Sprints

RGB LED BRIGHTNESS CONTROL

ARM

Develop the Timer Driver and use it to control the RGB LED brightness on the Tiva-C board based on the push button press.

Prepared By

Team 1 - Sub Team A

- Alaa Hisham
- Hossam Elwahsh

RGB LED Brightness Control

1. Project Introduction

Develop the Timer Driver and use it to control the RGB LED brightness on the Tiva-C board based on the push button press.

1.1. Project Components

- Tiva-C TM4C123G LaunchPad
- One push button **SW1**
- One RGB LED **(user RGB led)**

1.2. System Requirements

Hardware Requirements

- Use the TivaC board
- Use SW1 as an input button
- Use the RGB LED

Software Requirements

The RGB LED is OFF initially
The PWM signal has a 500 ms duration
The system has four states:

1. SW1 - First press
    - The Green LED will be on with a 30% duty cycle

2. SW1 - Second press
    - The Green LED will be on with a 60% duty cycle

3. SW1 -Third press
    - The Green LED will be on with a 90% duty cycle

4. SW1 - Fourth press will be off
    - The Green LED will be off

5. On the fifth press, system state will return to state 1

Implement your drivers

- Implement GPT Driver

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.

2.1.2. Layered Architecture

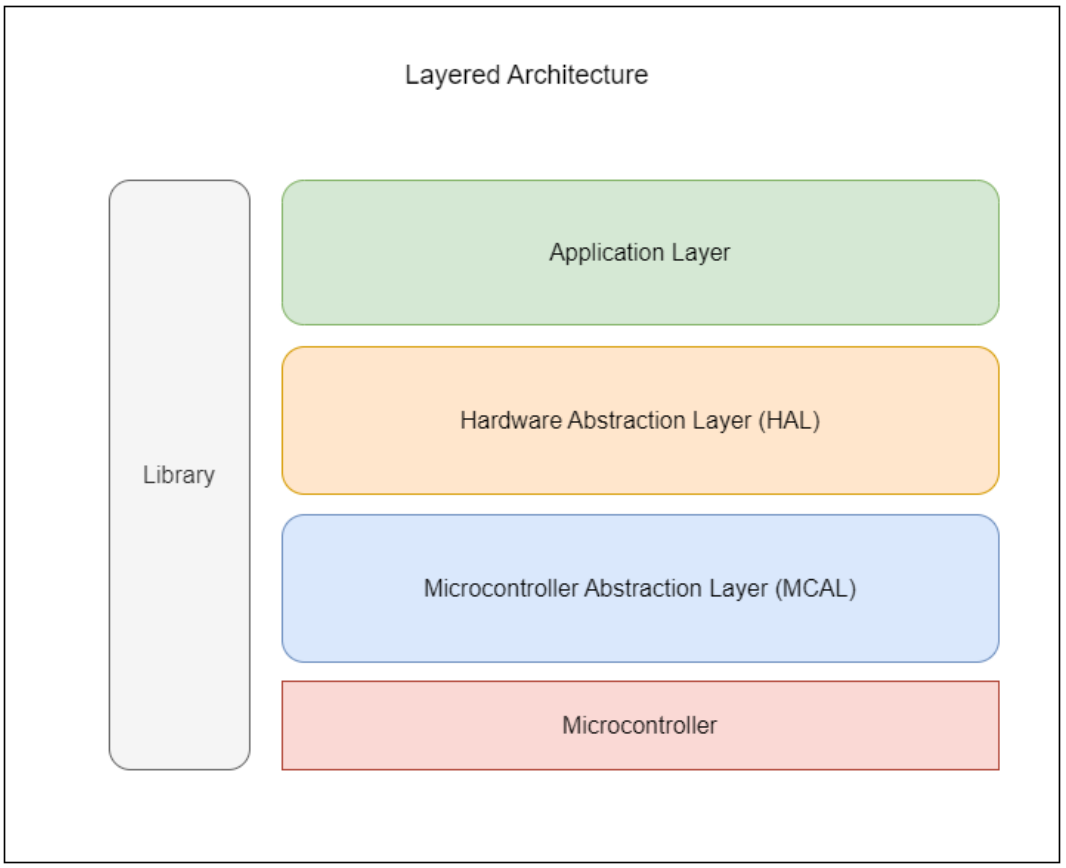
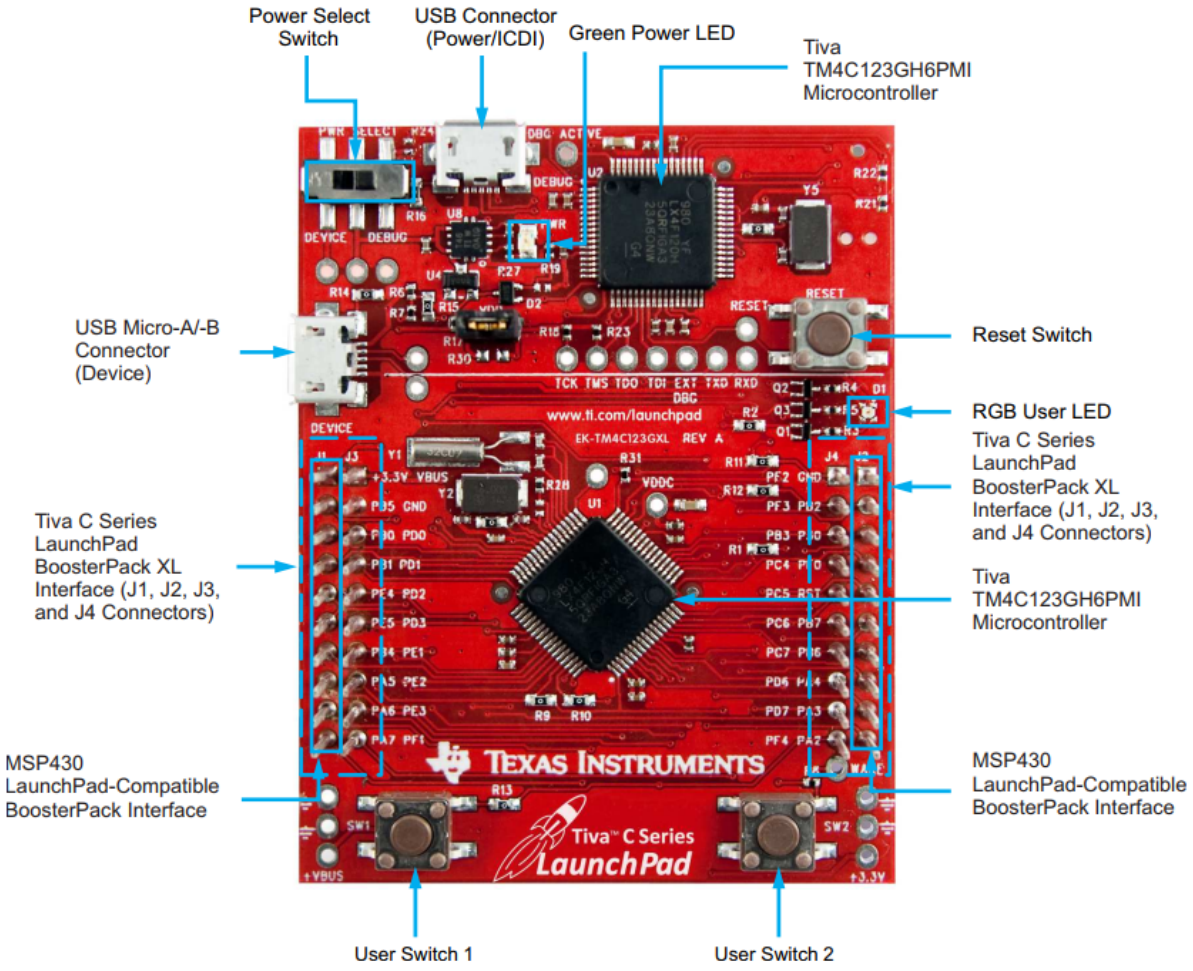


Figure 1. Layered Architecture Design

2.1.3. Tiva C Board Schematic

Figure 1-1. Tiva C Series TM4C123G LaunchPad Evaluation Board



2.2. Modules Description

2.2.1. GPIO (General Purpose Input/Output) Module

The GPIO (General Purpose Input/Output) driver in the Tiva C TM4C123G microcontroller provides a versatile interface for interacting with external devices through digital input and output pins. It allows the microcontroller to read input signals from sensors, buttons, or switches, and control output signals to drive LEDs, motors, or other devices. The GPIO driver plays a crucial role in enabling the TM4C123G microcontroller to communicate with the outside world.

2.2.2. 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.

2.2.3. LED Module

The LED driver enables control of Light-Emitting Diodes (LEDs) for various applications. LEDs are widely used for visual indicators, status displays, and user interface feedback in embedded systems.

2.2.4. GPT Module

The GPT (General-Purpose Timer) driver is a software module that provides a high-level interface for controlling and utilizing the general-purpose timers available on microcontrollers. These timers are versatile peripherals that offer various timing and counting functionalities, making them useful for a wide range of applications.

2.2.5. PWM Module

A Pulse Width Modulation (PWM) module is a hardware component or peripheral found in many microcontrollers and embedded systems. PWM is a technique used to control the average voltage or current supplied to a load by rapidly switching the output signal between ON and OFF states. It is widely used for applications such as motor control, LED dimming, audio synthesis, and power regulation.
At its core, PWM works by rapidly toggling the output between two states: high and low. The ratio of time spent in the high state (on-time) to the total time of the cycle (period) determines the average voltage or current level. By adjusting the duty cycle, which is the ratio of on-time to the period, the effective voltage or current applied to the load can be controlled.

