LED CONTROL v3

AHMED HESHAM – SARAH MOHAMED
EMBEDDED AUTOMOTIVE BOOTCAMP
6/23/23

LED CONTROL DESIGN v3

1. Project Introduction

The project to develop GPIO driver and GPT 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

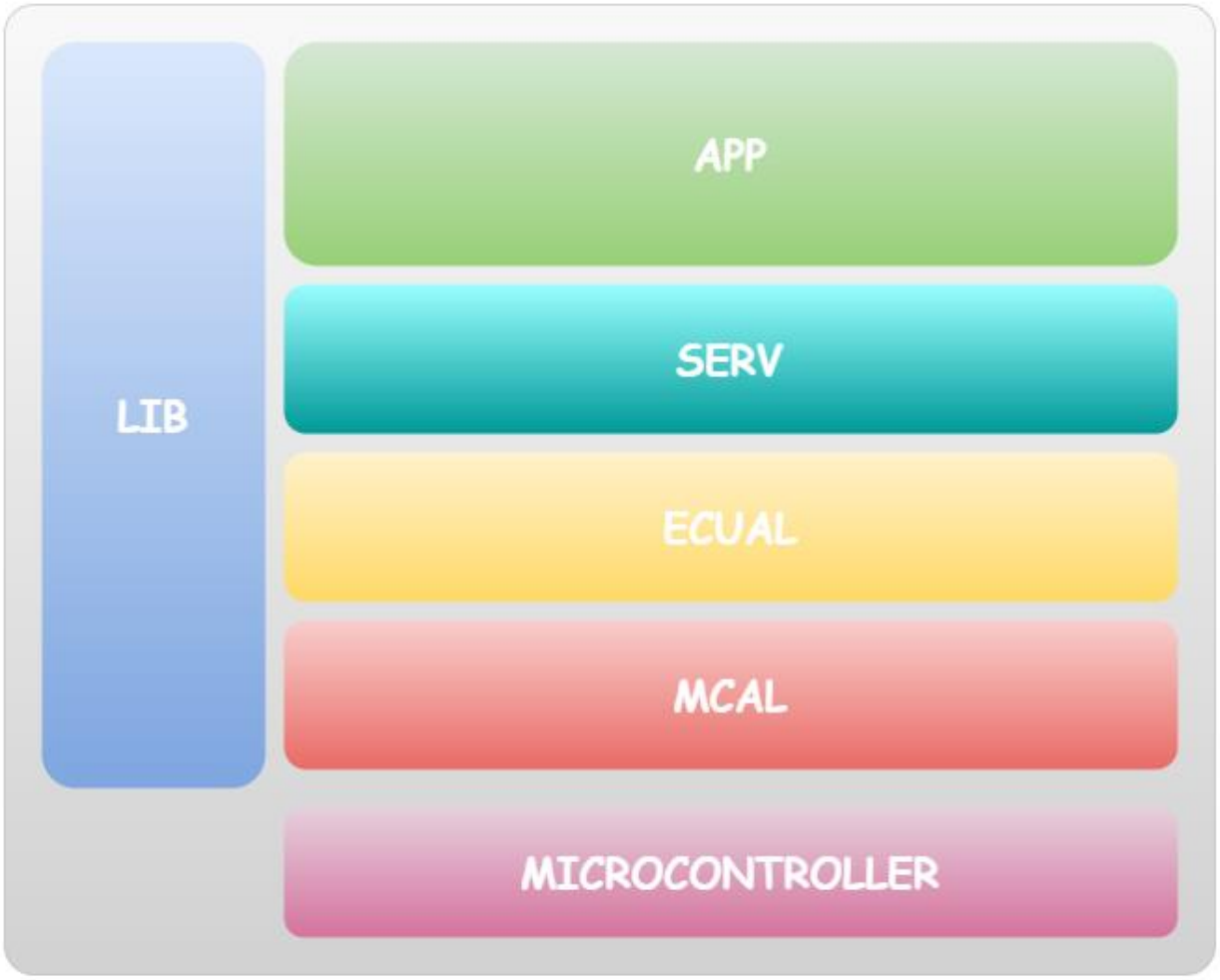
1.2. Project Requirements

1. The RGB LED is OFF initially
2. The PWM signal has a 500ms duration
3. The system has four states
    1. SW1 - First press
        1. The Green LED will be on with a 30% duty cycle
    2. SW1 - Second press
        1. The Green LED will be on with a 60% duty cycle
    3. SW1 -Third press
        1. The Green LED will be on with a 90% duty cycle
    4. SW1 - Fourth press will be off
        1. The Green LED will be off
    5. On the fifth press, system state will return to state 1

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. GPT

Programmable timers can be used to count or time external events that drive the Timer input pins. The TM4C123GH6PM General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks. Each 16/32-bit GPTM block provides two 16-bit timers/counters
(referred to as Timer A and Timer B) that can be configured to operate independently as timers or event counters, or concatenated to operate as one 32-bit timer or one 32-bit Real-Time Clock (RTC).
Each 32/64-bit Wide GPTM block provides 32-bit timers for Timer A and Timer B that can be concatenated to operate as a 64-bit timer.

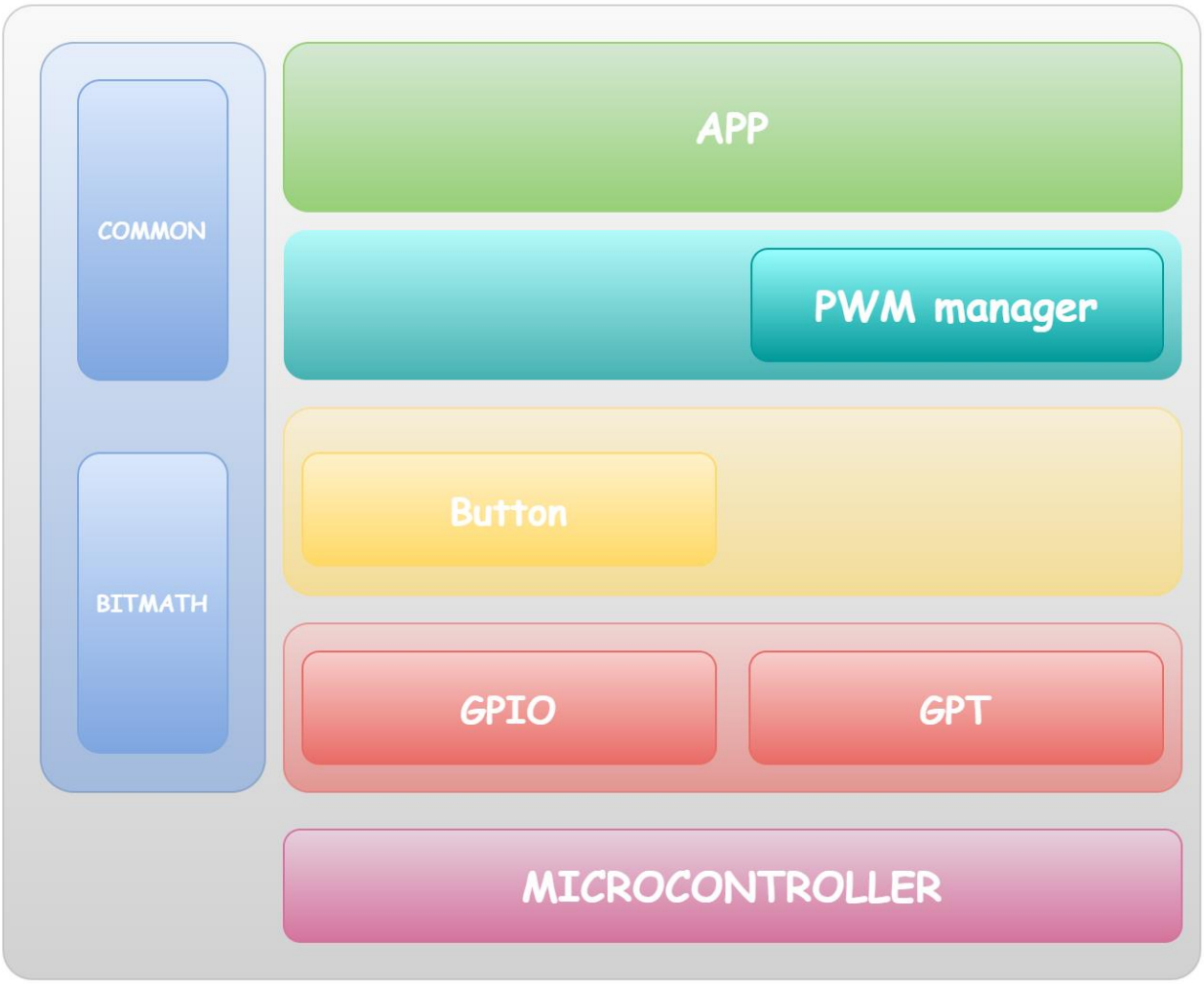
2.2.3. PWM Manager

PWM Manager is used to initialize an output pin to output PWM signal using normal timers generated by GPT.

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


COMMON
BITMATH
APP
PWM manager
Button
GPIO
GPT
MICROCONTROLLER

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
|                     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 funtion 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));
```

2.3.2.3. Timer Driver

```
| Brief
|      This function is used to initialize a specific general purpose timer
| Parameters
|       [in]  ptr_str_gpt_timer_config : Ptr to |the timer instance structure
|       [out] none
| Return
|             ERROR_OK : In case of success
|             GPT_INVALID_TIMER_INDEX : In case of choosing wrong channel index
|             GPT_INITIALIZED : In case of choosing initialized channel
|             GPT_INVALID_STANDARD : In case of choosing wrong standard
|             GPT_INVALID_BLOCKING_MODE : In case of choosing wrong blocking mode
|             GPT_INVALID_COUNTING_DIRECTION : In case of choosing wrong blocking mode
|             GPT_INVALID_GPT_MODE : In case of choosing wrong mode
|
enu_error_status_t_ gpt_init(str_gpt_timer_config_t_* ptr_str_gpt_timer_config);

| Brief
|      This function is used to stop a specific general purpose timer
| Parameters
|      [in]   enu_timer_use : Timer index used
|      [out]  none
| Return
|             ERROR_OK : In case of success
|             GPT_INVALID_TIMER_INDEX : In case of wrong pin index
|
enu_error_status_t_ gpt_stop_timer(enu_gpt_timer_used_t_ enu_timer_used);

| Brief
|      This function is used to load timer interval
| Parameters
|      [in]   enu_timer_used  : Timer index used
|             enu_counting_standard : Counting Standard -Seconds, mSeconds, uSeconds-
|             uint32_desired_time : Value to be loaded
|      [out]  none
| Return
|             ERROR_OK : In case of success
|             GPT_INVALID_TIMER_INDEX : In case of wrong pin index
|             GPT_INVALID_VALUE : In case of loading Zero
|             GPT_INVALID_STANDARD : In case of choosing a wrong standard
|
enu_error_status_t_ gpt_set_time(enu_gpt_timer_used_t_ enu_timer_used,
enu_gpt_counting_standard_t_ enu_counting_standard, uint64_t_ uint64_desired_time);

| Brief
|      This function is used to start a specific general purpose timer
| Parameters
|      [in]   enu_timer_used : Timer index used
|      [out]  none
| Return
|             ERROR_OK : In case of success
|             GPT_INVALID_TIMER_INDEX : In case of wrong pin index
|
enu_error_status_t_ gpt_start_timer(enu_gpt_timer_used_t_ enu_timer_used);
```

```
| Brief
|      This function is used to enable notification from a specific general purpose timer
| Parameters
|      [in]   enu_timer_used : Timer index used
|      [out]  none
| Return
|             ERROR_OK : In case of success
|             GPT_INVALID_TIMER_INDEX : In case of wrong pin index
|
enu_error_status_t_ gpt_enable_notificatoin(enu_gpt_timer_used_t_ enu_timer_used);

| Brief
|      This function is used to disable notification from a specific general purpose timer
| Parameters
|      [in]   enu_timer_used : Timer index used
|      [out]  none
| Return
|             ERROR_OK : In case of success
|             GPT_INVALID_TIMER_INDEX : In case of wrong pin index
|
enu_error_status_t_ gpt_disable_notificatoin(enu_gpt_timer_used_t_ enu_timer_used);

| Brief
|      This function is used to set a callback function for a specific general purpose timer
| Parameters
|      [in]   enu_timer_used : Timer index used
|      [out]  none
| Return
|             ERROR_OK : In case of success
|             GPT_INVALID_TIMER_INDEX : In case of wrong pin index
|
enu_error_status_t_ gpt_set_callback(enu_gpt_timer_used_t_ enu_timer_used, ptr_gpt_callback_t_
ptr_gpt_callback);

| Brief
|      This function is used to get an index for unused channel
| Parameters
|      [in]   none
|      [out]  ptr_uint8_timer_unused : Timer index unused
| Return
|             ERROR_OK : In case of success
|             GPT_ALL_TIMERS_USED : In case of all timers are used
|
enu_error_status_t_ gpt_get_unused_channel(uint8_t_* ptr_uint8_timer_unused);
```

2.3.3. HAL APIs

2.3.3.1. LED APIs

```
|       @brief                This function is initalize pin
|       @param [in]           none
|       @return               ERROR_OK       :       In case of successeion
|       @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 successeion
|       @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 successeion
|       @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 initalizing button
|       @param [in]           ptr_func                :    pointer to callback
|       @return               ERROR_OK       :      In case of successeion
|      @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 successeion
|       @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. SERV APIs

