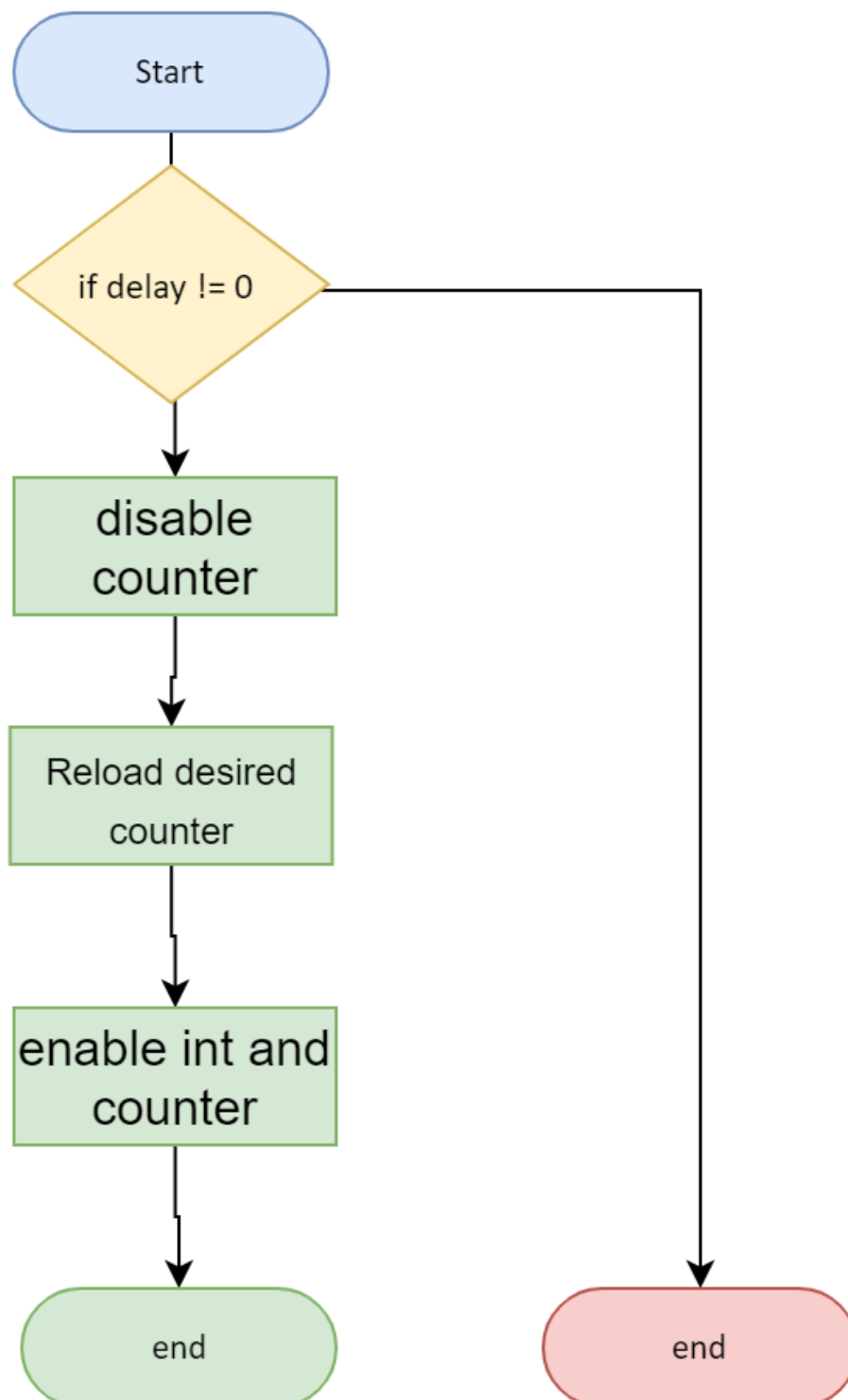
3.1.1.1. Sysstick_init