2.2.6. Design

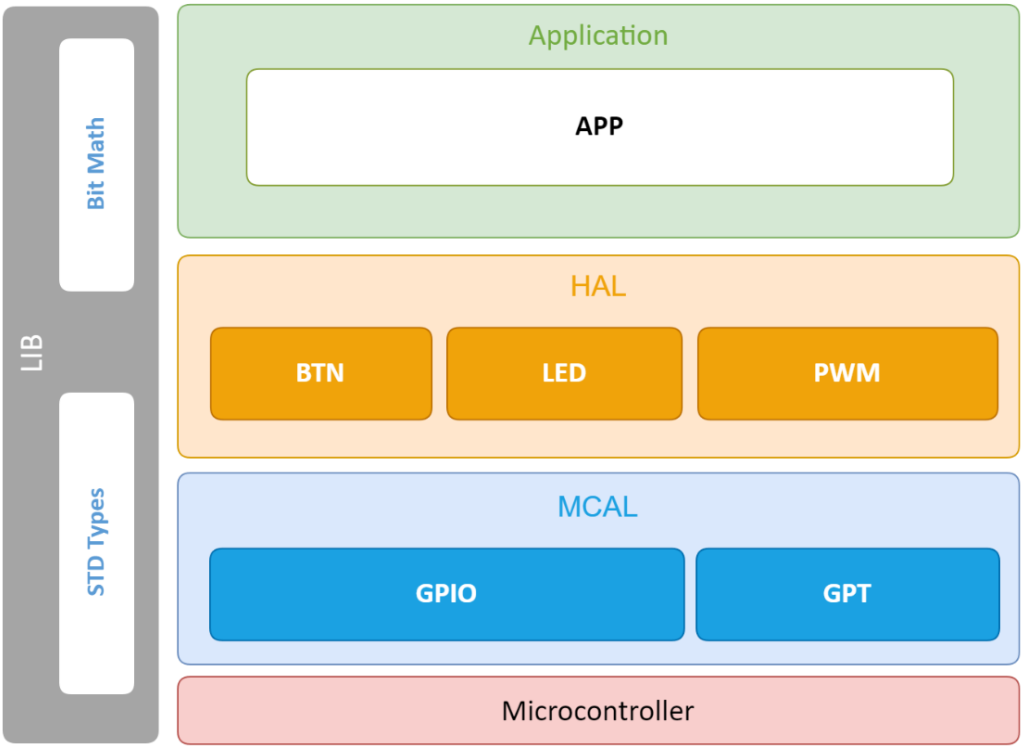


Figure 3. System Modules Design

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

```
/*-----------------------------------------------------------*/
/- MACROS
/-----------------------------------------------------------*/
#define PORT_CLR    0x00
#define PORT_SET    0xff

/*-----------------------------------------------------------*/
/- PRIMITIVE TYPES
/-----------------------------------------------------------*/
typedef void (*gpio_cb)(void);

/*-----------------------------------------------------------*/
/- ENUMS
/-----------------------------------------------------------*/
typedef enum
{
	GPIO_PORT_A =0	,
	GPIO_PORT_B		,
	GPIO_PORT_C		,
	GPIO_PORT_D		,
	GPIO_PORT_E		,
	GPIO_PORT_F		,
	GPIO_PORT_TOTAL
}en_gpio_port_t;

typedef enum
{
	GPIO_PIN_0 = 0	,
	GPIO_PIN_1		,
	GPIO_PIN_2		,
	GPIO_PIN_3		,
	GPIO_PIN_4		,
```

```
	GPIO_PIN_5		,
	GPIO_PIN_6		,
	GPIO_PIN_7		,
	GPIO_PIN_TOTAL
}en_gpio_pin_t;

typedef enum
{
	DIGITAL	= 0	,
	ANALOG		,
	ALT_FUNC
}en_gpio_pin_mode_t;


typedef enum
{
	LOW = 0,
	HIGH
}en_gpio_pin_level_t;


typedef enum
{
	INPUT		= 0 ,
	OUTPUT		,
	INPUT_ANALOG	,
	INPUT_PULL_UP	,
	INPUT_PULL_DOWN	,
	OUTPUT_OPEN_DRAIN	,
	ALT_FUNCTION	,
	PIN_CFG_TOTAL
}en_gpio_pin_cfg_t;

typedef enum
{
	PIN_CURRENT_2MA = 0	,
	PIN_CURRENT_4MA		,
	PIN_CURRENT_8MA
}en_gpio_pin_current_t;

typedef enum
{
	FALLING_EDGE = 0	,
	LOW_LEVEL		,
	RISING_EDGE		,
	HIGH_LEVEL		,
	BOTH_EDGES		,
	INT_EVENT_TOTAL
}en_gpio_int_event_t;
```

```
typedef enum
{
	GPIO_OK                = 0 ,
	GPIO_INVALID_PORT          ,
	GPIO_INVALID_PIN           ,
	GPIO_INVALID_PIN_CFG       ,
	GPIO_INVALID_INT_EVENT     ,
	GPIO_ERROR
}en_gpio_error_t;

/*-----------------------------------------------------------/
/- STRUCTURES
/-----------------------------------------------------------*/
typedef struct
{
	en_gpio_port_t        port     ; /* The port of the pin to configure */
	en_gpio_pin_t         pin      ; /* The pin number to configure */
	en_gpio_pin_cfg_t     pin_cfg ;
	en_gpio_pin_current_t current ; /* o/p current on the pin(ignored if input) */
}st_gpio_cfg_t;
```

```
| @breif Function initialize a gpio pin
|
| This function configures any gpio pin with the
| configurations set in the referenced structure
|
| @Parameters
|     [in] ptr_str_pin_cfg : pointer to the pin configuration structure
|
| Return
|        GPIO_OK          : If the operation is done successfully
|        GPIO_INALID_PORT : If the passed port is not a valid port
|        GPIO_INALID_PIN  : If the passed pin is not a valid pin
|        GPIO_ERROR   : If the passed pointer is a null pointer
|
en_gpio_error_t gpio_pin_init (st_gpio_cfg_t* pin_cfg);

| @breif Function to set the value of an entire port
|
| @Parameters
|            [in]  en_a_port      : The desired port
|            [in]  u8_a_portVal   : The value to set the port to
| Return
|       GPIO_OK           : If the operation is done successfully
|      GPIO_INVALID_PORT : If the passed port is not a valid port
|      GPIO_ERROR        : If the pin value is invalid (not HIGH/LOW)
|                         or if the port is not configured as an output port
|
```

```
en_gpio_error_t gpio_setPortVal(en_gpio_port_t en_a_port, uint8_t_
u8_a_portVal);

| @breif Function to set the value of a given pin
|
| This function sets the value of the given pin to
| the given pin value
|
| @Parameters
|      [in]  en_a_port      : The port of the desired pin
|      [in]  en_a_pin       : The desired pin to set the value of
|      [in]   en_a_pinVal   : The value to set the bit to
|
| Return
|      GPIO_OK              : If the operation is done successfully
|      GPIO_INVALID_PORT    : If the passed port is not a valid port
|      GPIO_INVALID_PIN     : If the passed pin is not a valid pin
|      GPIO_ERROR           : If the pin value is invalid (not HIGH/LOW)
|                             or if the pin is not configured as an output pin
|
en_gpio_error_t gpio_setPinVal (en_gpio_port_t en_a_port, en_gpio_pin_t
en_a_pin, en_gpio_pin_level_t en_a_pinVal);

|
| @breif Function to toggle the value of a given pin
|
| @Parameters
|            [in]  en_a_port    : The port of the desired pin
|            [in]  en_a_pin     : The desired pin to set the value of
|
| Return
|            GPIO_OK            : If the operation is done successfully
|            GPIO_INVALID_PORT  : If the passed port is not a valid port
|            GPIO_INVALID_PIN   : If the passed pin is not a valid pin
|            GPIO_ERROR         : If the pin is not configured as an output pin
|
en_gpio_error_t gpio_togPinVal (en_gpio_port_t en_a_port, en_gpio_pin_t
en_a_pin);
```

```
| @breif Function to get the value of a given pin
|
| This function reads the value of the given pin and
| returns the value in the given address
|
| @Parameters
|            [in]  en_a_port   : The port of the desired pin
|            [in]  en_a_pin    : The desired pin to read value of
|            [out] pu8_a_val   : pointer to variable to store the pin value
|
|
| Return
|            GPIO_OK           : If the operation is done successfully
|            GPIO_INVALID_PORT : If the passed port is not a valid port
|            GPIO_INVALID_PIN  : If the passed pin is not a valid pin
|            GPIO_ERROR        : If the passed pointer is a null pointer
|
en_gpio_error_t gpio_getPinVal (en_gpio_port_t en_a_port, en_gpio_pin_t
en_a_pin, en_gpio_pin_level_t* pu8_a_Val);
```