2.3.4.1. PWM manager APIs

```
| Brief
|      This function is used to initialize a pwm pin
| Parameters
|      [in]   ptr_str_pwm_config : Ptr to the pwm instance structure
|      [out]  none
| Return
|             ERROR_OK : In case of success
|             PWM_TIMER_INIT_FAILED : In case of the timer initialization failed
|             PWM_PIN_INIT_FAILED : In case of the pin initialization failed
|             PWM_INIT_FAILED : In case of there is a timer available
|
enu_error_status_t_ pwm_init(str_pwm_config_t_* ptr_str_pwm_config);

| Brief
|      This function is used to set duty cycle
| Parameters
|      [in]   uint8_pwm_timer_index : Timer used
|             uint8_pwm_duty_cycle : desired duty cycle
|      [out]  none
| Return
|             ERROR_OK : In case of success
|             PWM_INVALID_DUTY_CYCLE : In case of choosing wrong duty cycle
|
enu_error_status_t_ pwm_set_duty_cycle(uint8_t_ uint8_pwm_timer_index, uint8_t_
uint8_pwm_duty_cycle);

| Brief
|      This function is used to set frequency used
| Parameters
|      [in]   uint8_pwm_timer_index : Timer used
|             uint32_pwm_frequency : desired frequency
|      [out]  none
| Return
|             ERROR_OK : In case of success
|             PWM_INVALID_FREQUENCY : In case of the timer initialization failed
|
enu_error_status_t_ pwm_set_frequency(uint8_t_ uint8_pwm_timer_index, uint32_t_
uint32_pwm_frequency);

| Brief
|       This function is used to start outputing pwm
| Parameters
|      [in]   uint8_pwm_timer_index : Timer used
|      [out]  none
| Return
|             ERROR_OK : In case of success
|             PWM_FAILED : In case of PWM failed to start
|
enu_error_status_t_ pwm_start(uint8_t_ uint8_pwm_timer_index);
```

```
| Brief
|      This function is used to stop outputing pwm
| Parameters
|      [in]   uint8_pwm_timer_index : Timer used
|      [out]  none
| Return
|             ERROR_OK : In case of success
|             PWM_FAILED : In case of PWM failed to stop
|
enu_error_status_t_ pwm_stop(uint8_t_ uint8_pwm_timer_index);
```