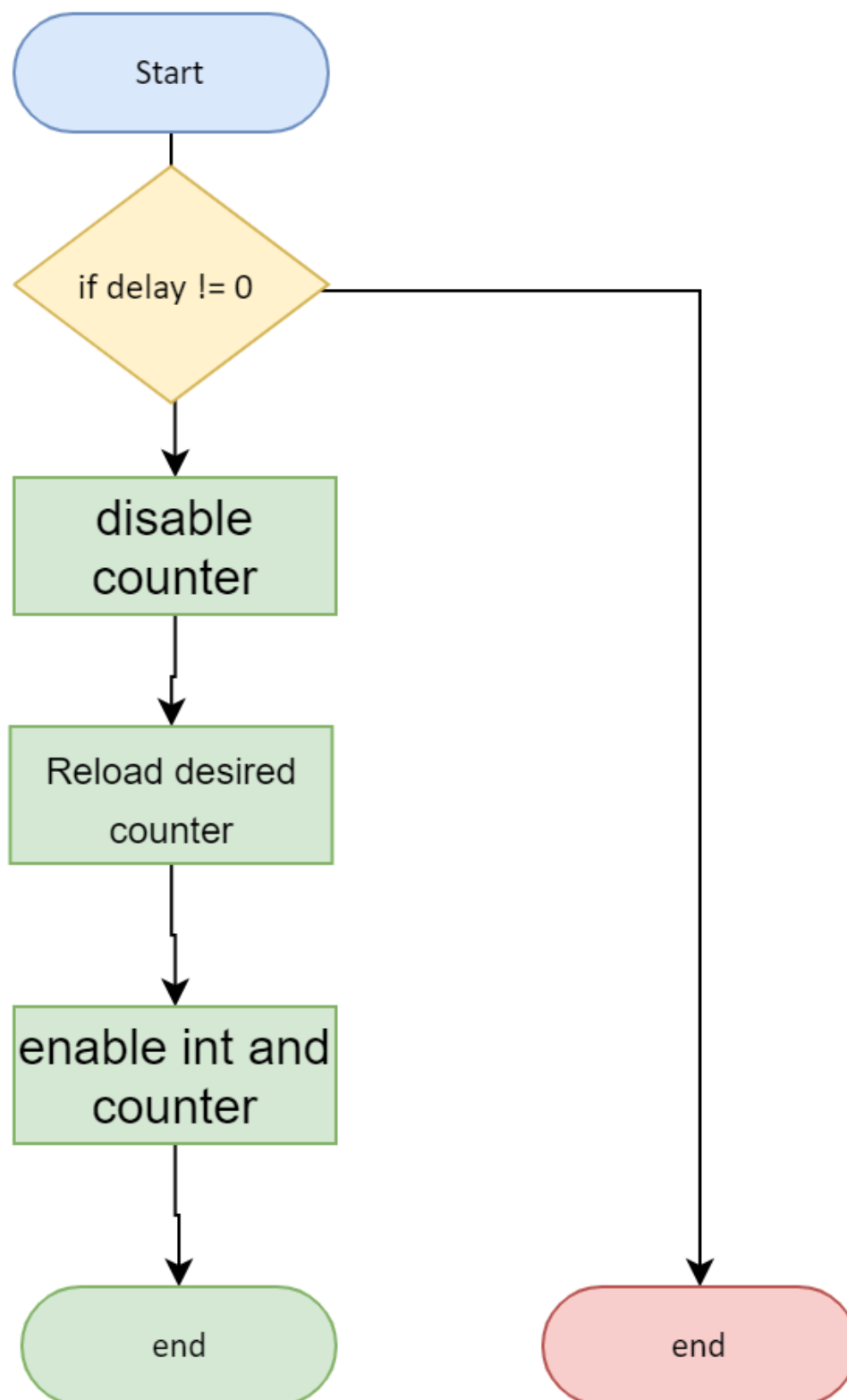
3.1.1.2. SysTick_disable



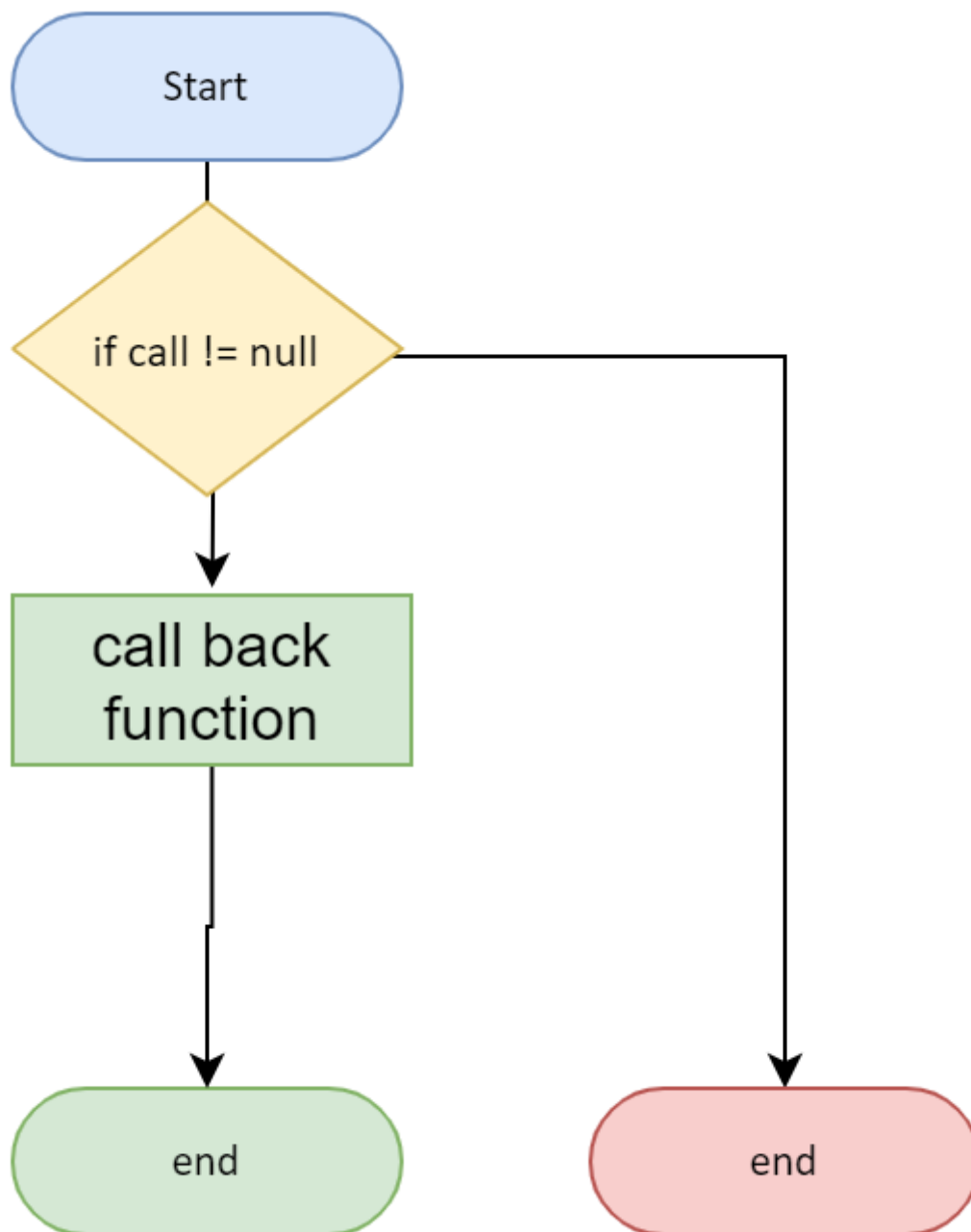