2.3.2.2. GPT Driver

```
typedef enum{
    GPT_OK          = 0 ,
    GPT_INVALID_ARGS    ,
    GPT_INVALID_CFG     ,
    GPT_NOT_SUPPORTED   ,
    GPT_ERROR           ,
    GPT_STATUS_TOTAL
}en_gpt_status_t;

typedef enum{
    TIME_IN_SEC      = 1        ,
    TIME_IN_MS       = 1000     ,
    TIME_IN_US       = 1000000  ,
    TIME_UNITS_TOTAL
}en_gpt_time_unit_t;

typedef enum{
    CH_0    = 0 ,
    CH_1        ,
    CH_2        ,
    CH_3        ,
    CH_4        ,
    CH_5        ,
    CH_6_W      ,
    CH_7_W      ,
    CH_8_W      ,
    CH_9_W      ,
    CH_10_W     ,
    CH_11_W     ,
    CH_TOTAL
}en_gpt_channel_t;

| Initializes GPT timers
|
| Return
|            GPT_OK              :   If Success
|            GPT_ERROR           :   If Failed
|            GPT_INVALID_CFG     :   Bad Config
|
en_gpt_status_t gpt_init(void);


| Calculates timer values then starts a delay on the requested timer channel
|
| Parameters
|          [in]   en_a_gpt_channel      :     Timer Channel
|          [in]   uint32_a_delay_value  :     Delay value in Time Units
```

```
|          [in]   en_a_gpt_time_unit    :     Time units (seconds,
|               milli-seconds, micro-seconds)
|
| Return
|            GPT_OK                :     If Success
|            GPT_INVALID_ARGS      :     Failed, Invalid Args Given
|            GPT_NOT_SUPPORTED     :     Failed, Delay value requested
|                                 isn't supported by selected timer channel
|
en_gpt_status_t gpt_start(en_gpt_channel_t en_a_gpt_channel, uint32_t_
uint32_a_delay_value, en_gpt_time_unit_t en_a_gpt_time_unit);

| Stops timer channel
|
| Parameters
|          [in]   en_a_gpt_channel    :   Timer Channel
|
|
| Return
|            GPT_OK              :     If Success
|            GPT_ERROR           :     Failed, Error occurred
|            GPT_INVALID_ARGS    :     Failed, Invalid Args Given
|
en_gpt_status_t gpt_stop(en_gpt_channel_t en_a_gpt_channel);

| Get Timer Current Elapsed time
|
| Parameters
|          [in]   en_a_gpt_channel            :   Timer Channel
|          [out]  ptr_uint32_a_elapsed_time   :   Ptr to store elapsed
|               time value
|
| Return
|            GPT_OK                      :     If Success
|            GPT_ERROR                   :     Failed
|            GPT_INVALID_ARGS            :     Failed, Invalid Args Given
|
en_gpt_status_t gpt_get_elapsed_time(en_gpt_channel_t en_a_gpt_channel,
uint32_t_ * ptr_uint32_a_elapsed_time);


| Get Timer Current Remaining time
|
| Parameters
|          [in]   en_a_gpt_channel            :   Timer Channel
|          [out]  ptr_uint32_a_rem_time       :   Ptr to store remaining
|               time value
|
|
```

```
| Return
|           GPT_OK                    :   If Success
|           GPT_ERROR                 :   Failed
|           GPT_INVALID_ARGS          :   Failed, Invalid Args Given
|
en_gpt_status_t gpt_get_remaining_time(en_gpt_channel_t en_a_gpt_channel,
uint32_t_ * ptr_uint32_a_rem_time);


| Sets callback function to a timer channel
|
| Parameters
|          [in]   en_a_gpt_channel                  :   Timer channel
|          [in]   ptr_vd_fun_vd_a_gpt_notification  :   Ptr to callback
|               function to store
|
| Return
|           GPT_OK                    :   If Success
|           GPT_ERROR                 :   Failed
|           GPT_INVALID_ARGS          :   Failed, Invalid Args Given
|
en_gpt_status_t gpt_set_callback(en_gpt_channel_t en_a_gpt_channel,
ptr_vd_fun_vd_t ptr_vd_fun_vd_a_gpt_notification);


|
| Enables timer channel notification (callback)
|
| Parameters
|               en_a_gpt_channel    :   Timer Channel
|
| Return
|           GPT_OK              :   If Success
|           GPT_ERROR           :   Failed
|           GPT_INVALID_ARGS    :   Failed, Invalid Args Given
|
en_gpt_status_t gpt_enable_notification(en_gpt_channel_t en_a_gpt_channel);

|  Disable timer channel notification (callback)
|
| Parameters
|               en_a_gpt_channel    :   Timer Channel
|
|
| Return
|           GPT_OK              :   If Success
|           GPT_ERROR           :   Failed
|           GPT_INVALID_ARGS    :   Failed, Invalid Args Given
|
```

```
en_gpt_status_t gpt_disable_notification(en_gpt_channel_t en_a_gpt_channel);
```

2.3.3. HAL APIs

2.3.3.1. LED APIs

```
| Initializes LED on given port & pin
|
| Parameters
|          [in]   en_a_led_port    :   LED Port
|          [in]   en_a_led_pin     :   LED Pin number in en_led_port
|
| Return
|        LED_OK              :   In case of Successful Operation
|        LED_ERROR           :   In case of Failed Operation
|
en_led_error_t_ led_init(en_led_port_t_ en_a_led_port, en_led_pin_t_ en_a_led_pin);
```

```
|  Turns on LED at given port/pin
|
| Parameters
|          [in]   en_a_led_port     :   LED Port
|          [in]   en_a_led_pin      :   LED Pin number in en_led_port
|
| Return
|        LED_OK              :   In case of Successful Operation
|          LED_ERROR           :   In case of Failed Operation
|
en_led_error_t_ led_on(en_led_port_t_ en_a_led_port, en_led_pin_t_ en_a_led_pin);
```

```
| Turns off LED at given port/pin
|
| Parameters
|          [in]   en_a_led_port     :   LED Port
|          [in]   en_a_led_pin      :   LED Pin number in en_led_port
|
| Return
|        LED_OK              :   In case of Successful Operation
|        LED_ERROR           :   In case of Failed Operation
|
en_led_error_t_ led_off(en_led_port_t_ en_a_led_port, en_led_pin_t_ en_a_led_pin);
```

```
| Toggles LED at given port/pin
|
| Parameters
|          [in]   en_a_led_port   :   LED Port
|          [in]   en_a_led_pin    :   LED Pin number in en_led_port
|
| Return
|        LED_OK              :   In case of Successful Operation
|        LED_ERROR           :   In case of Failed Operation
|
en_led_error_t_ led_toggle(en_led_port_t_ en_a_led_port, en_led_pin_t_ en_a_led_pin);
```

2.3.3.2. BTN APIs

```
/*-----------------------------------------------------------/
/- ENUMS
/-----------------------------------------------------------*/
/* button Pins */
typedef enum{
	BTN_PIN_0	=	0	,
	BTN_PIN_1			,
	BTN_PIN_2			,
	BTN_PIN_3			,
	BTN_PIN_4			,
	BTN_PIN_5			,
	BTN_PIN_6			,
	BTN_PIN_7			,
	BTN_PIN_TOTAL
}en_btn_pin_t_;

/* button Ports */
typedef enum
{
	BTN_PORT_A	=	0	,
	BTN_PORT_B			,
	BTN_PORT_C			,
	BTN_PORT_D			,
	BTN_PORT_E			,
	BTN_PORT_F			,
	BTN_PORT_TOTAL
}en_btn_port_t_;

typedef enum
{
	BTN_STATE_NOT_PRESSED = 0 ,
	BTN_STATE_PRESSED
}en_btn_state_t_;

typedef enum
{
	BTN_INTERNAL_PULL_UP = 0  ,
	BTN_INTERNAL_PULL_DOWN    ,
	BTN_EXTERNAL_PULL_UP      ,
	BTN_EXTERNAL_PULL_DOWN    ,
	BTN_PULL_TOTAL
}en_btn_pull_t_;

typedef enum
{
	BTN_ACTIVATED= 0 ,
	BTN_DEACTIVATED
}en_btn_active_state_t_;
```

```
typedef enum
{
	BTN_STATUS_OK = 0                 ,
	BTN_STATUS_INVALID_PULL_TYPE ,
	BTN_STATUS_INVALID_STATE    ,
	BTN_STATUS_DEACTIVATED
}en_btn_status_code_t_;

/*-------------------------------------------------------------/
/- STRUCTS
/-------------------------------------------------------------*/
typedef struct
{
	en_btn_port_t_                          en_btn_port    ;
	en_btn_pin_t_                           en_btn_pin     ;
	en_btn_pull_t_                          en_btn_pull_type ;
	/** Read only */
	en_btn_active_state_t_           en_btn_activation;
}st_btn_config_t_;
```

```
|
| Function to initialize a given button instance
|
| Parameters
|       ptr_str_btn_config       : pointer to the desired button structure
|
| Return
|       BTN_STATUS_OK             : When the operation is successful
|       BTN_STATUS_INVALID_STATE : Button structure pointer is a NULL_PTR
|       BTN_STATUS_INVALID_PULL_TYPE: If the pull type field in button
|                                      structure is set to invalid value
|
en_btn_status_code_t_ btn_init(st_btn_config_t_* ptr_st_btn_config);

|
| Function to read the current button state
|
| Parameters
|      [in]  ptr_str_btn_config : pointer to the desired button structure
|      [out] ptr_enu_btn_state  : pointer to variable to store the button state
|
| Return
|      BTN_STATUS_OK            : When the operation is successful
|      BTN_STATUS_INVALID_STATE : Btn cfg struct and/or btn state ptrs are NULL_PTRs
|      BTN_INVALID_PULL_TYPE    : pull type field in btn structure has invalid value
|      BTN_STATUS_DEACTIVATED   : If we read from a deactivated button
|
en_btn_status_code_t_ btn_read(st_btn_config_t_* ptr_st_btn_config, en_btn_state_t_*
ptr_en_btn_state);
```