2.3.5. APP APIs

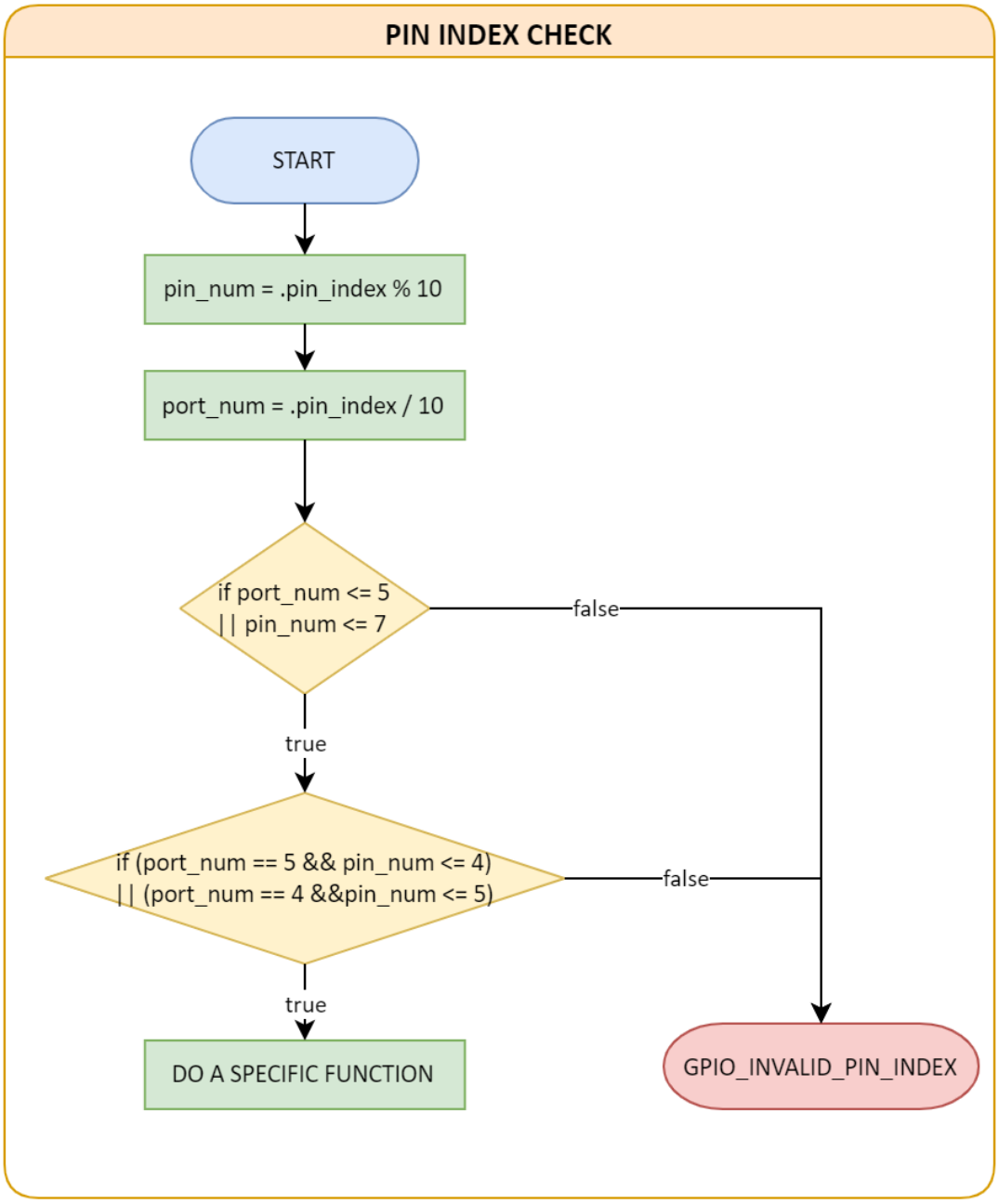
```
| Brief
|                     This function is used to initialize drivers used
| Parameters
|      [in]           none
|      [out]   none
|
| Return
|                     none
|
void app_init(void)

| 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

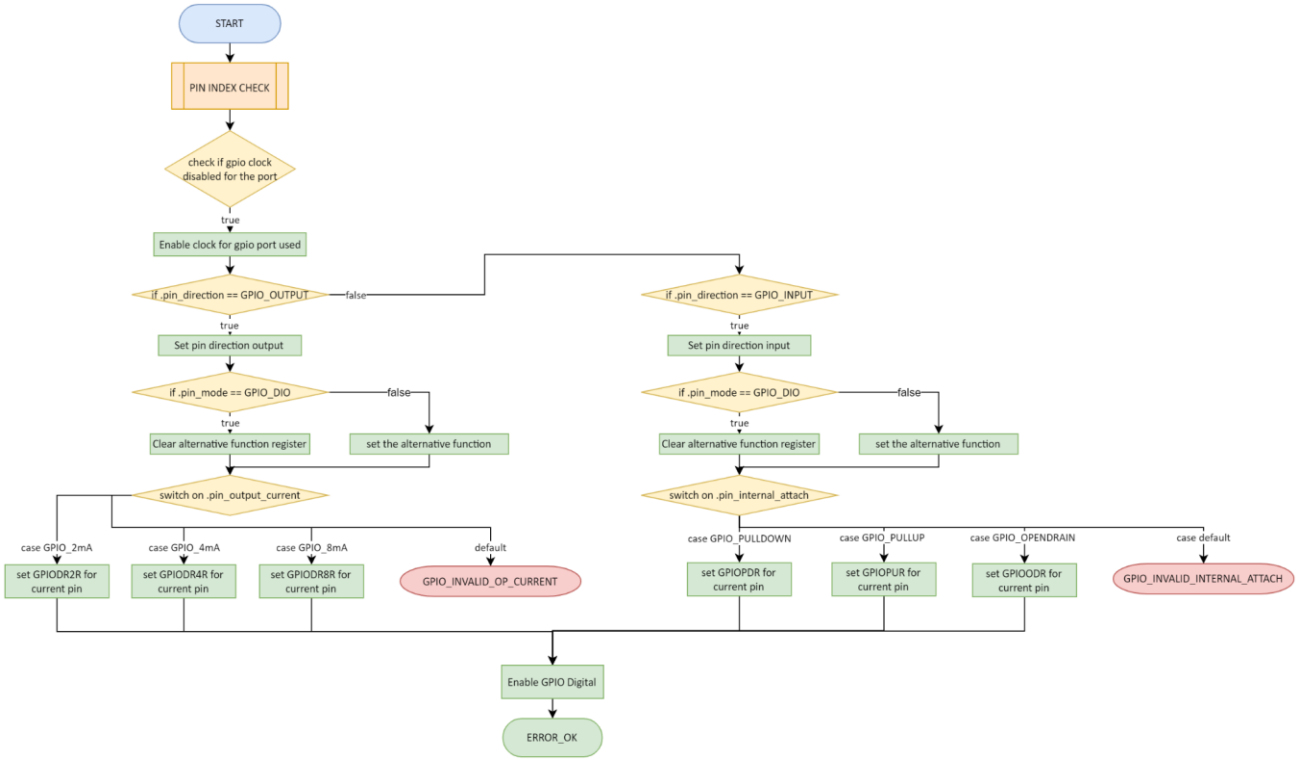
3.1.1.a. PIN INDEX CHECK sub-process



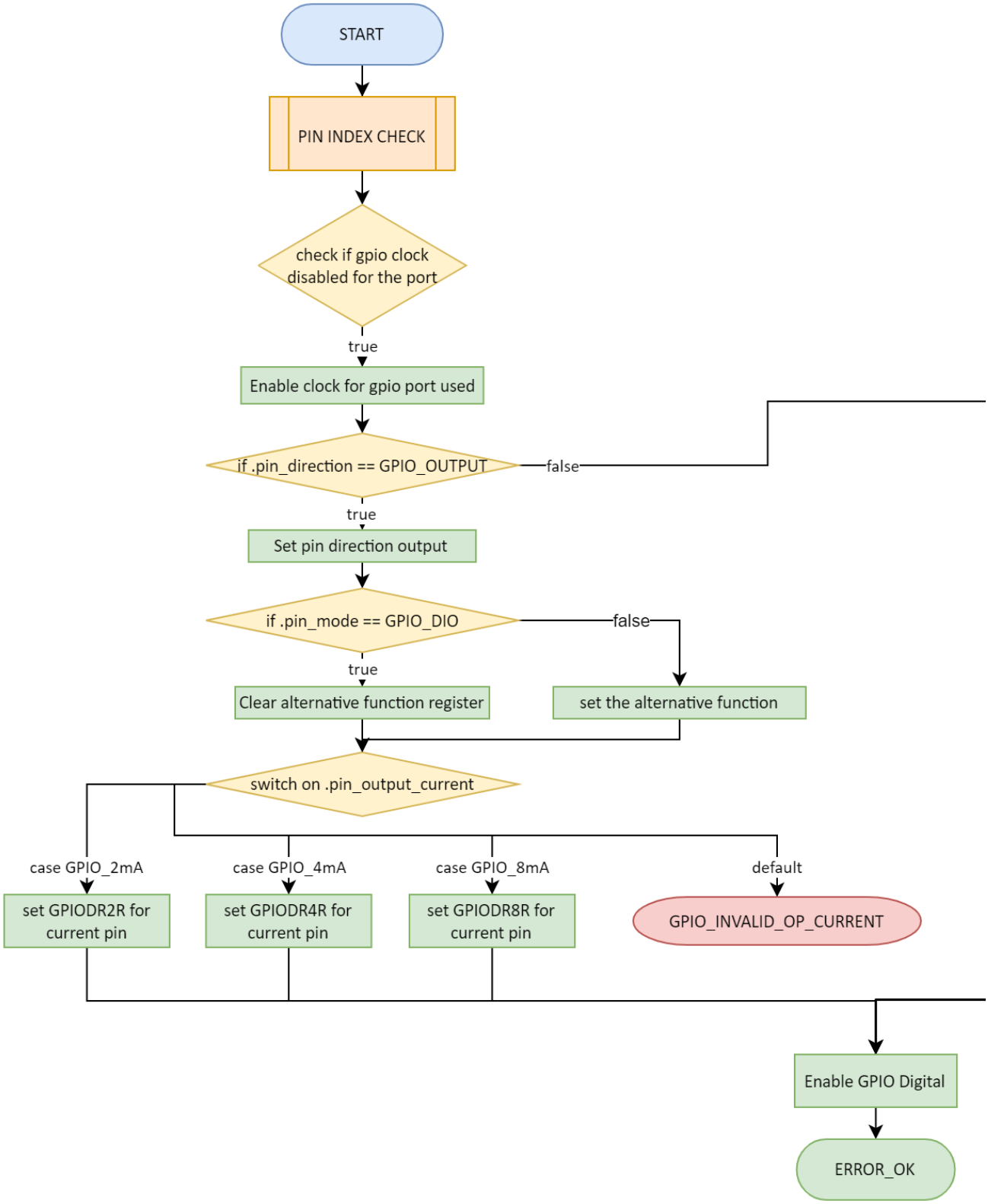
3.1.1.1. gpio_pin_init

Full diagram

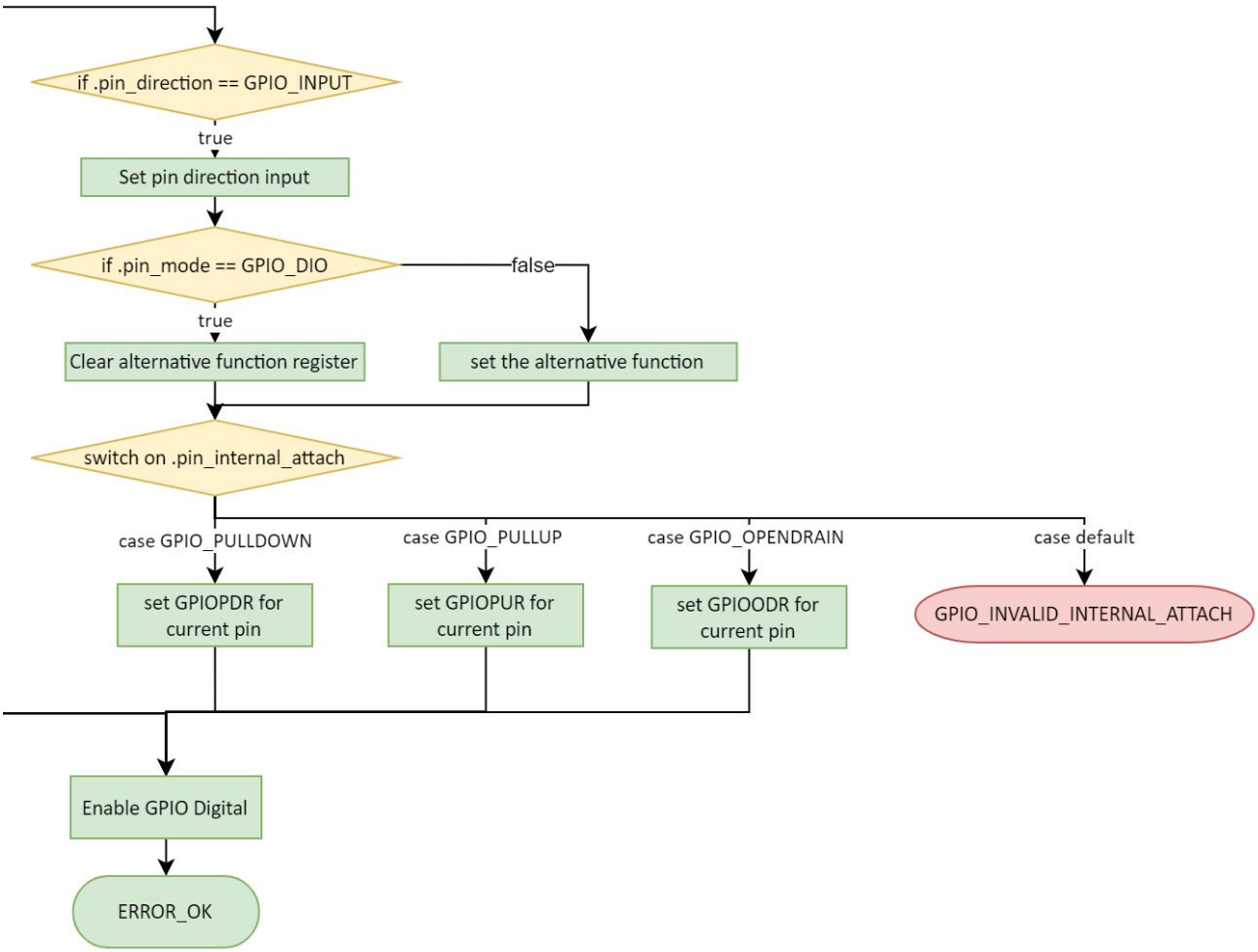
Part 1 and part 2 in the next pages



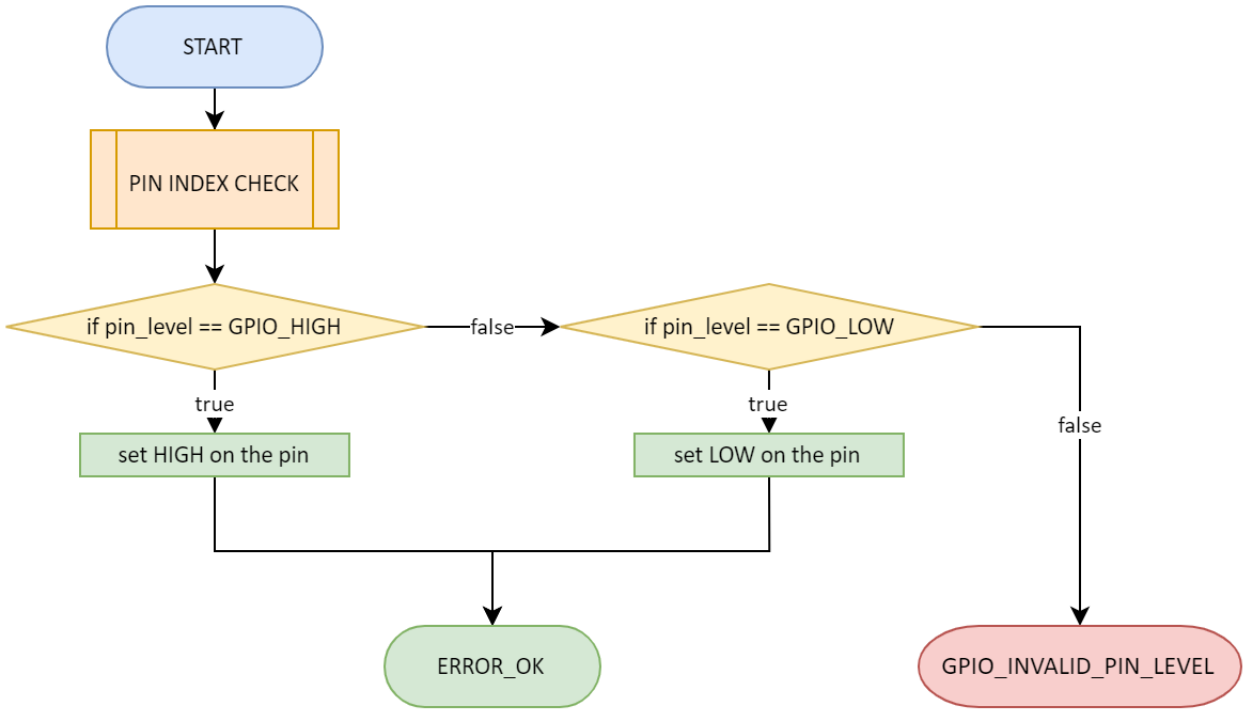