3.1.1.3. Systick_reload



3.1.1.4. systick_enable



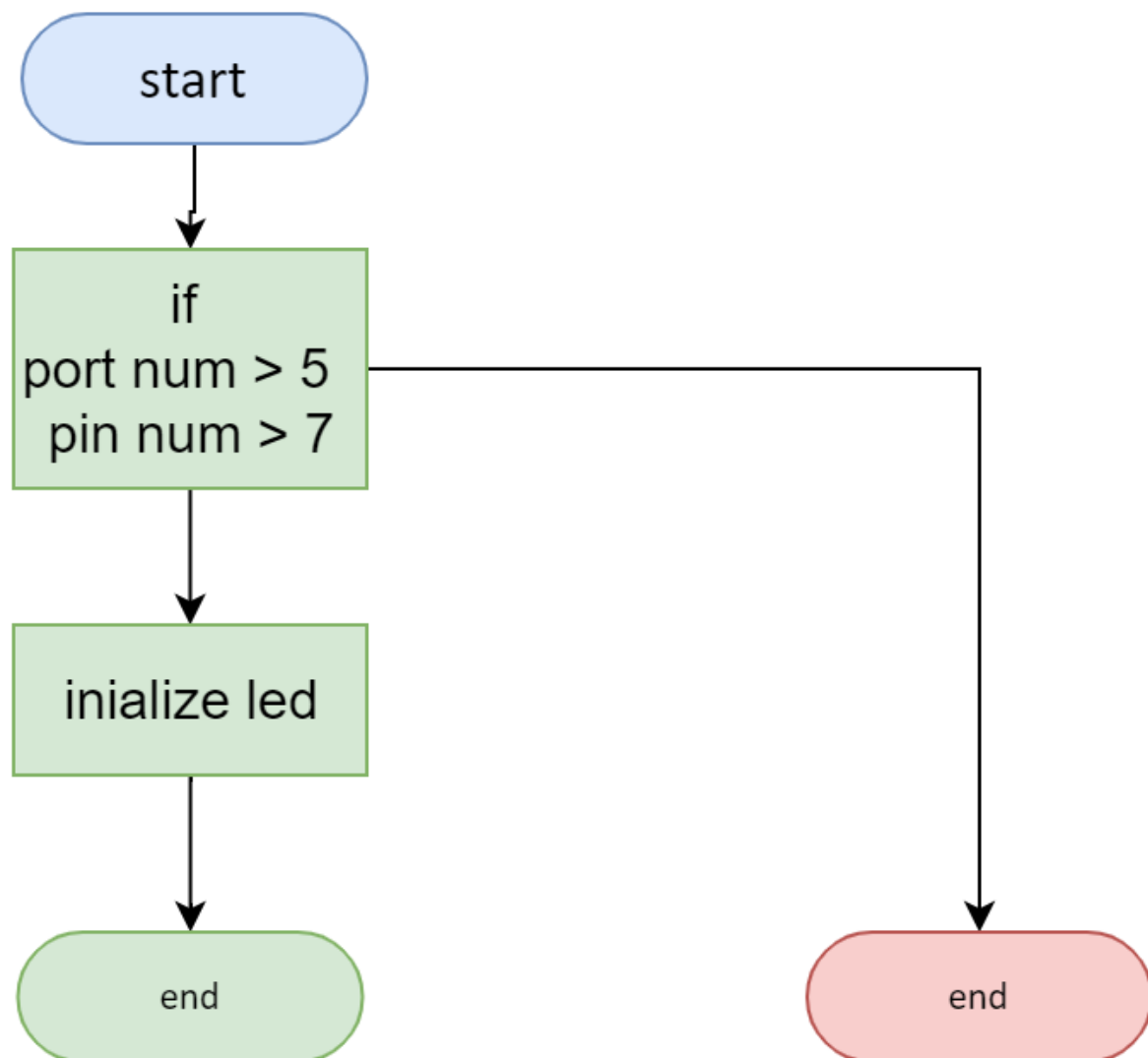