2.3.3.3. PWM APIs

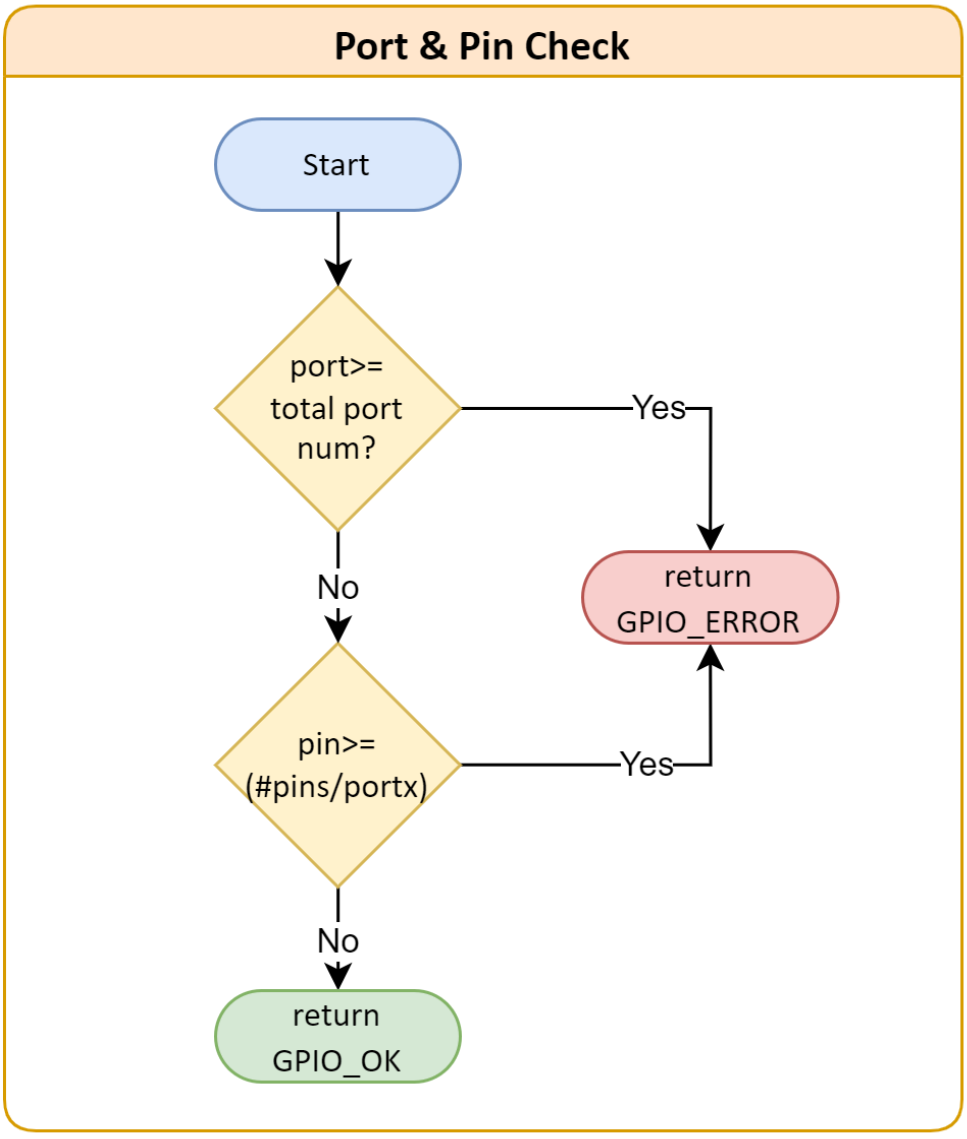
```
|
| Initializes pwm module with defined configurations
|
| Return
|       PWM_OK    : Operation is done successfully
|       PWM_ERROR : Operation Failed
|
en_pwm_error_t pwm_init(void);

|
| Adjust the signal parameters (period and duty cycle)
|
| Parameters
|          [in] en_a_channel_id : PWM channel to generate the signal on
|          [in] u8_a_dutyCycle : duty cycle of the PWM signal (0 - 100)
|          [in] u16_a_msPeriod : the period of the PWM signal
|
| Return
|       PWM_OK : Operation is done successfully
|       PWM_INVALID_CHANNEL : When the passed channel is not supported
|       PWM_ERROR : When the duty cycle is invalid (>100)
|
en_pwm_error_t pwm_adjust_signal(en_pwm_channel_id_t en_a_channel_id, uint8_t_
u8_a_dutyCycle, uint16_t_ u16_a_msPeriod);

|
| Start generating pwm signal on the given channel
|
| Parameters
|          [in] en_a_channel_id : PWM channel to generate the signal on
|
| Return
|       PWM_OK : Operation is done successfully
|       PWM_INVALID_CHANNEL : When the passed channel is not supported
|       PWM_ERROR : Operation failed
en_pwm_error_t pwm_start(en_pwm_channel_id_t en_a_channel_id);

|
| Stop generating a pwm signal on the given channel
|
| Parameters
|          [in] en_a_channel_id : PWM channel to stop
|
| Return
|       PWM_OK : Operation is done successfully
|       PWM_INVALID_CHANNEL : When the passed channel is not supported
|       PWM_ERROR : Operation failed

en_pwm_error_t pwm_stop (en_pwm_channel_id_t en_a_channel_id);
```