Part 1


START
PIN INDEX CHECK
check if gpio clock disabled for the port
true
Enable clock for gpio port used
if .pin_direction == GPIO_OUTPUT
false
true
Set pin direction output
if .pin_mode == GPIO_DIO
false
true
Clear alternative function register
set the alternative function
switch on .pin_output_current
case GPIO_2mA
case GPIO_4mA
case GPIO_8mA
default
set GPIODR2R for current pin
set GPIODR4R for current pin
set GPIODR8R for current pin
GPIO_INVALID_OP_CURRENT
Enable GPIO Digital
ERROR_OK

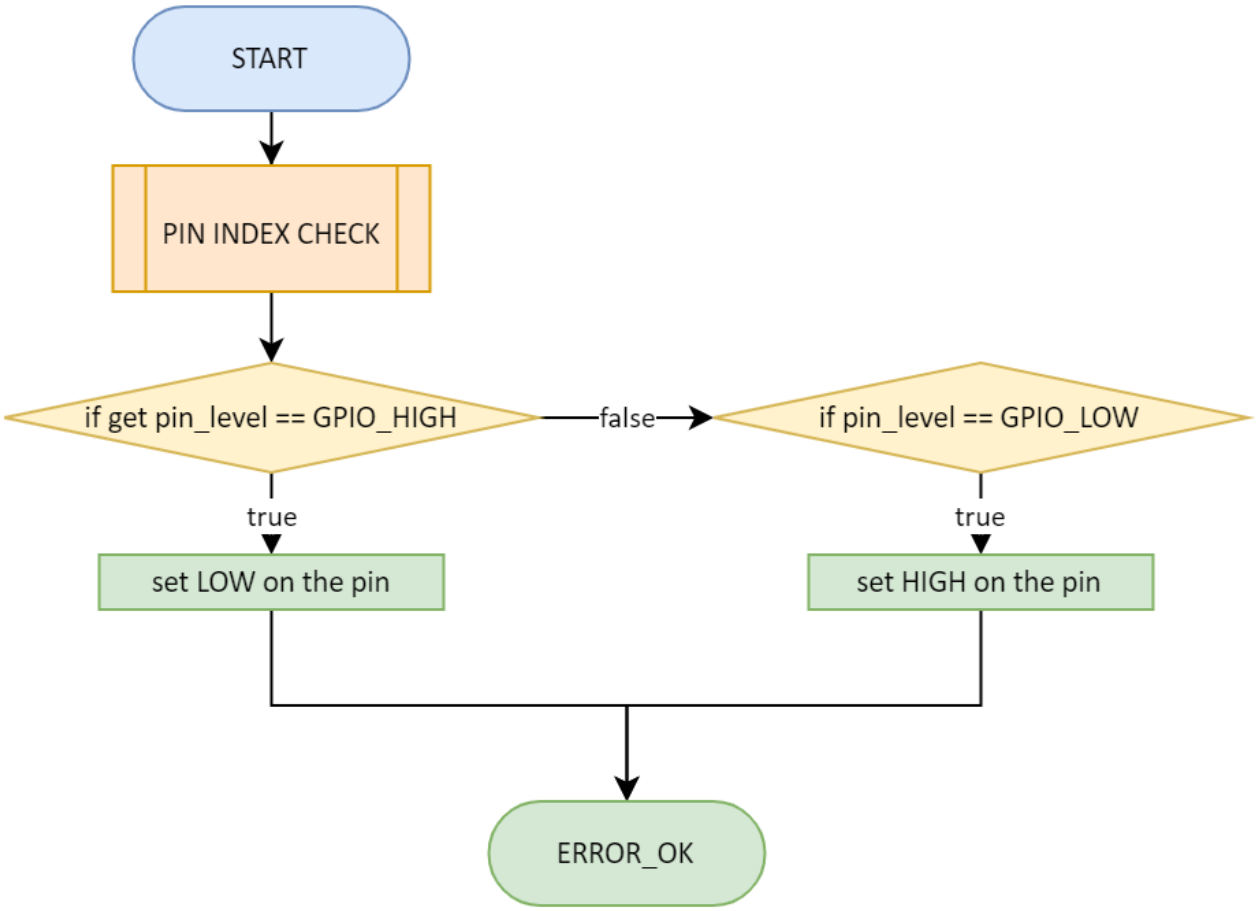
Part 2