3.1.1.5. SysTick_Handler



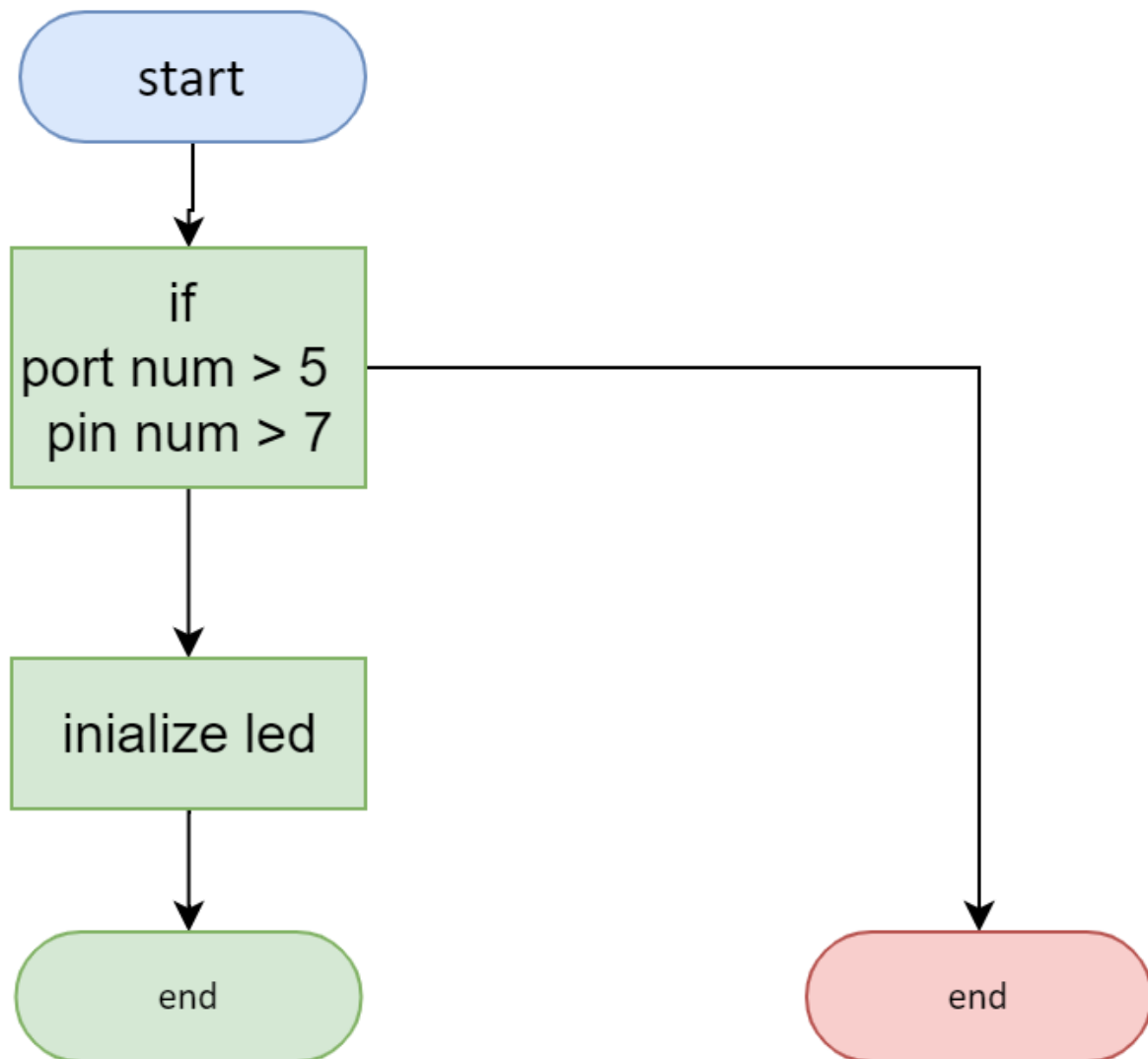