3. Low Level Design

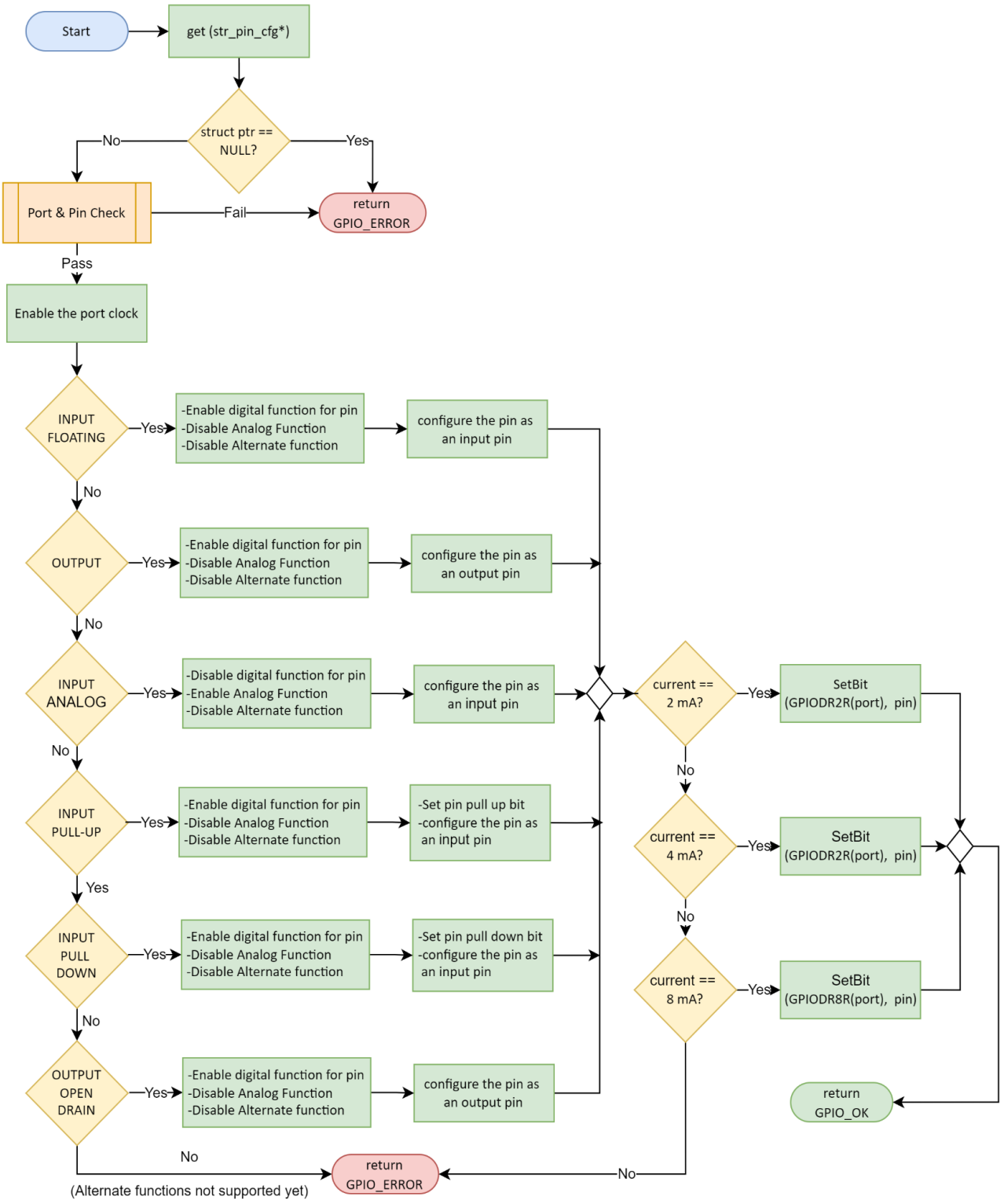
3.1. MCAL Layer

3.1.1. GPIO Module

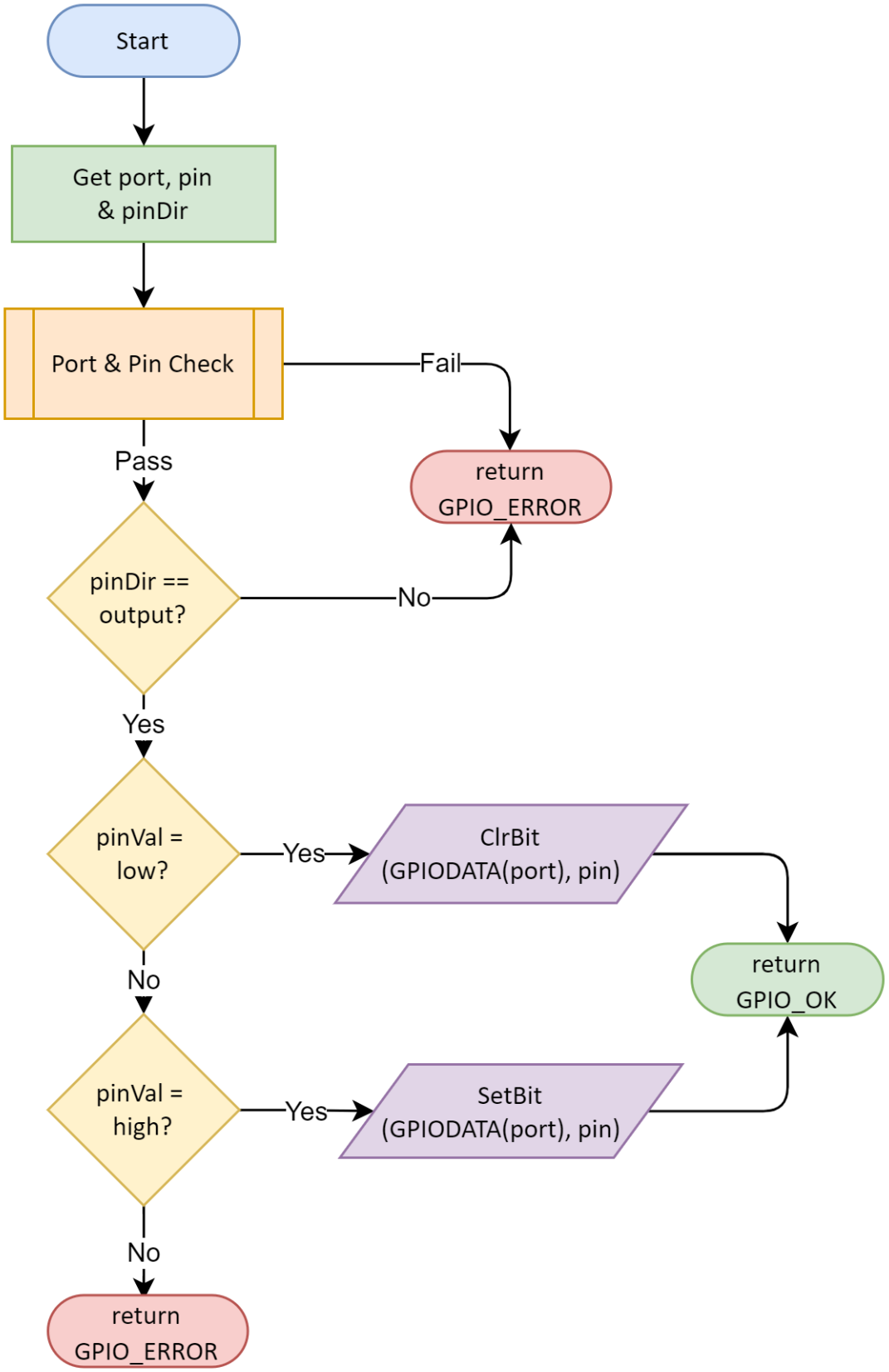
3.1.1.a. sub process



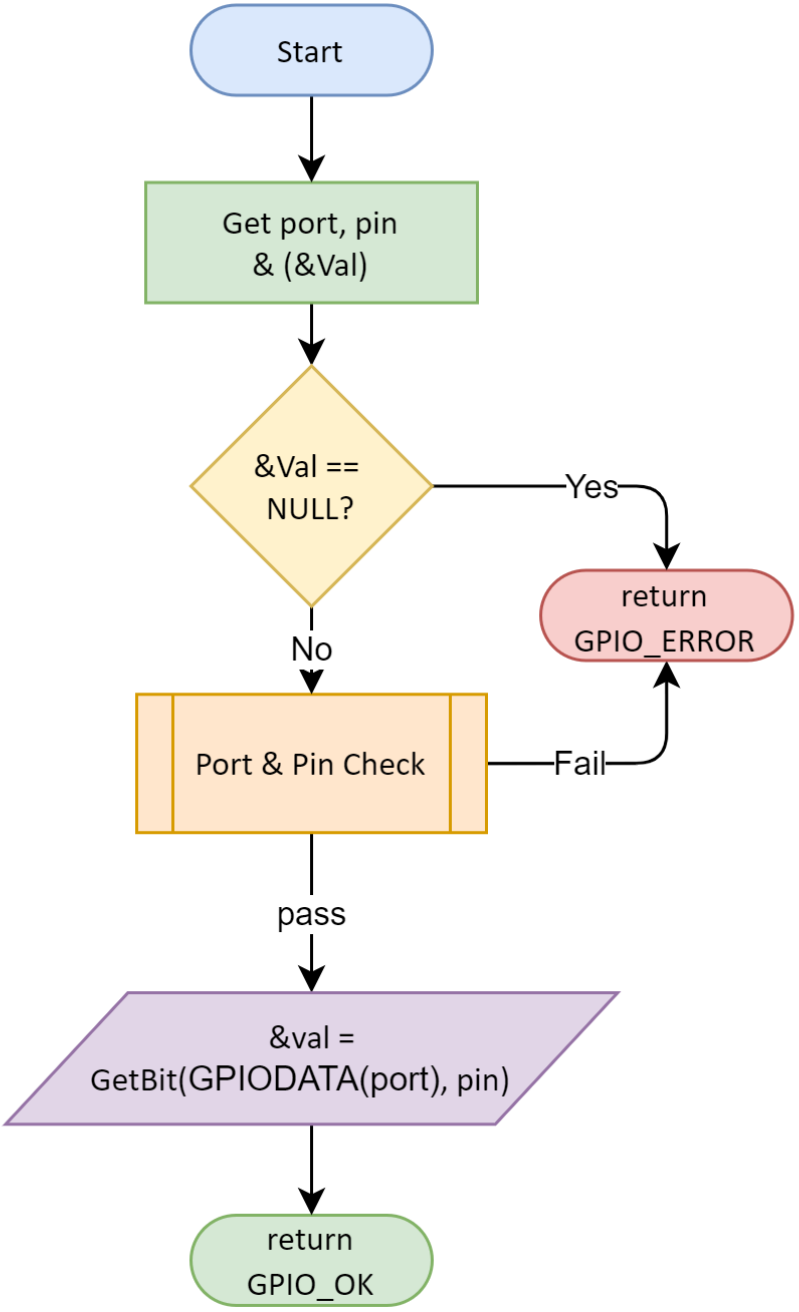
3.1.1.1. GPIO_pin_init



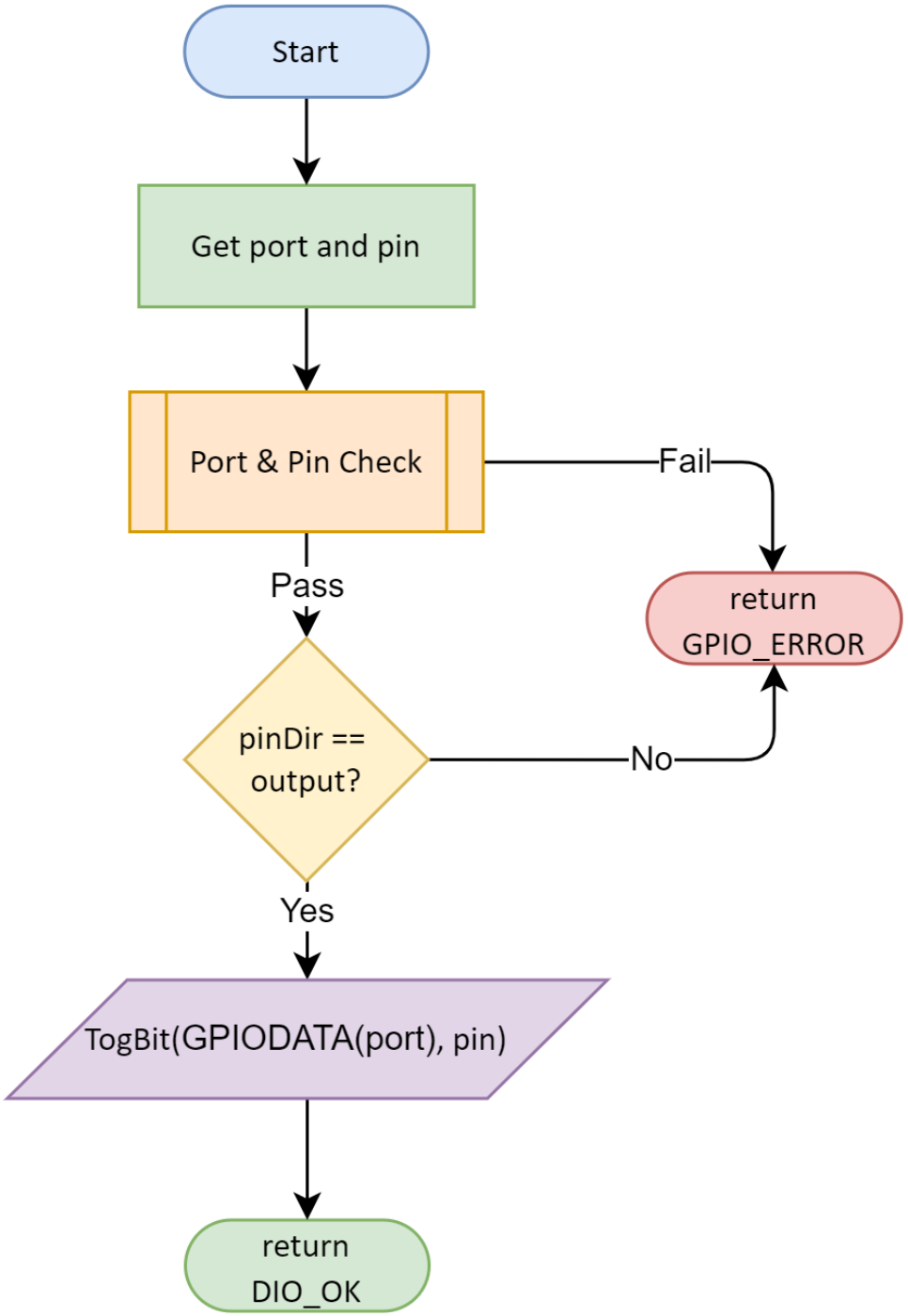