if .pin_direction == GPIO_INPUT
true
Set pin direction input
if .pin_mode == GPIO_DIO
false
true
Clear alternative function register
set the alternative function
switch on .pin_internal_attach
case GPIO_PULLDOWN
case GPIO_PULLUP
case GPIO_OPENDRAIN
case default
set GPIOPDR for current pin
set GPIOPUR for current pin
set GPIOODR for current pin
GPIO_INVALID_INTERNAL_ATTACH
Enable GPIO Digital
ERROR_OK

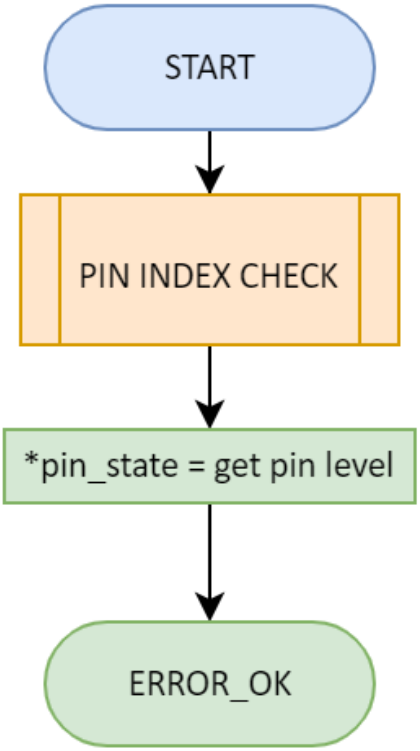
3.1.1.2. gpio_pin_write



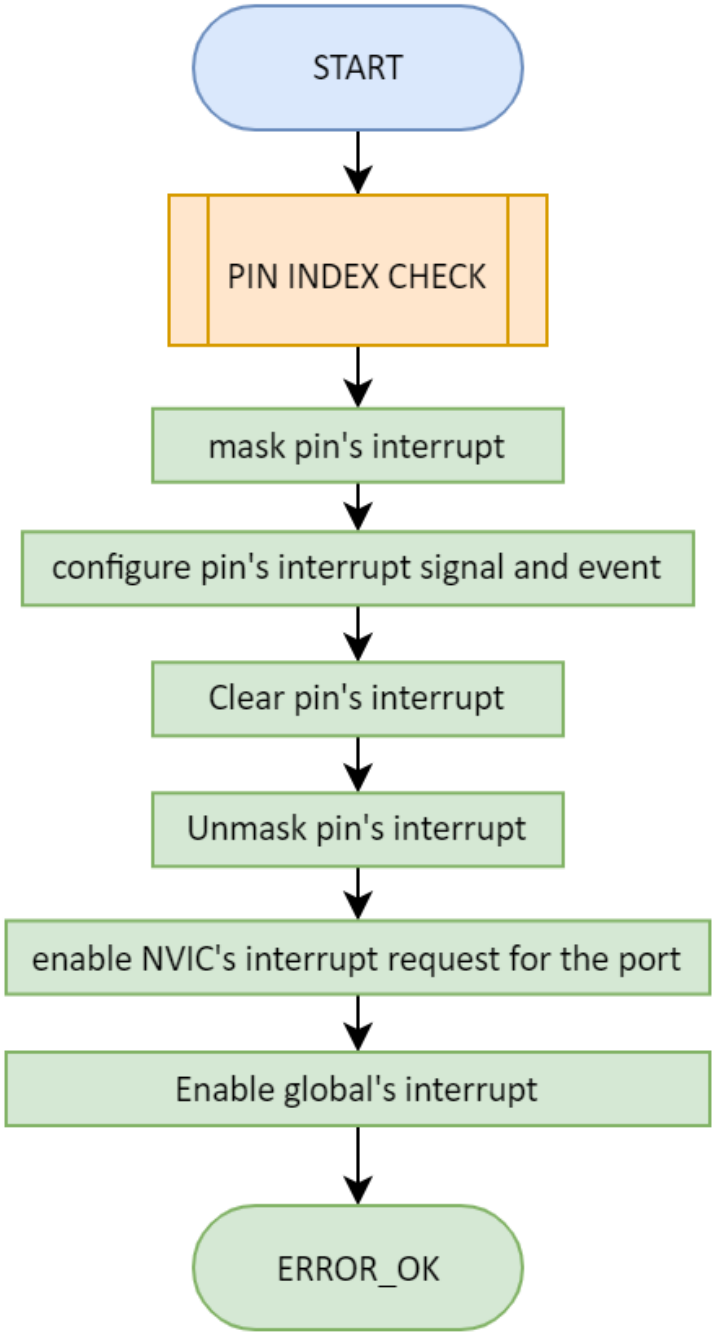
3.1.1.3. gpio_pin_toggle



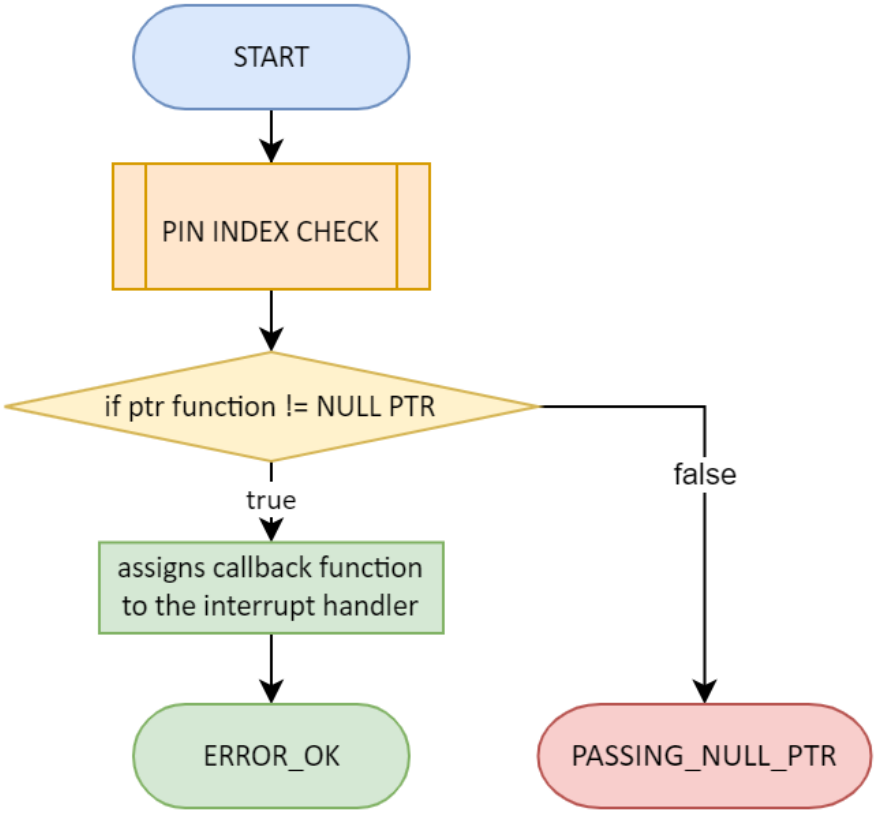