3.2. HAL Layer

3.2.1. LED Module

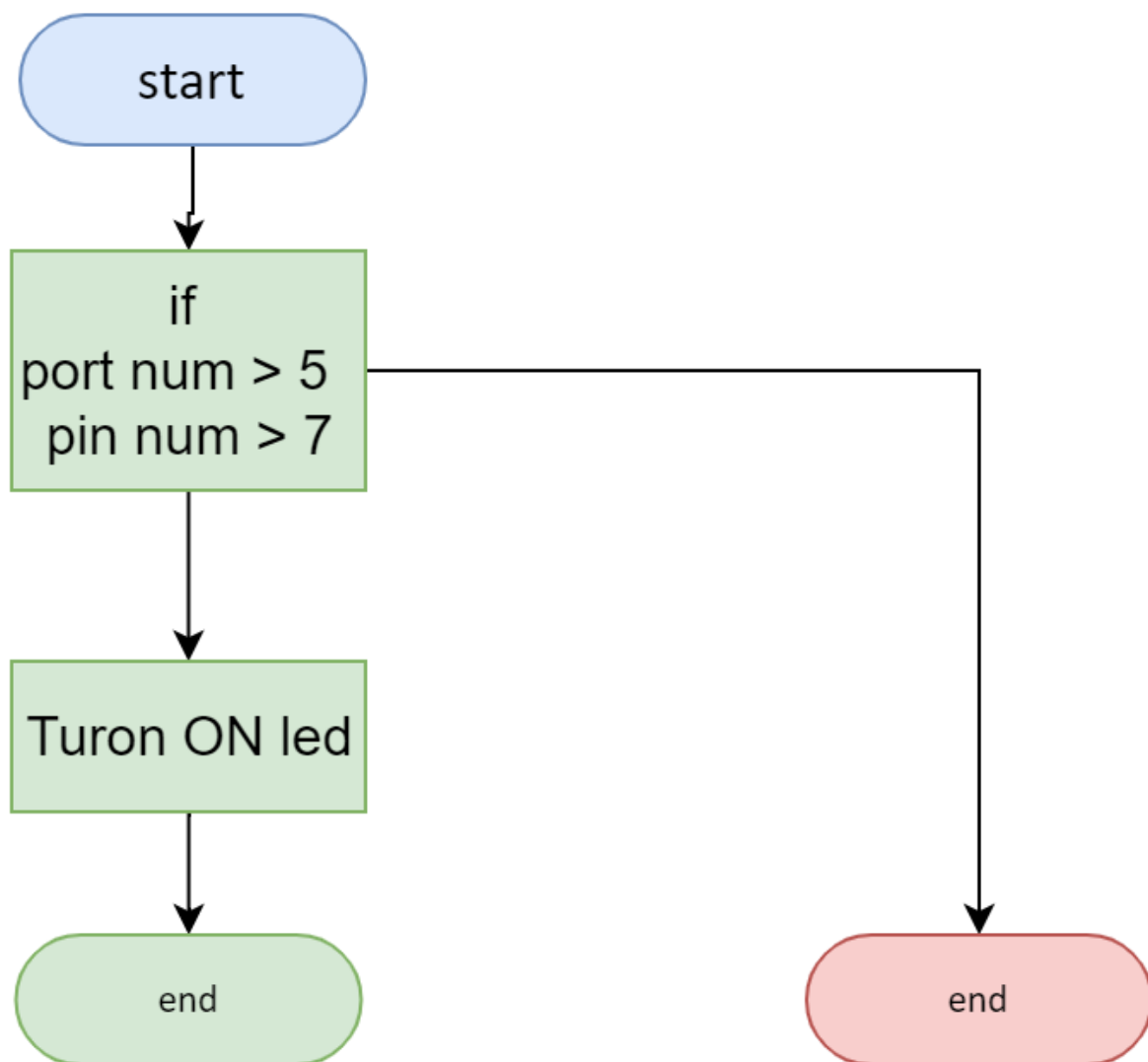