3.1.1.2. GPIO_setPinVal



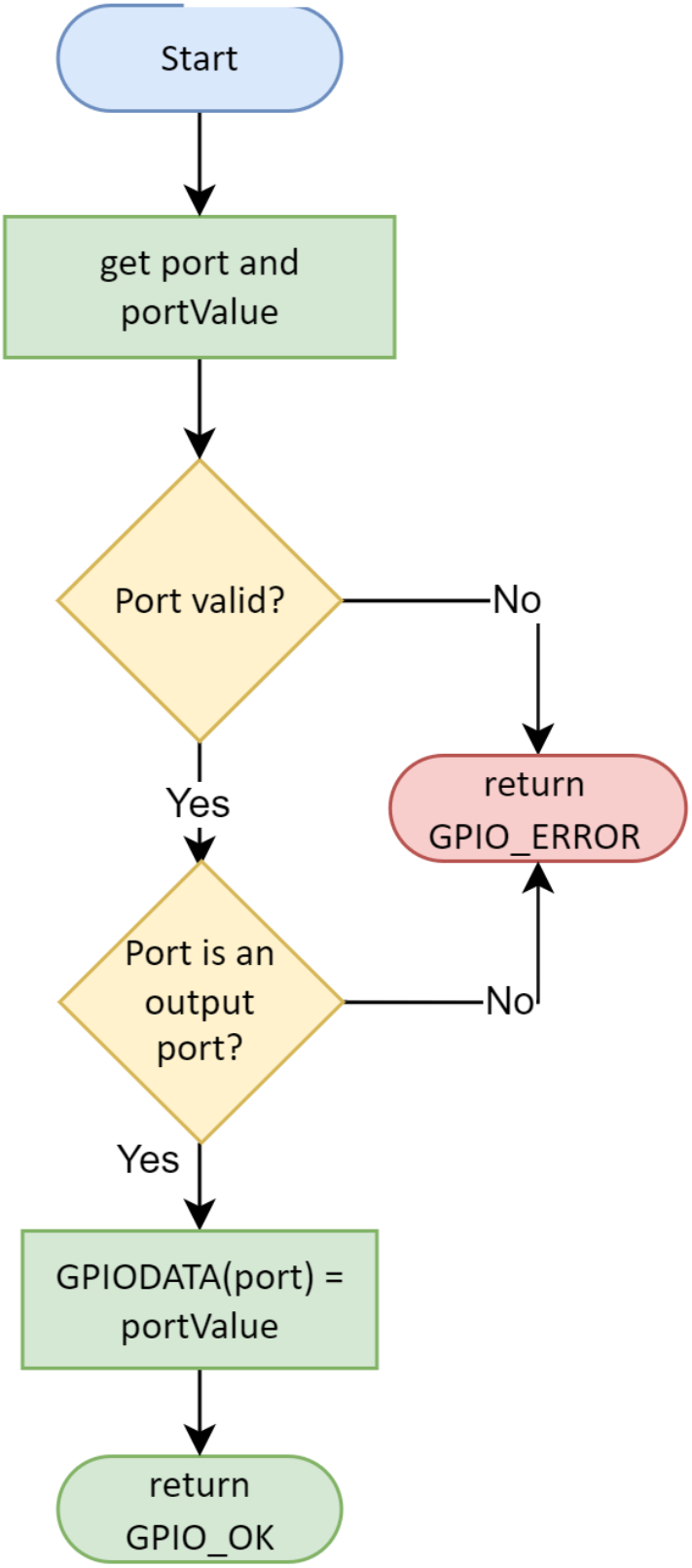
3.1.1.3. GPIO_getPinVal



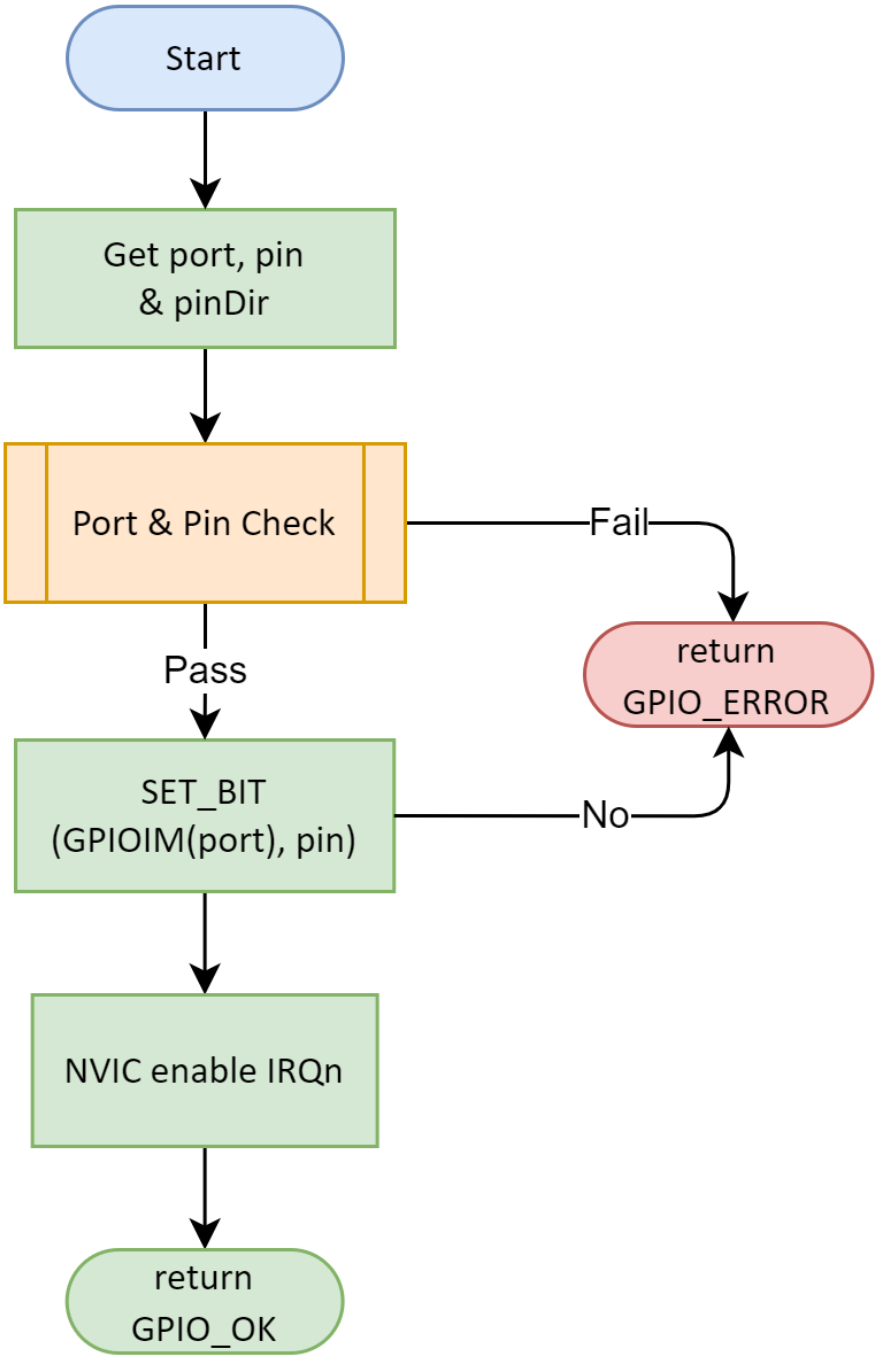
3.1.1.4. GPIO_togPinVal



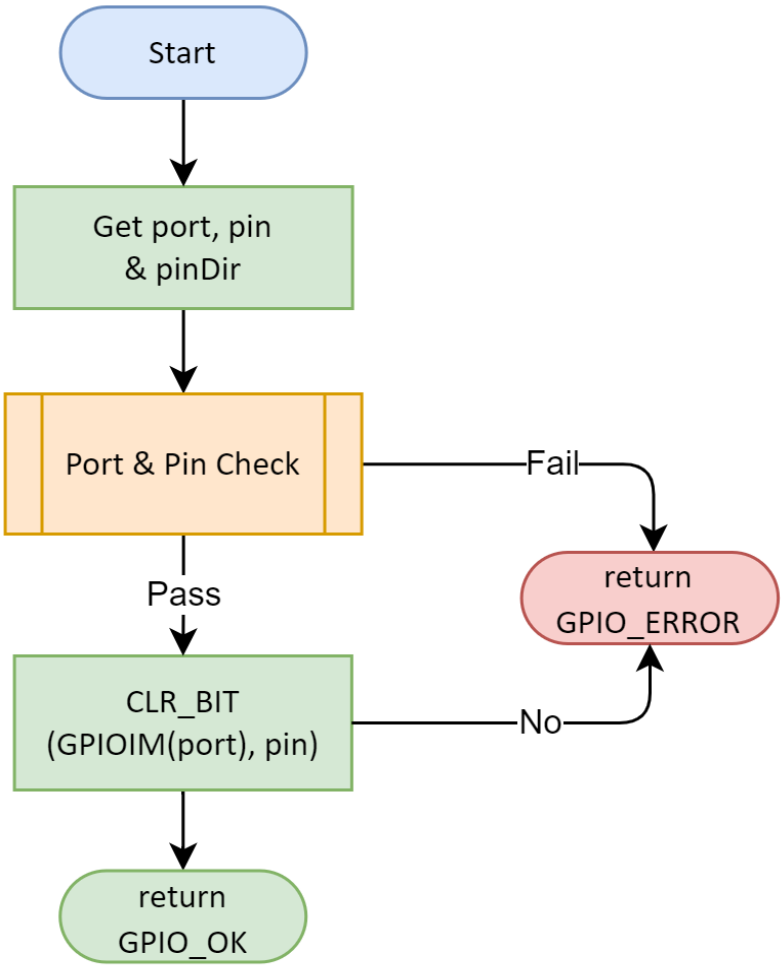
3.1.1.5. GPIO_setPortVal