3.1.1.4. gpio_pin_read



3.1.1.5. gpio_pin_enable_notification

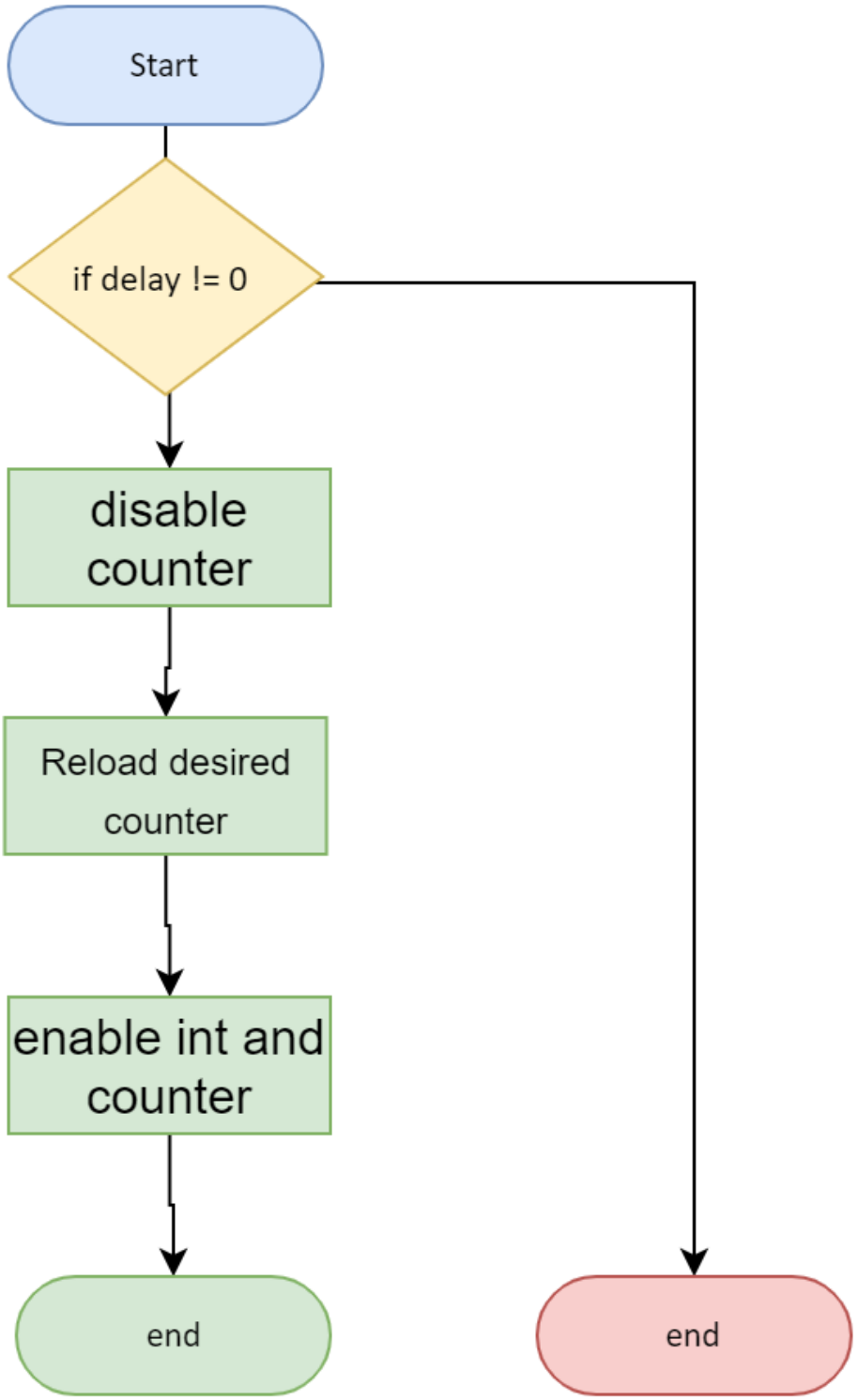


3.1.1.6. gpio_pin_set_callback

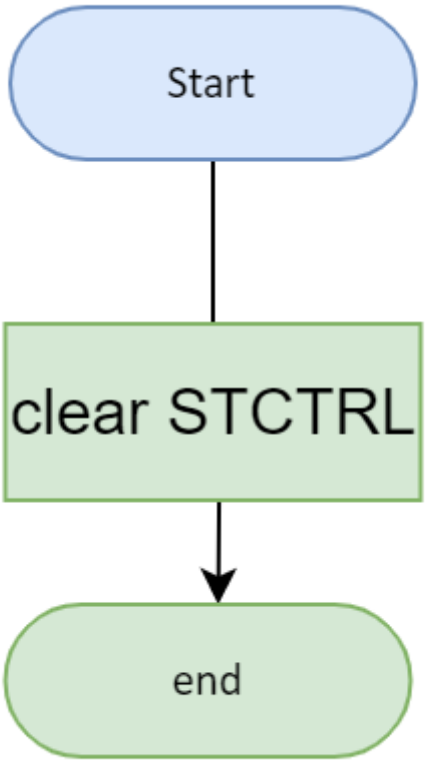


3.1.2. systick Module

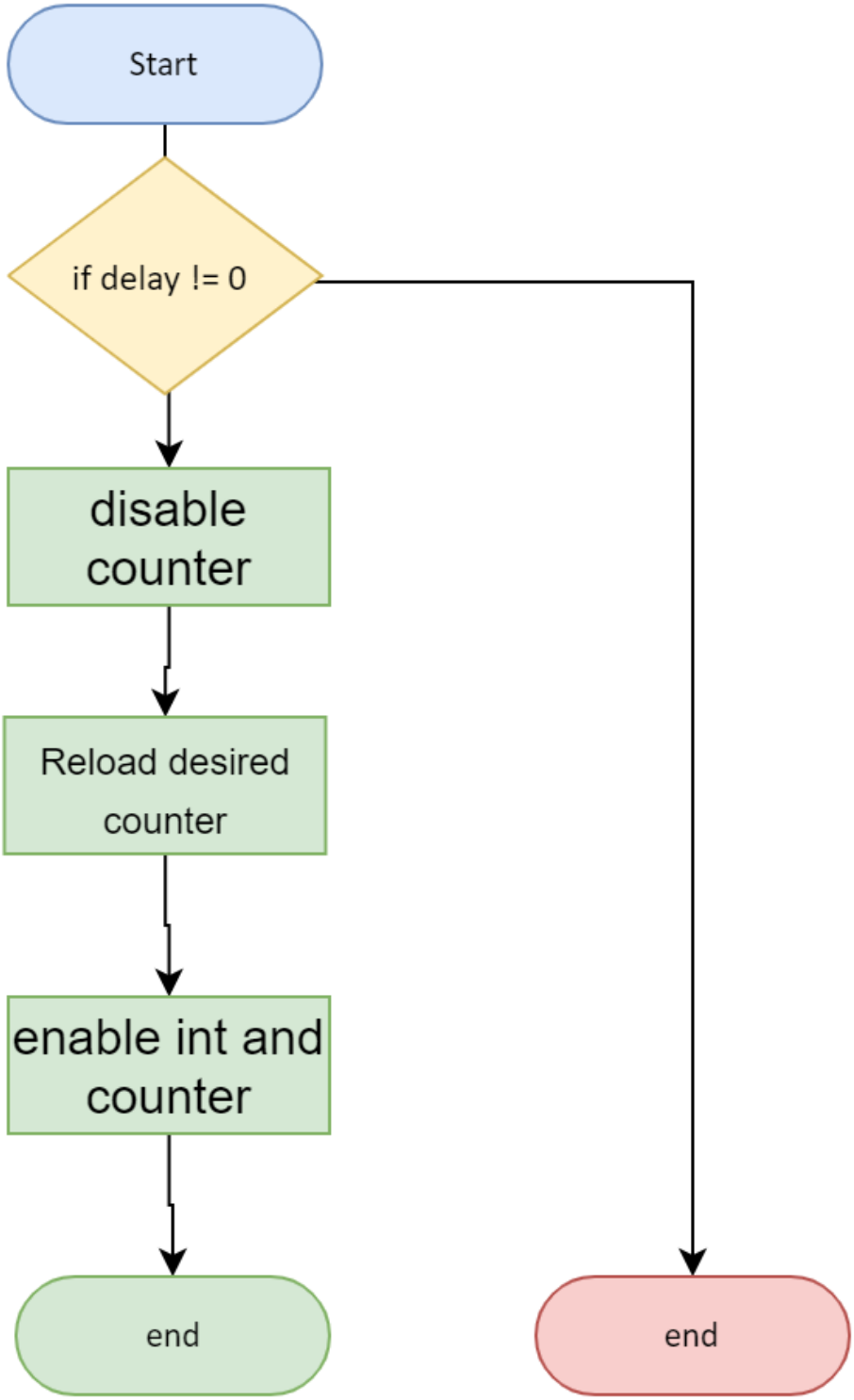
3.1.2.1. Systick_init



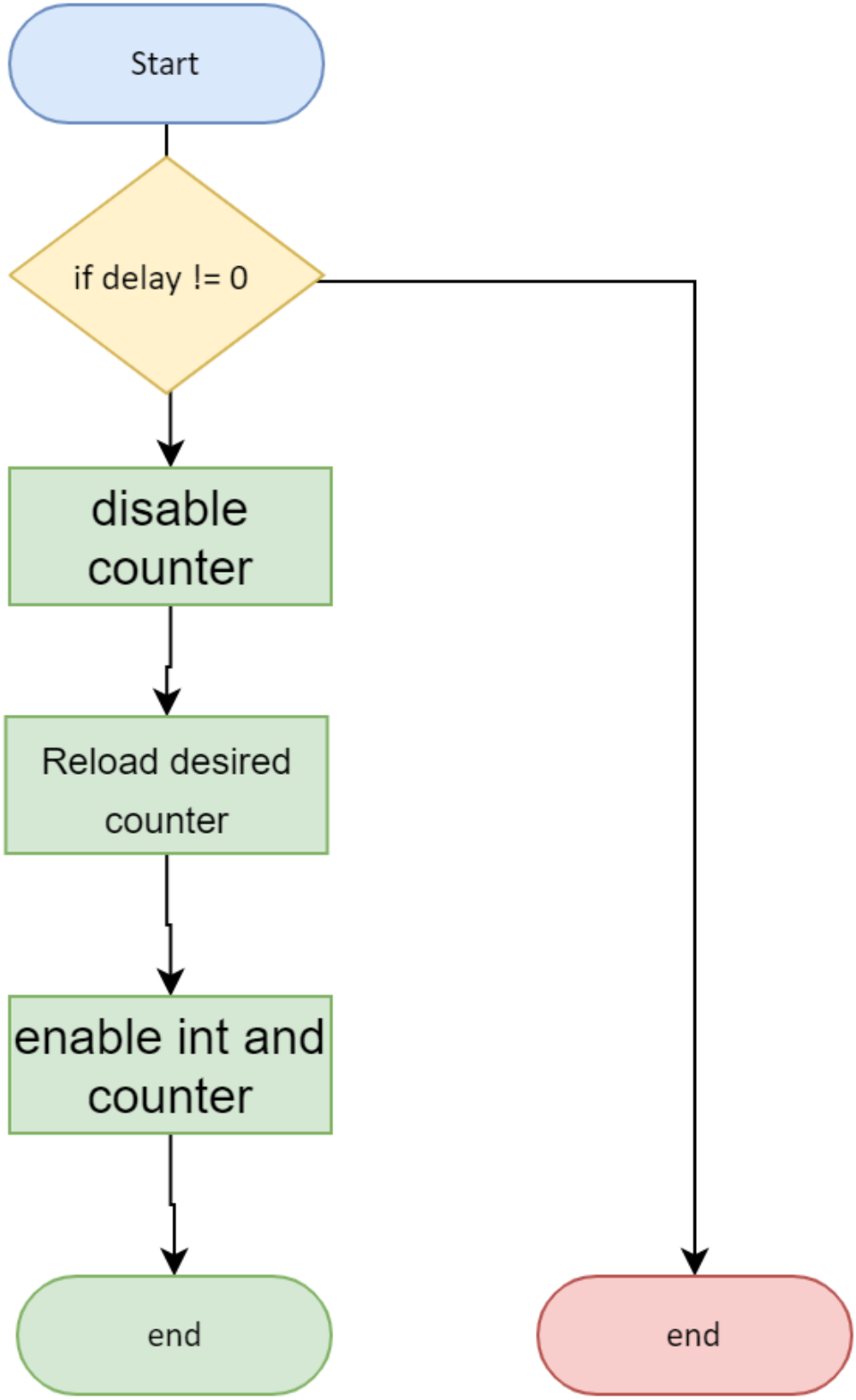
3.1.2.2. Systick_disable



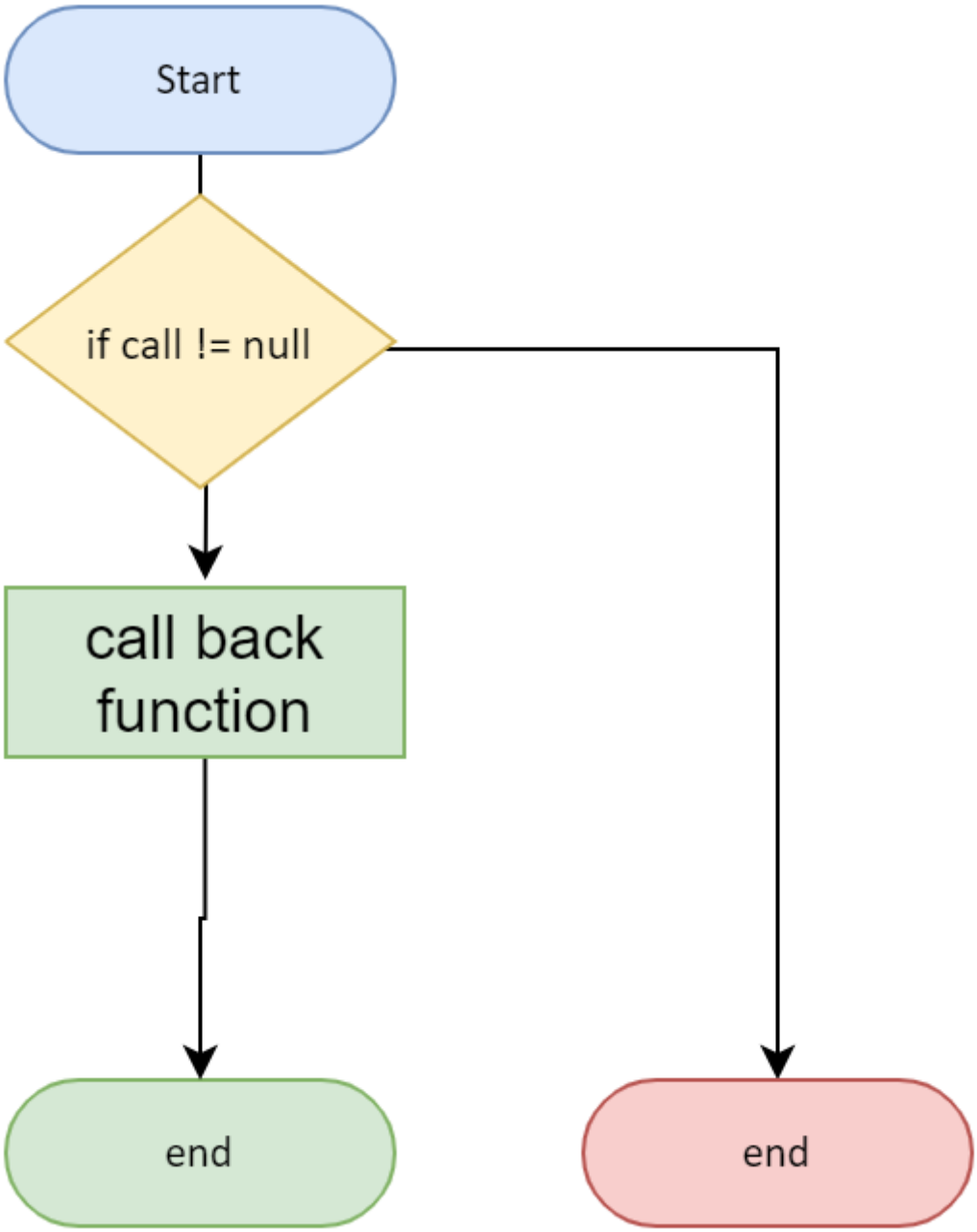
3.1.2.3. Systick_reload



3.1.2.4. systick_enable

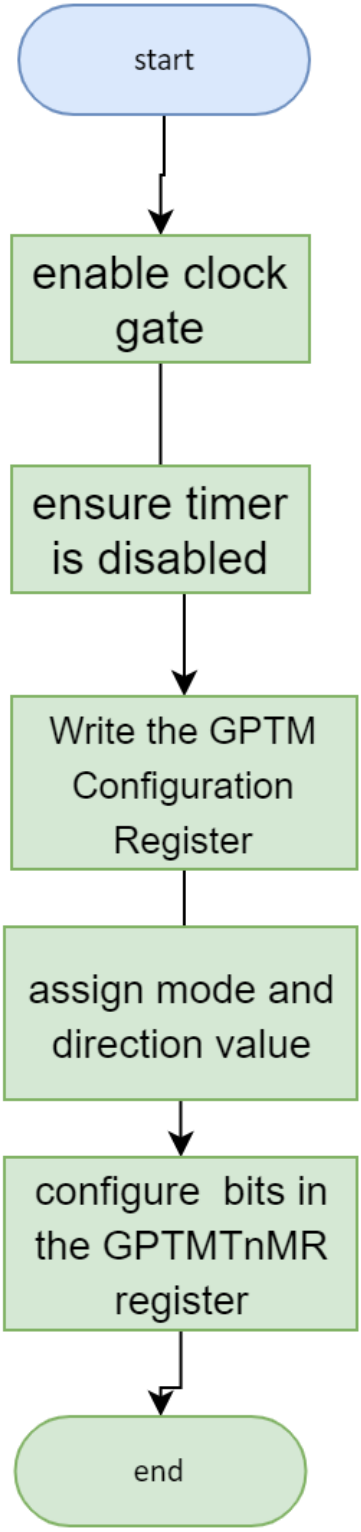


3.1.2.5. SysTick_Handler

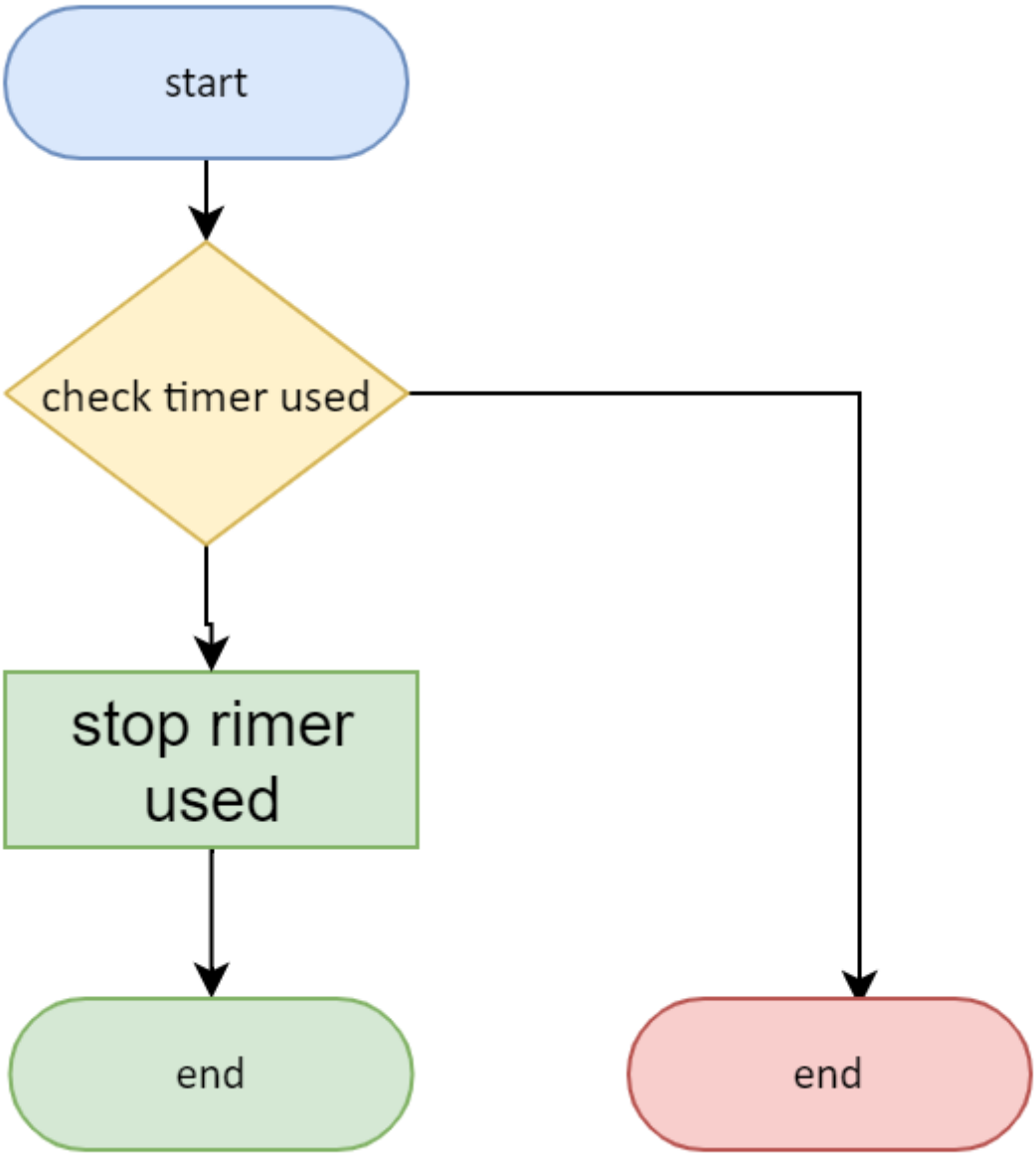


3.1.3. Timer Module