3.2.1.1. Led_init



3.2.1.2. Led_on

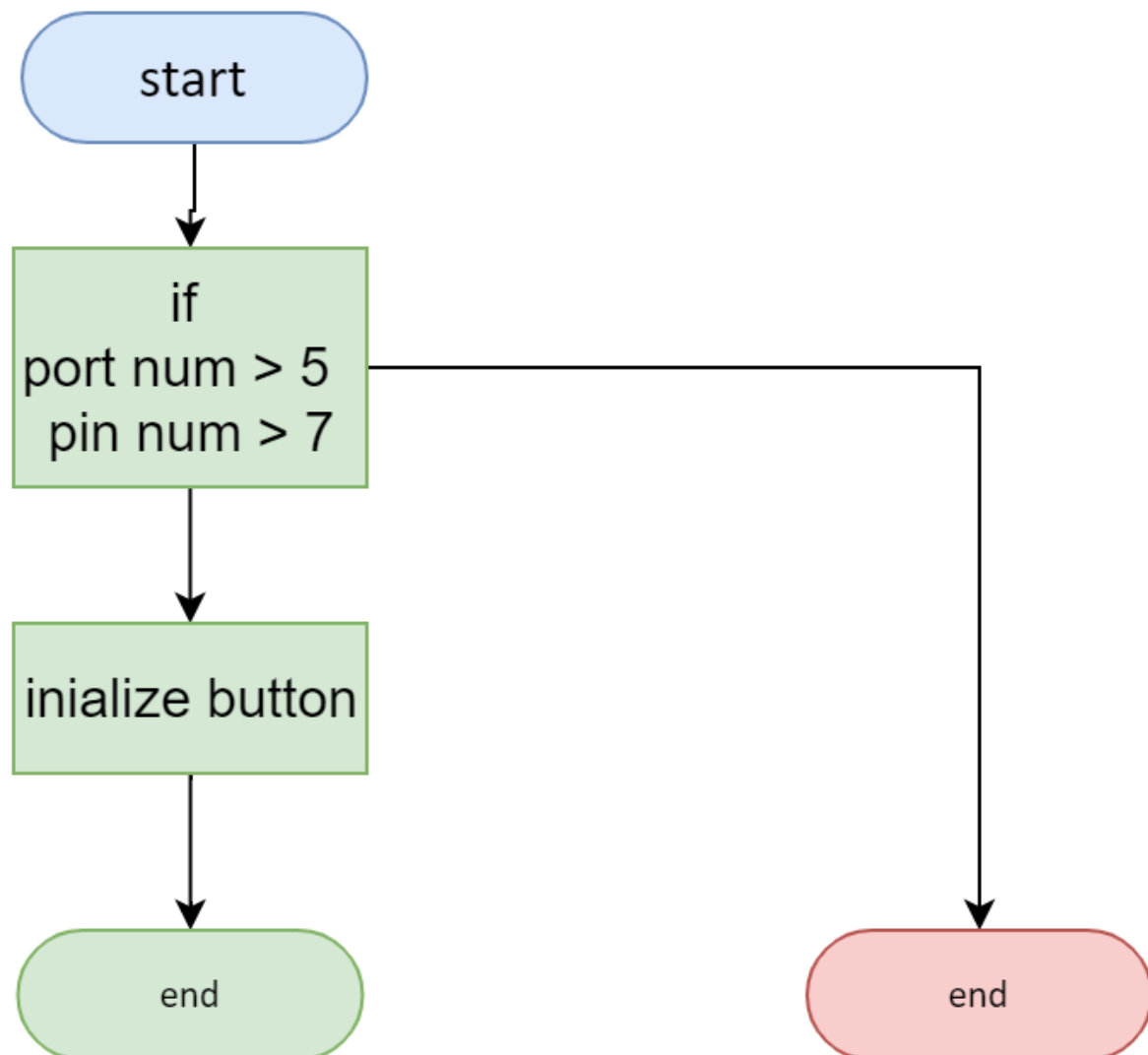


3.2.1.3. Led_off



3.2.2. BTN Module

3.2.2.1. Button_init



3.2.2.2. Button_get_state