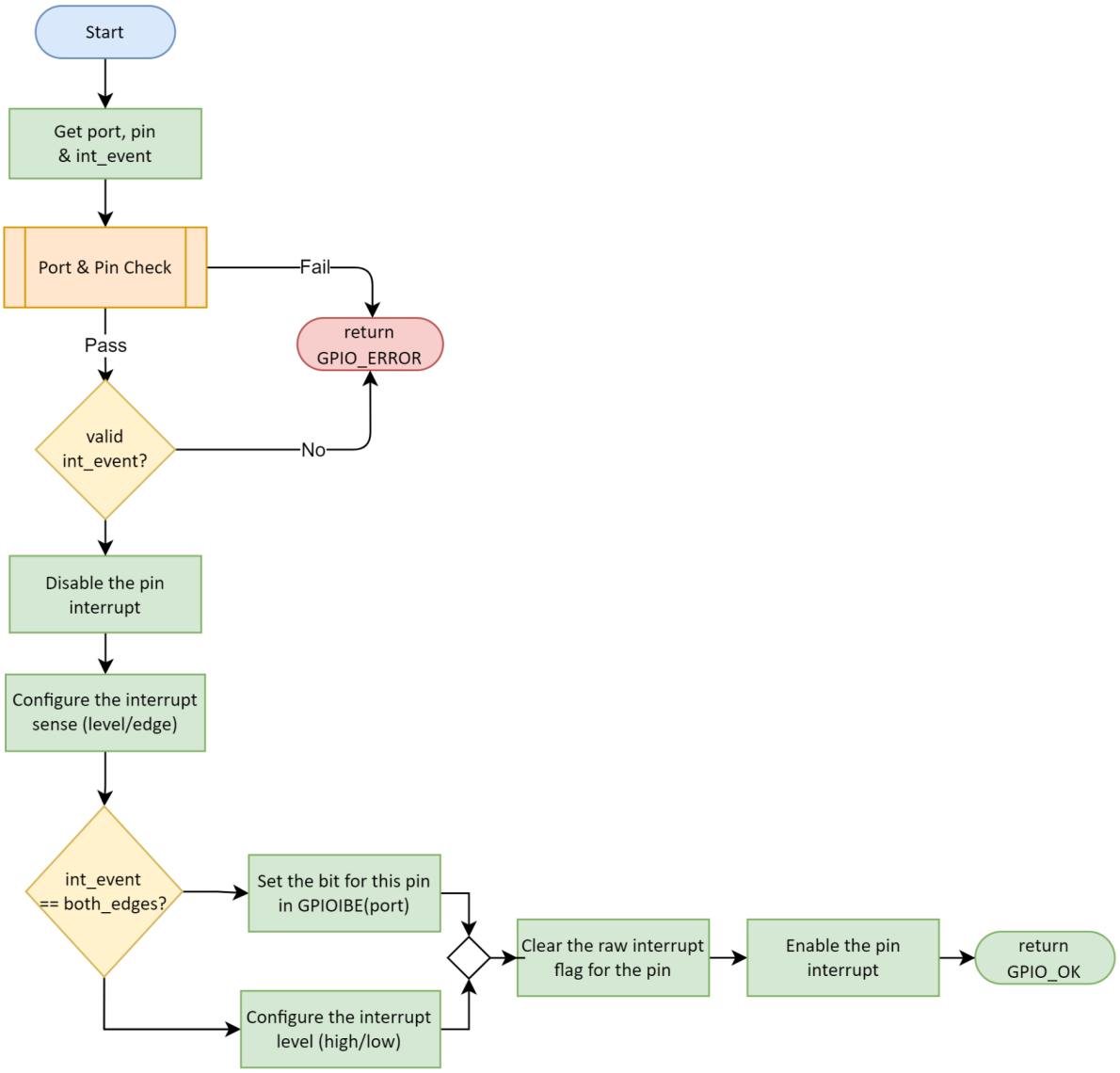
3.1.1.6. GPIO_enableInt



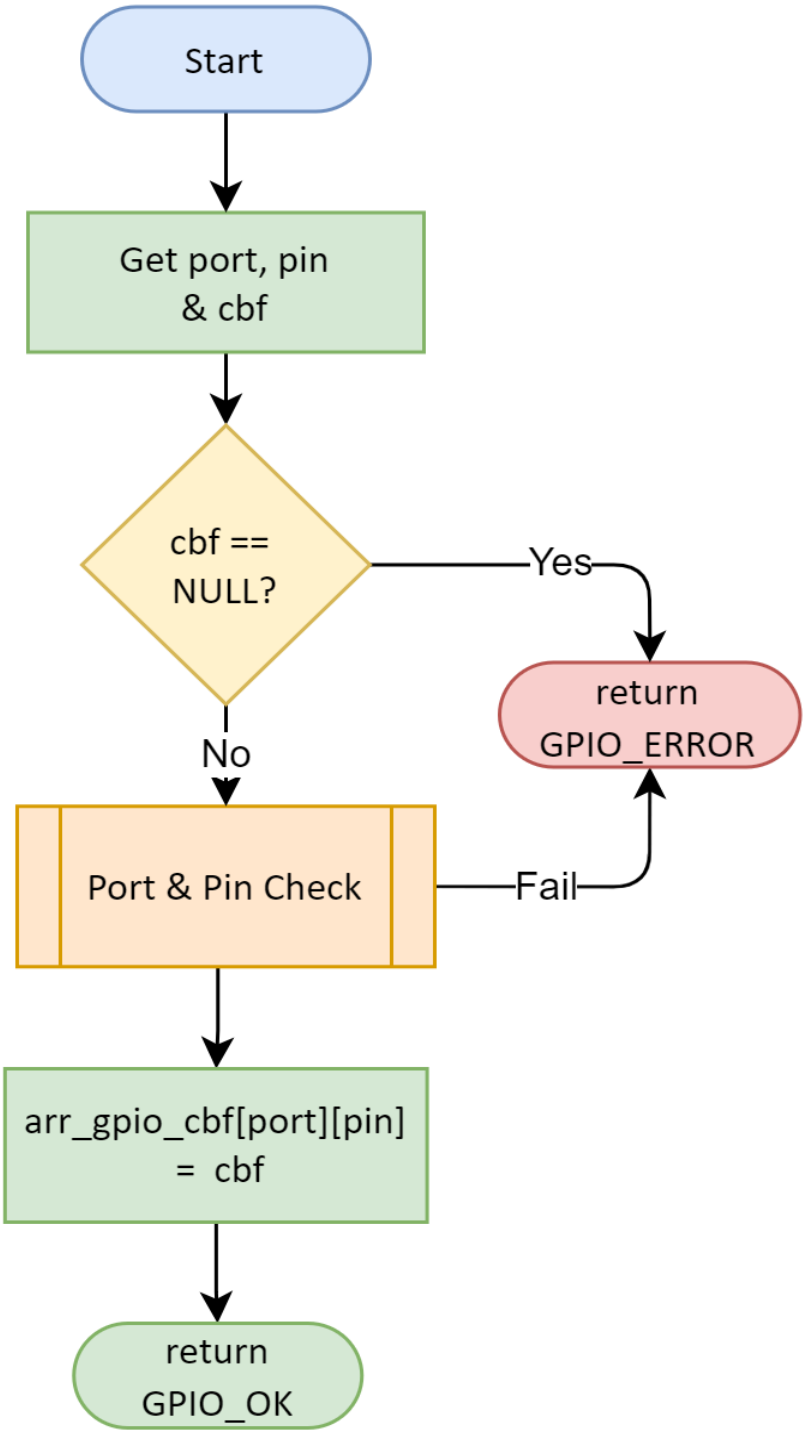
3.1.1.7. GPIO_disableInt



3.1.1.8. GPIO_setIntSense

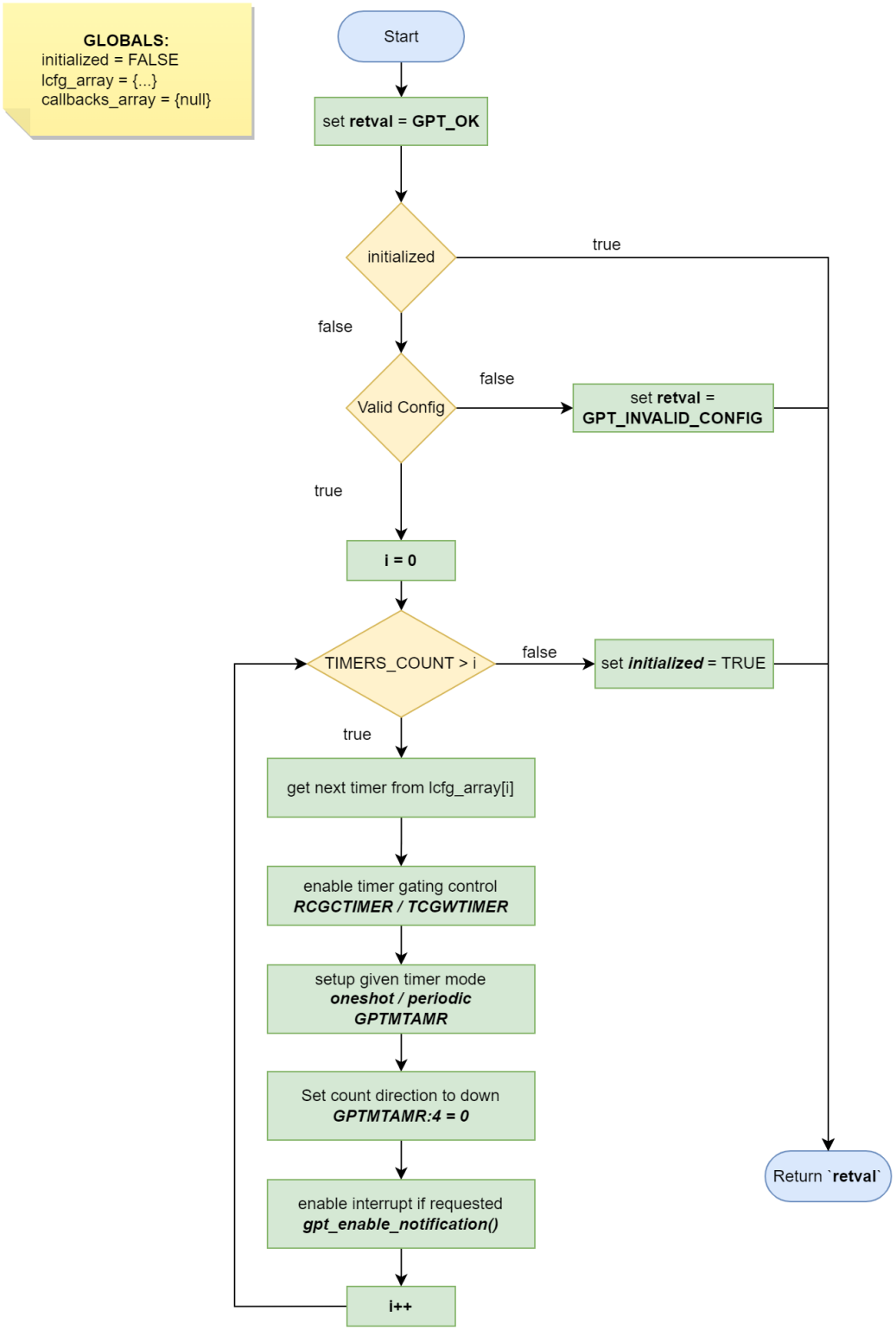


3.1.1.9. GPIO_setIntCallback