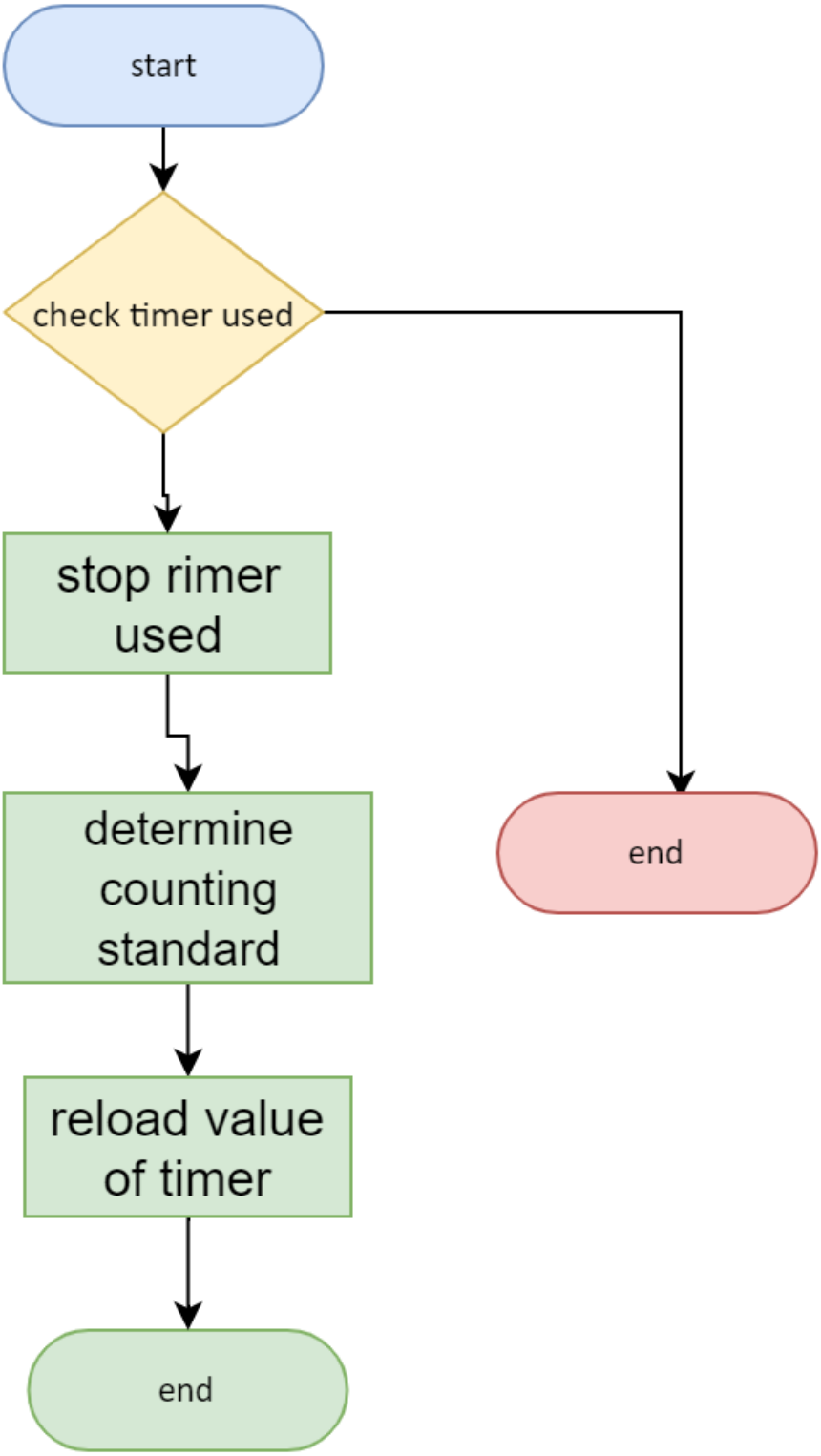
3.1.3.1. Gpt_init



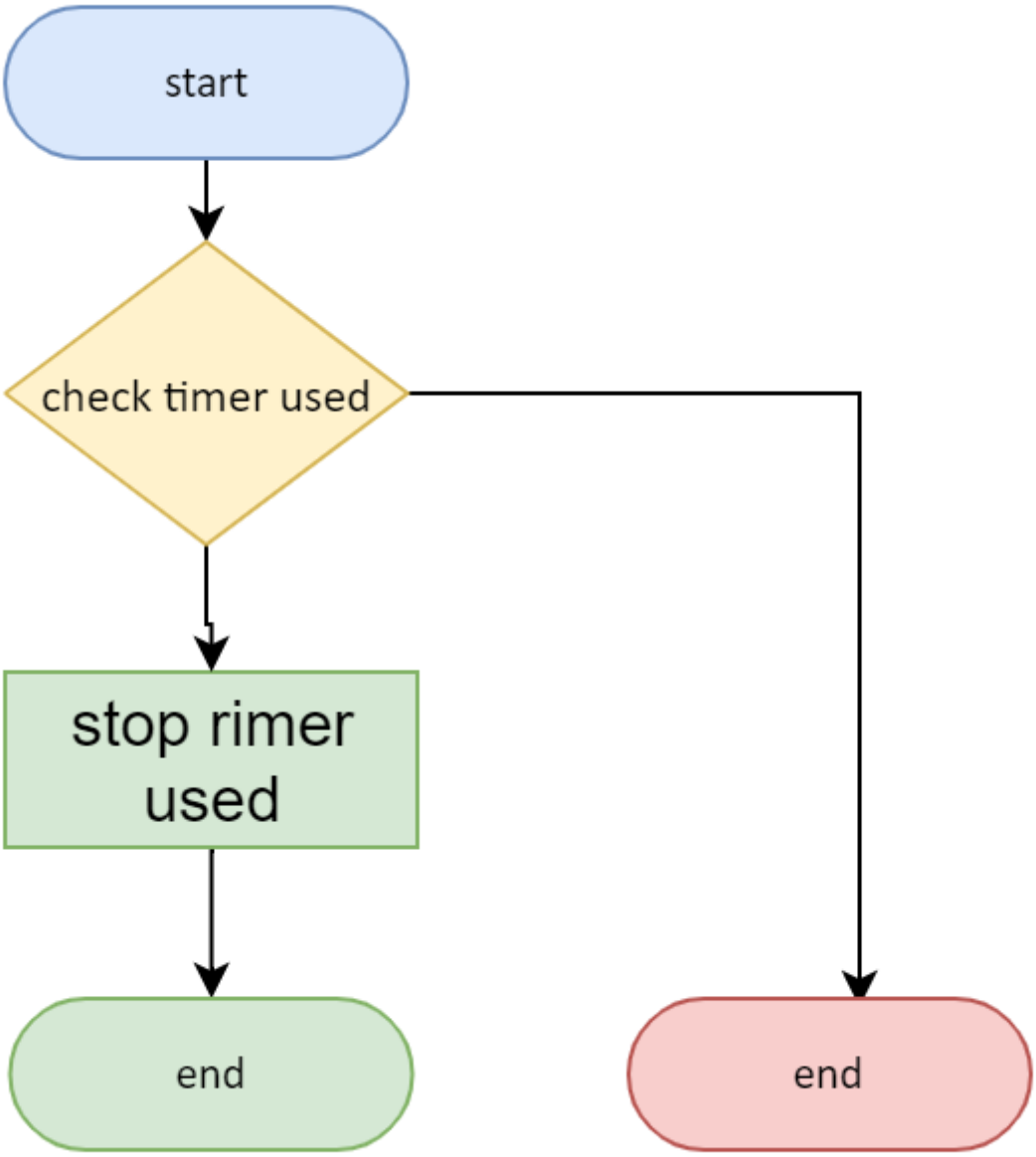
3.1.3.2. Gpt_stop_timer



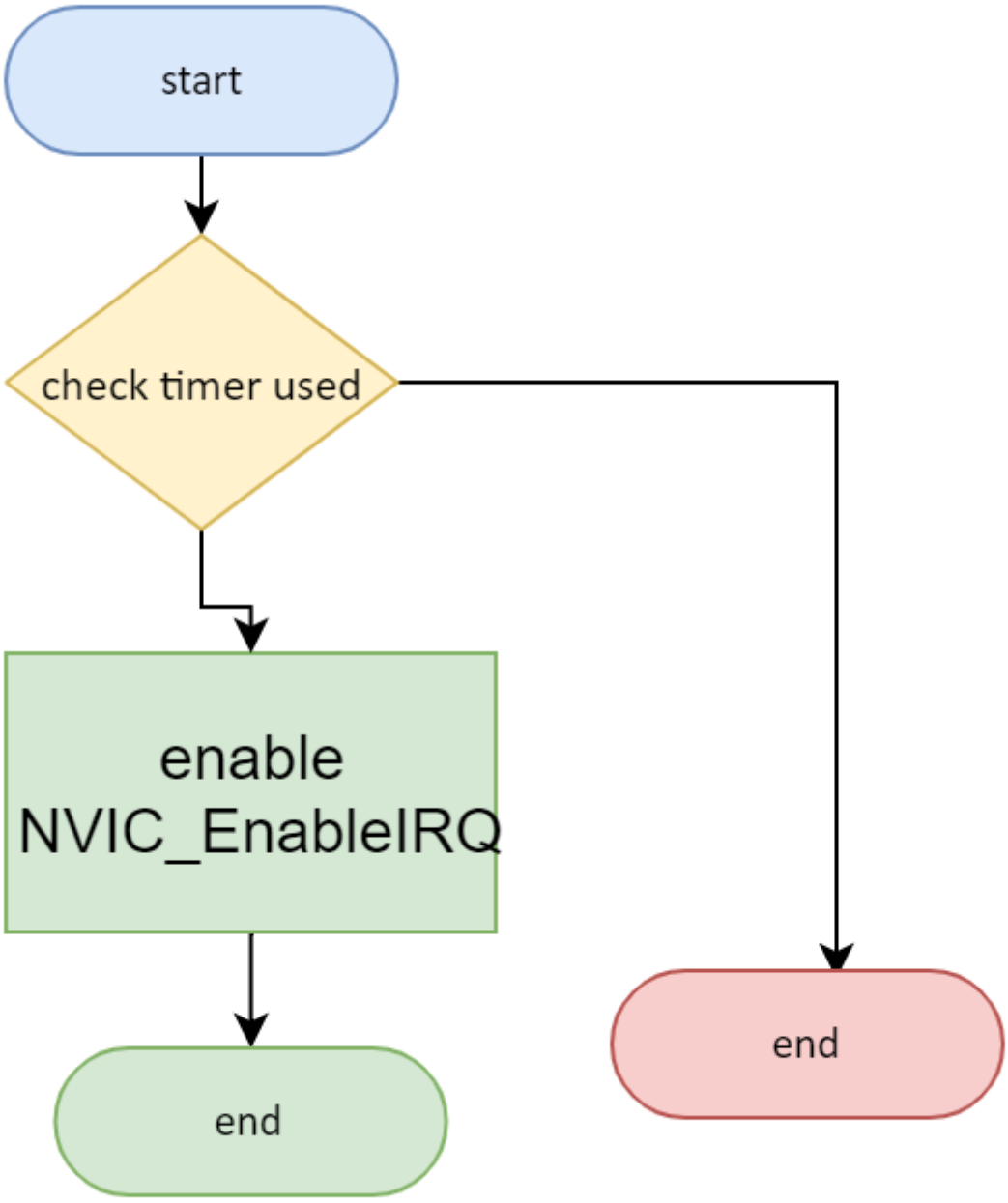
3.1.3.3. Gpt_set_time



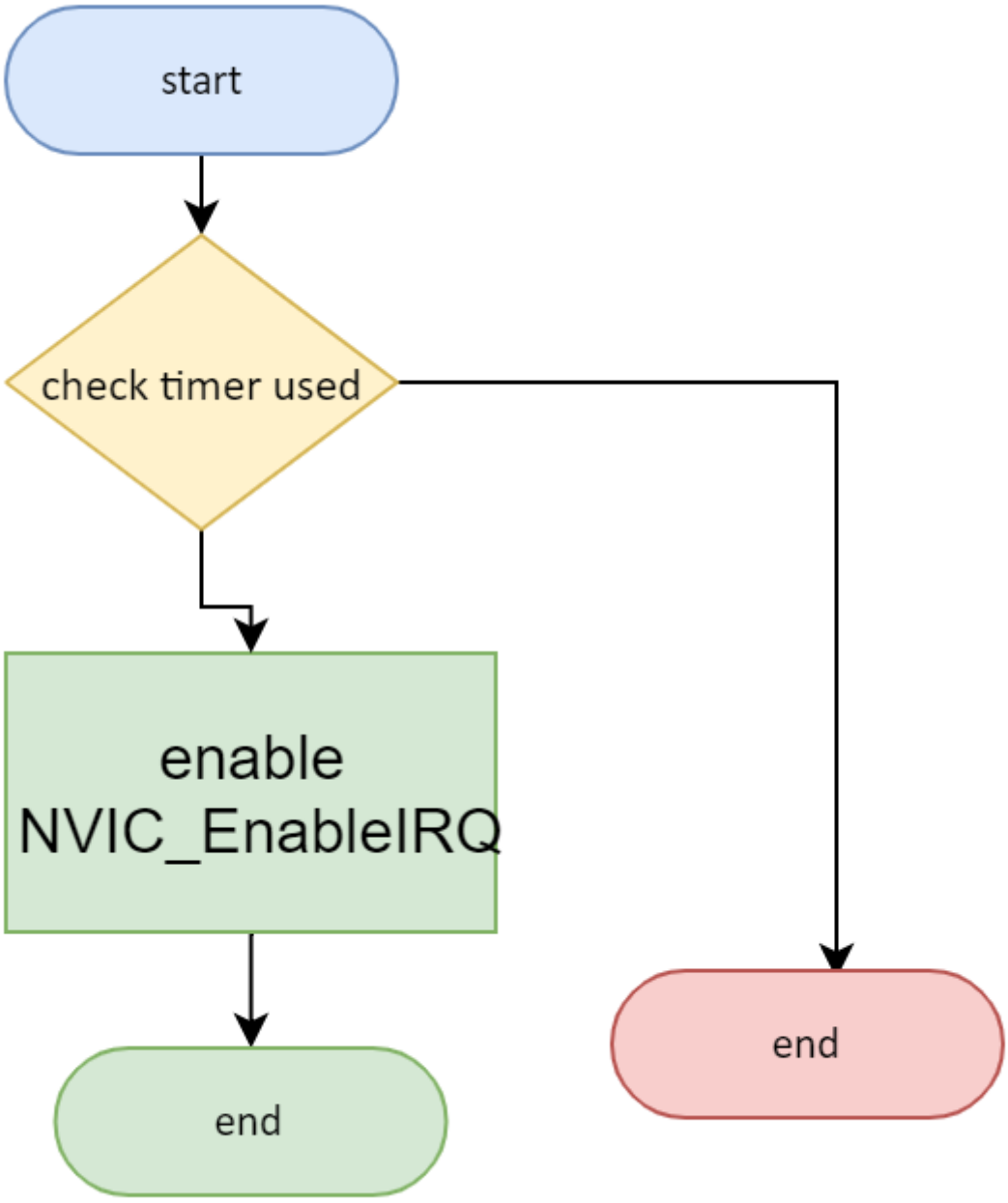
3.1.3.4. Gpt_start_timer



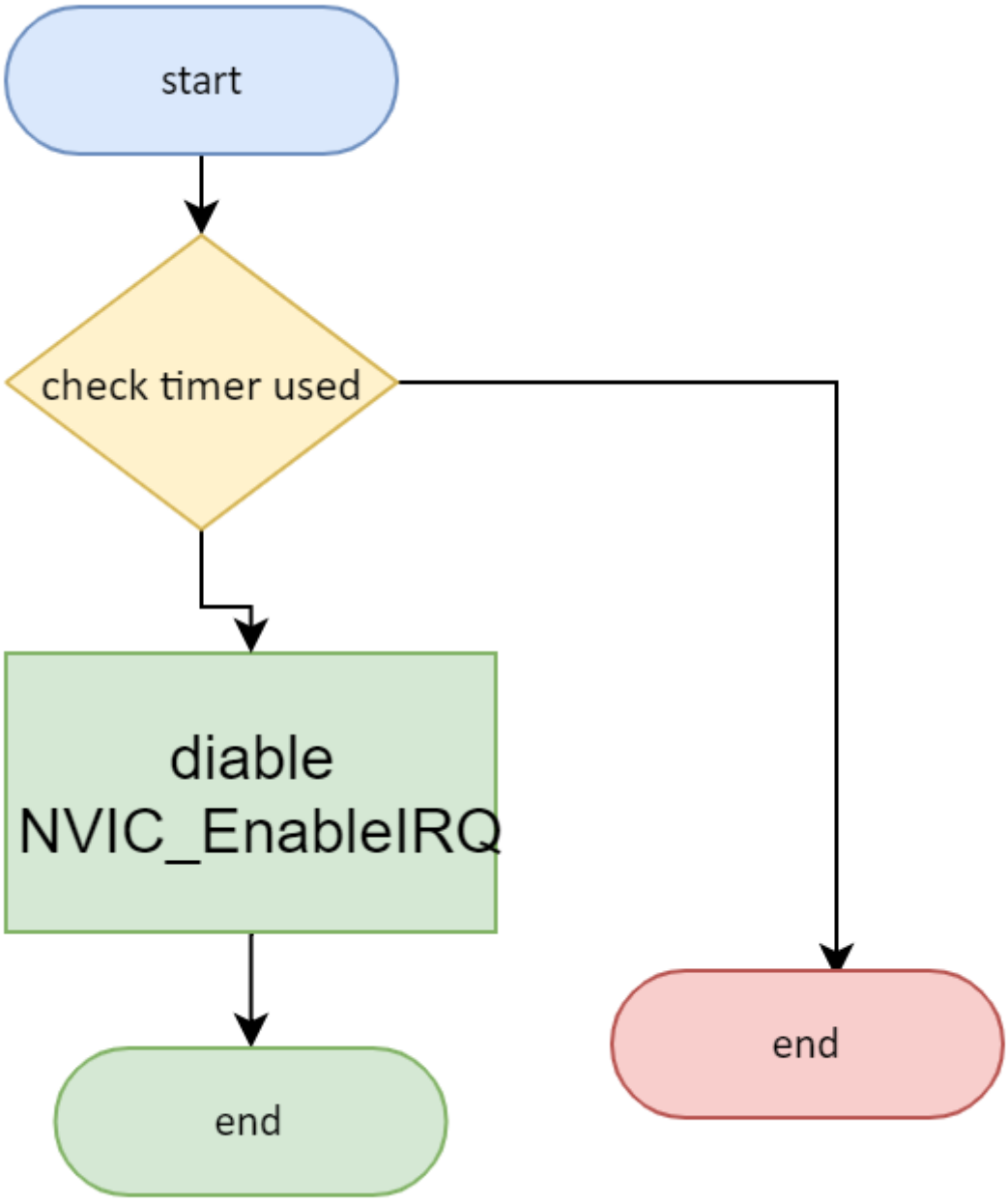
3.1.3.5. Gpt_enable_notificatoin



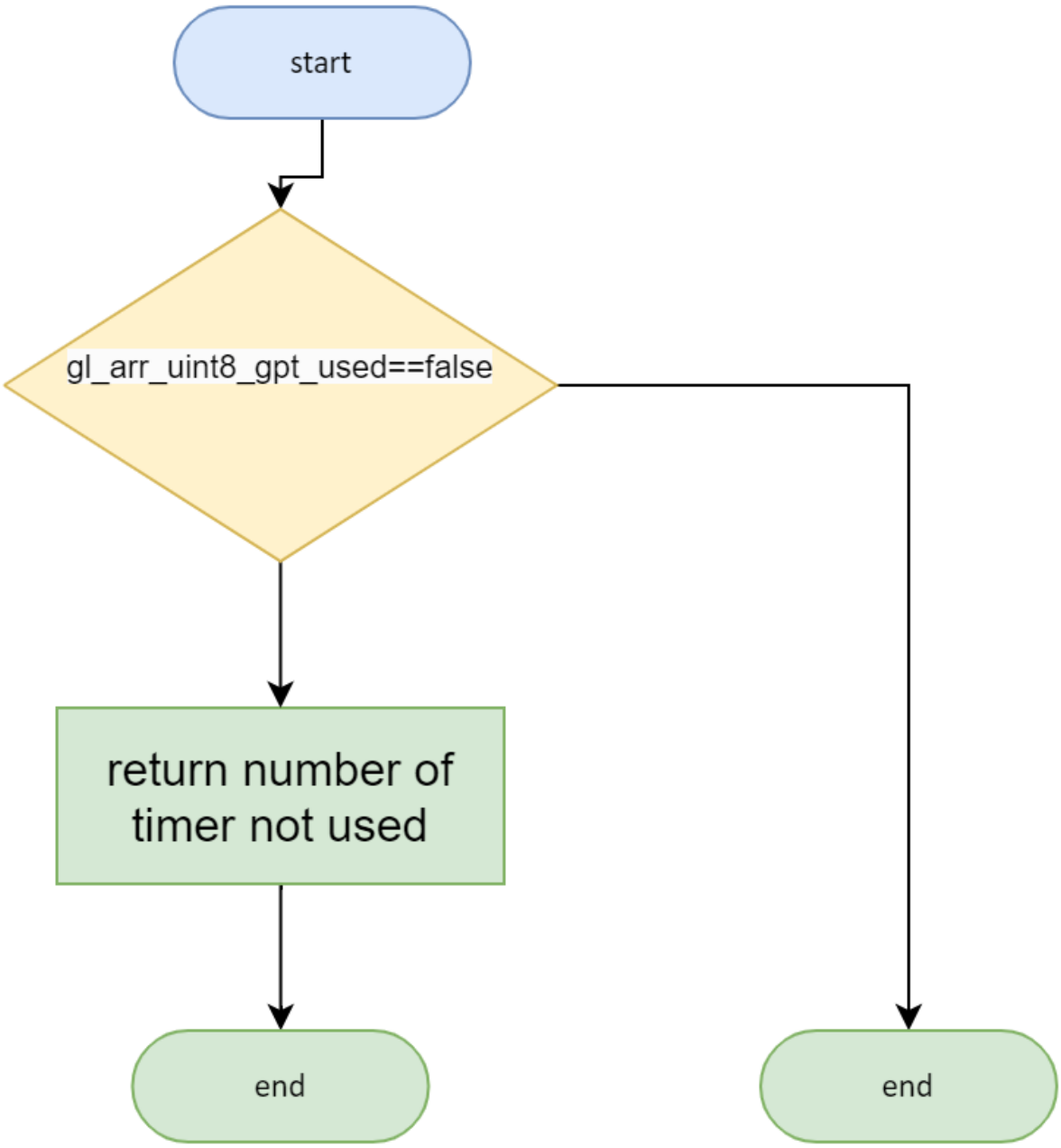
3.1.3.6. Gpt_disable_notificatoin



3.1.3.7. Gpt_set_callback



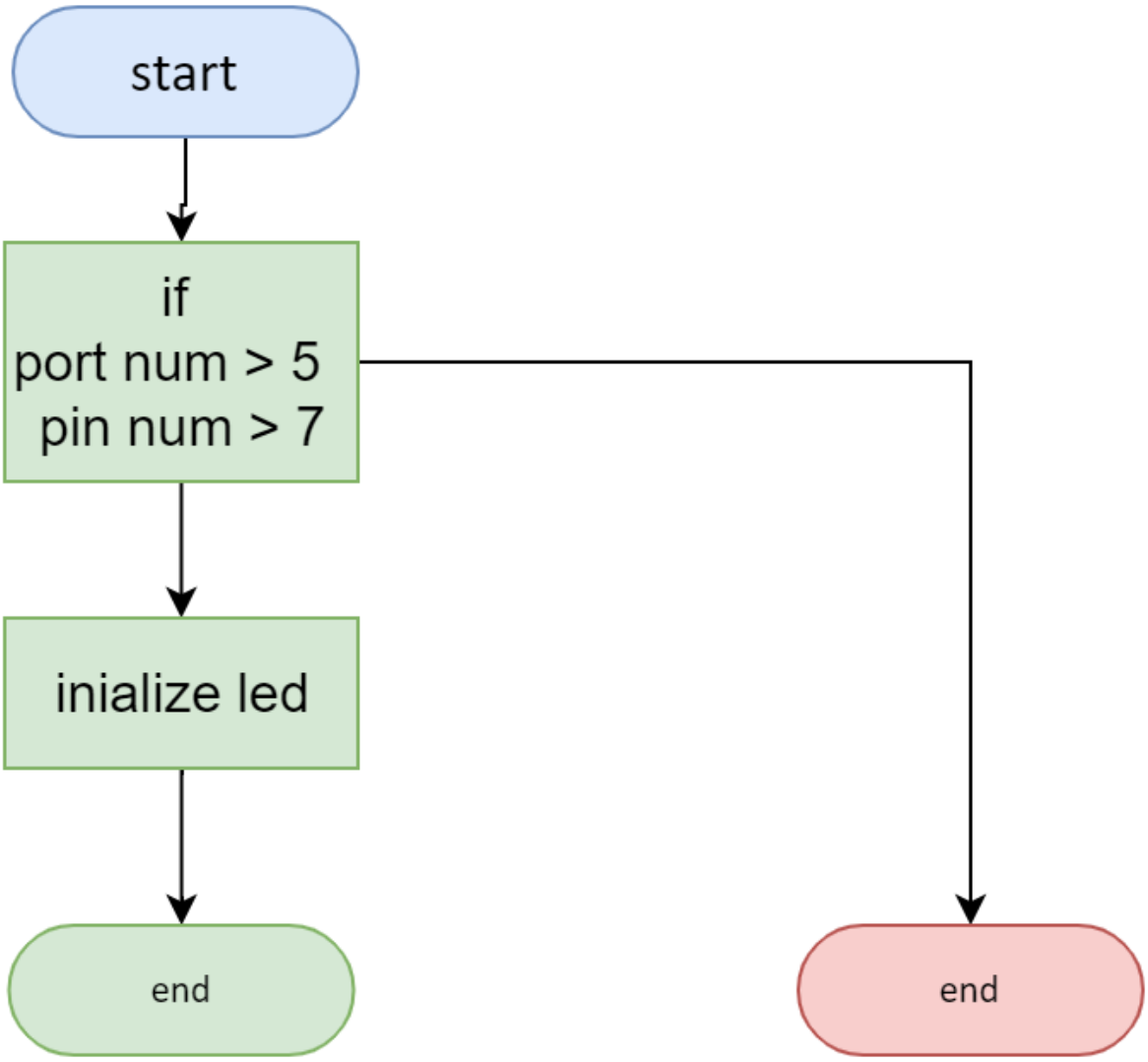
3.1.3.8. Gpt_get_unused_channel



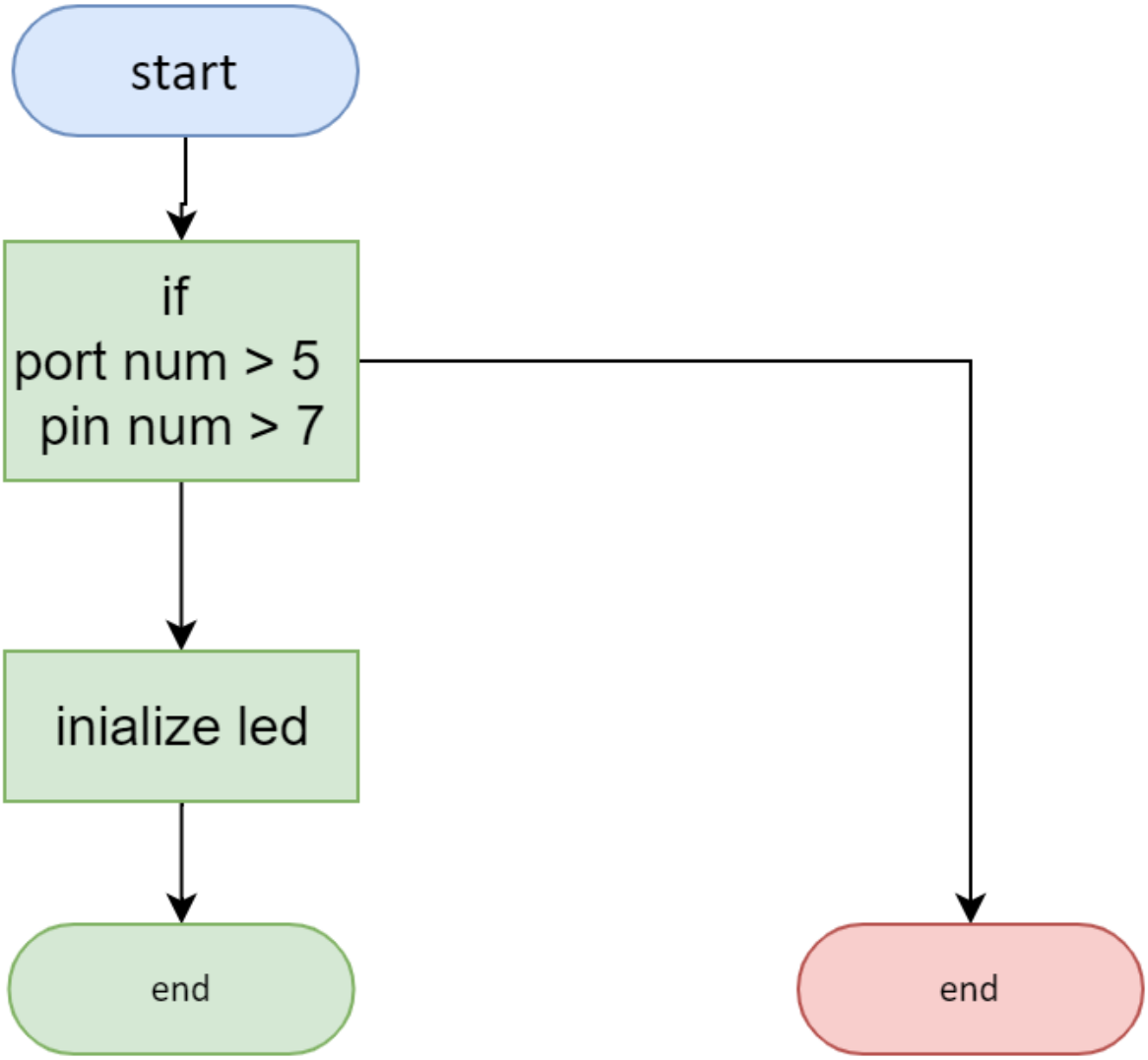
3.2. HAL Layer

3.2.1. LED Module

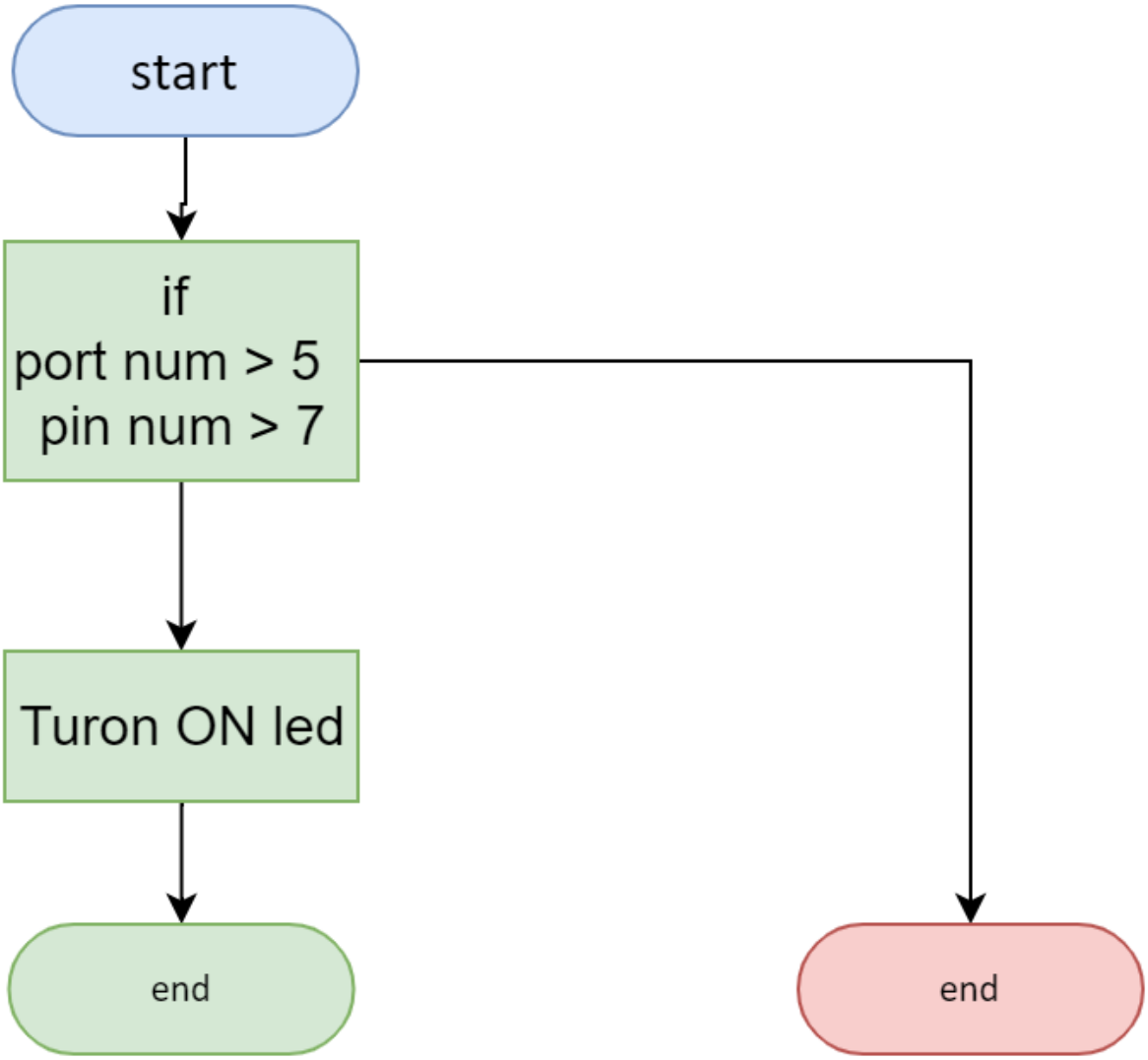
3.2.1.1. Led_init



3.2.1.2. Led_on

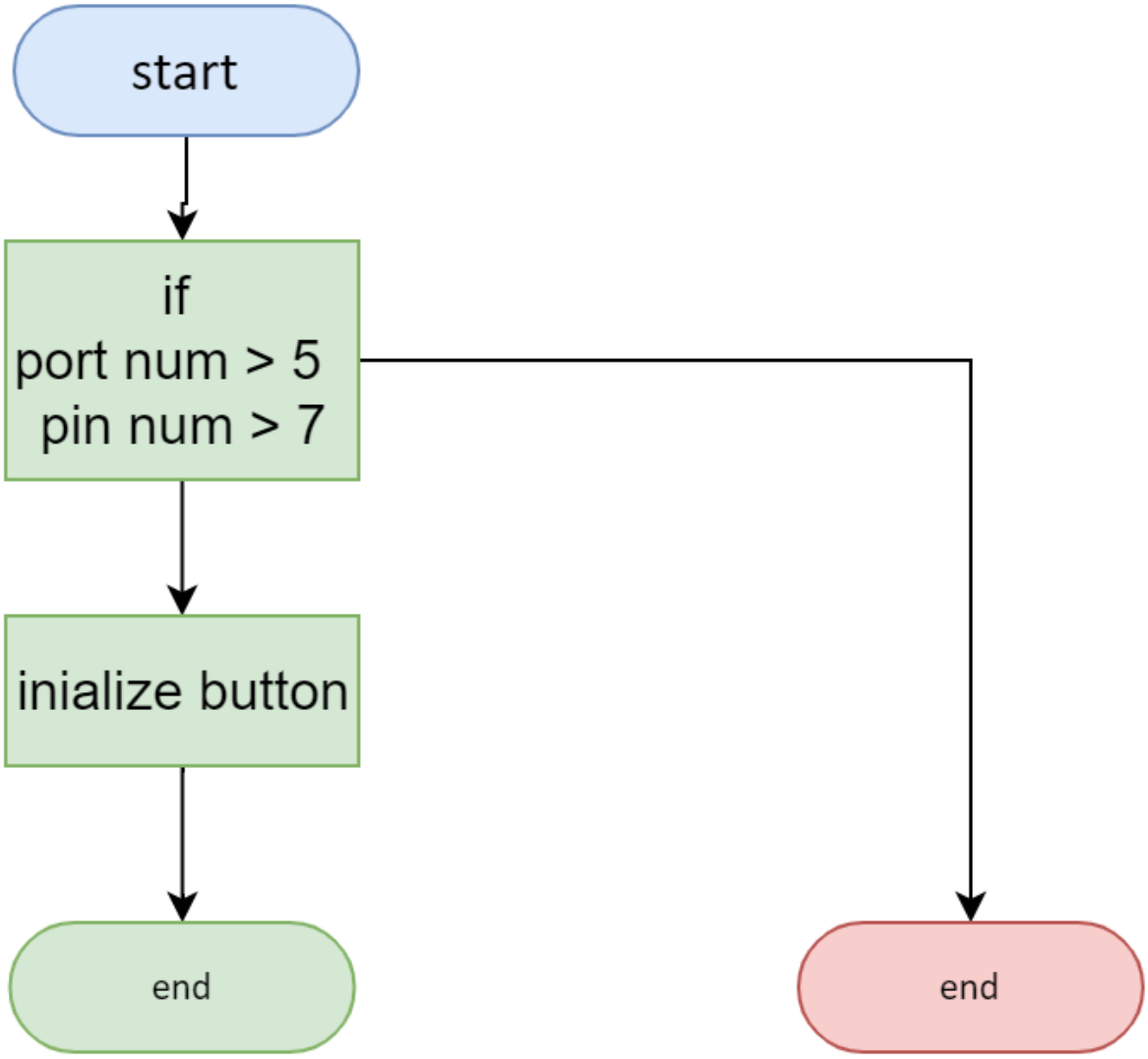


3.2.1.3. Led_off

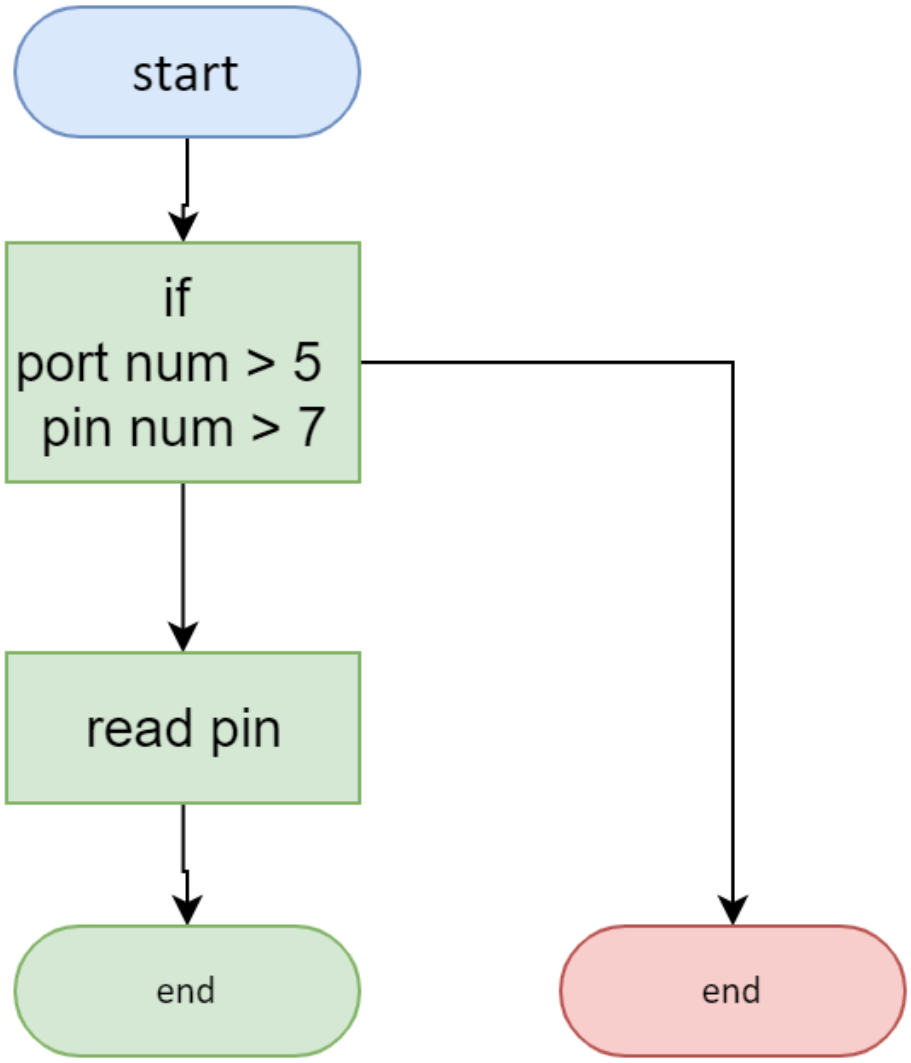


3.2.2. BTN Module

3.2.2.1. Button_init



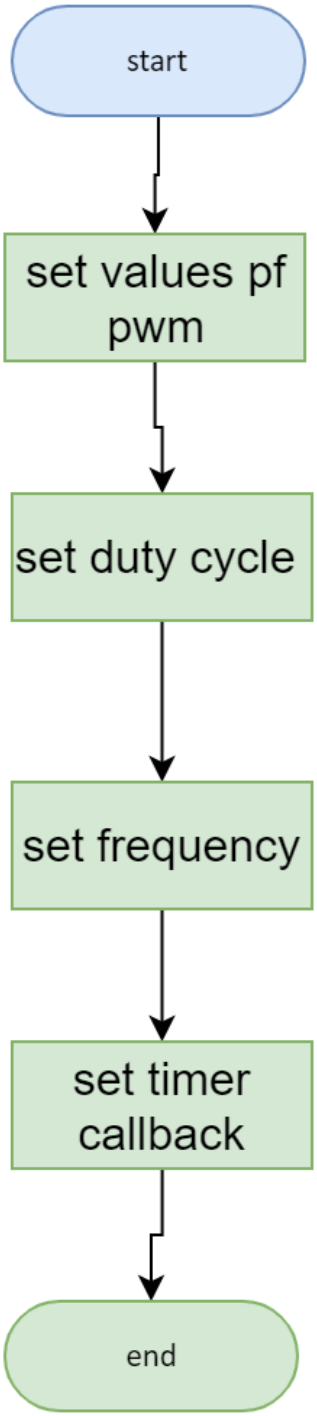
3.2.2.2. Button_get_state



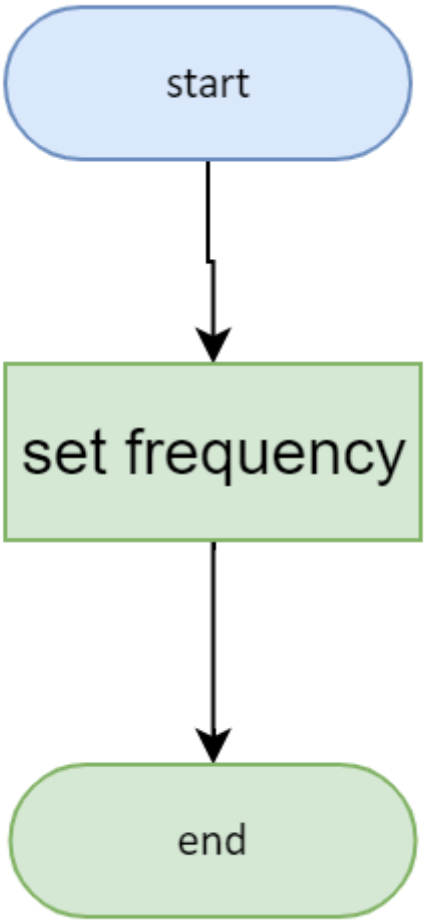
3.3. SERV Layer

3.3.1. PWM Module

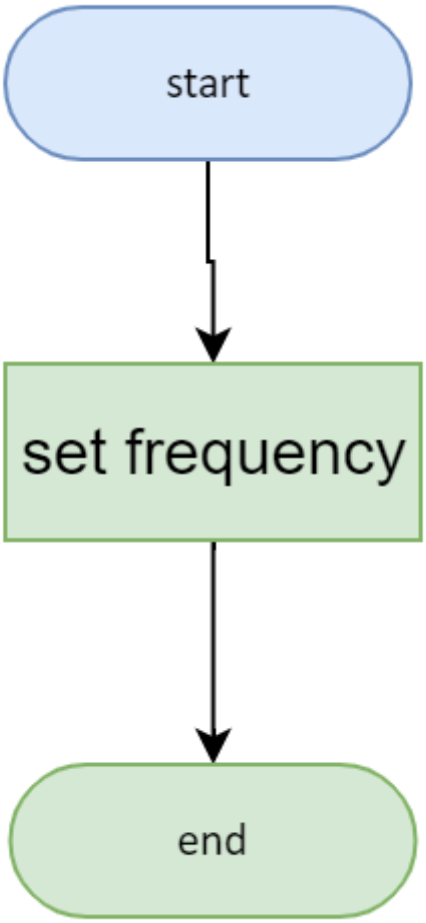
3.3.1.1. Pwm_init



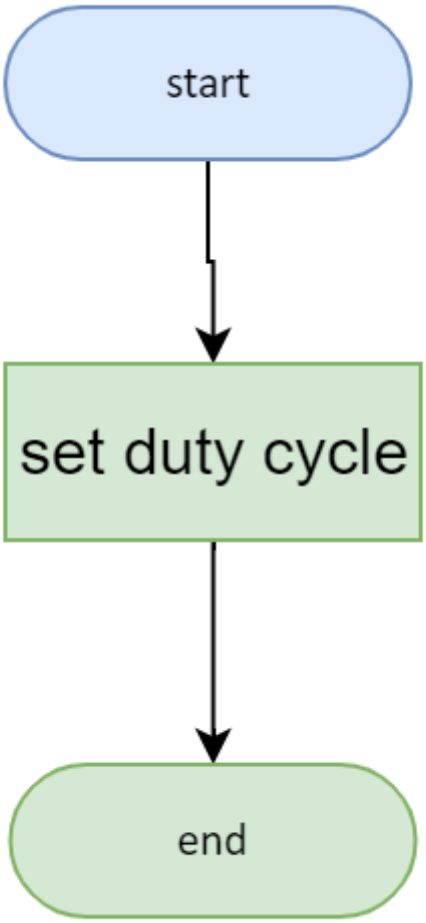
3.3.1.2. Pwm_set_frequency



3.3.1.3. Pwm_set_duty_cycle



3.3.1.4. Pwm_start



3.3.1.5. Pwm_stop