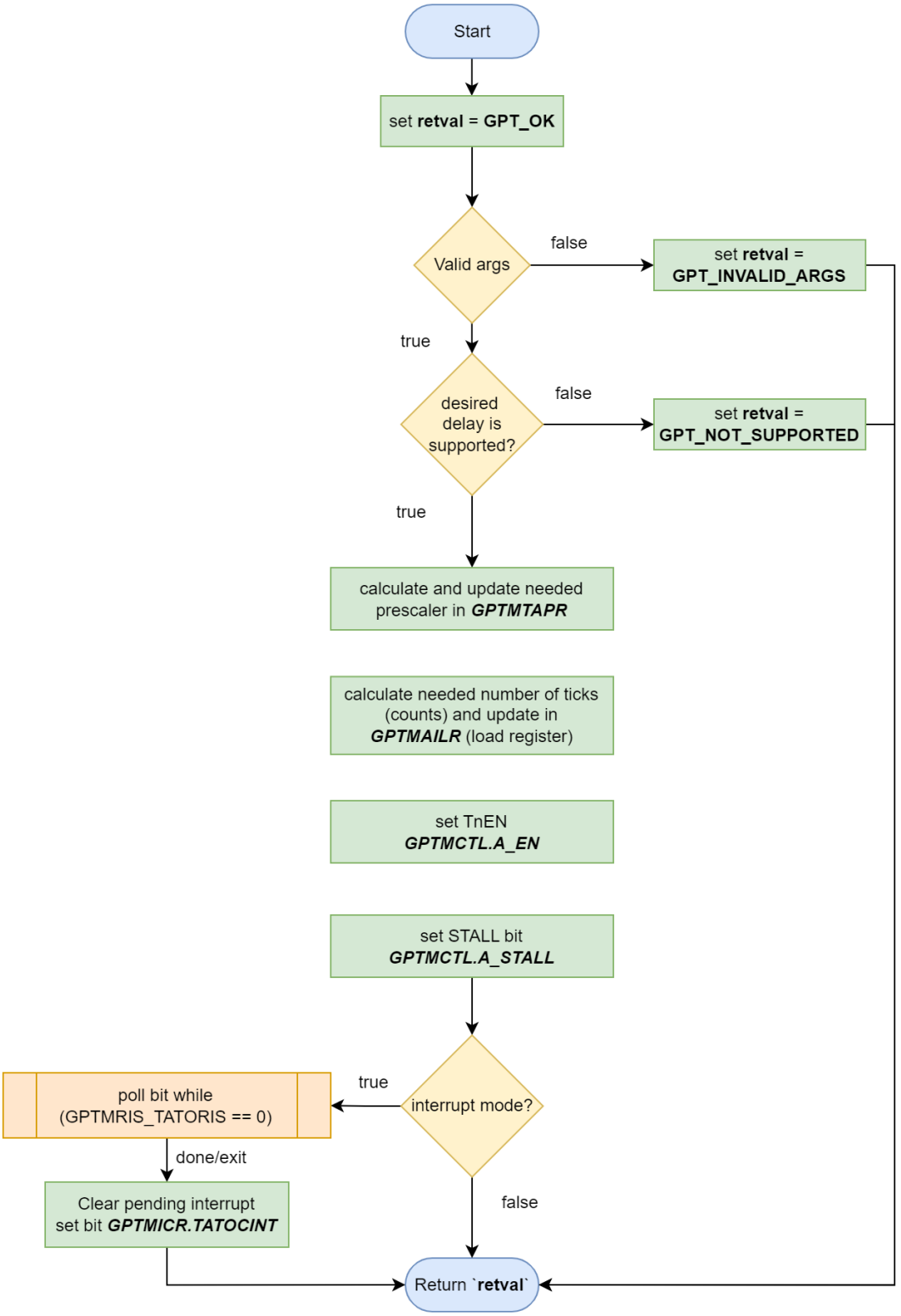


3.1.2. GPT (General Purpose Timer)

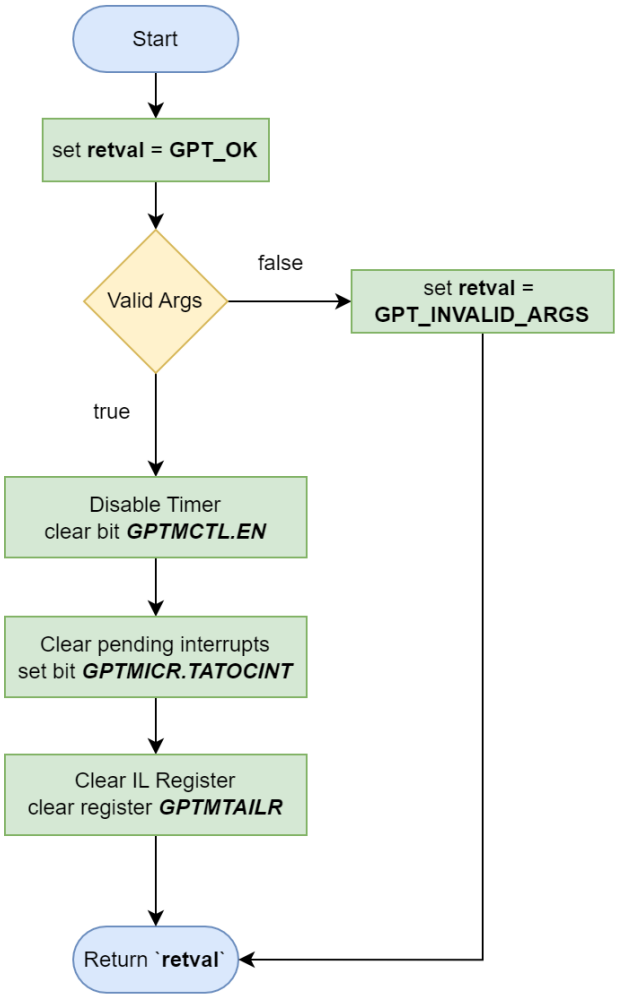
3.1.2.1. gpt_init



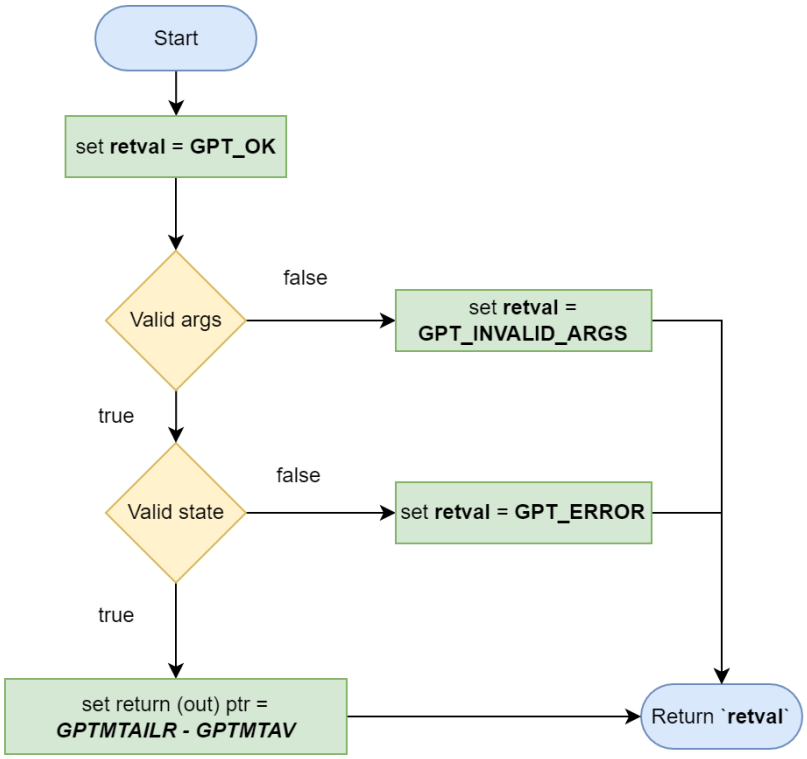
3.1.2.2. gpt_start



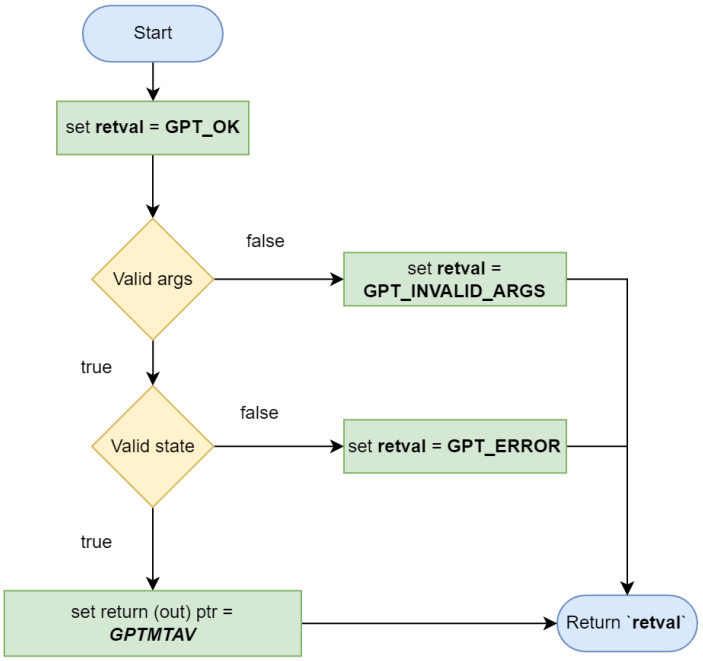
3.1.2.3. gpt_stop



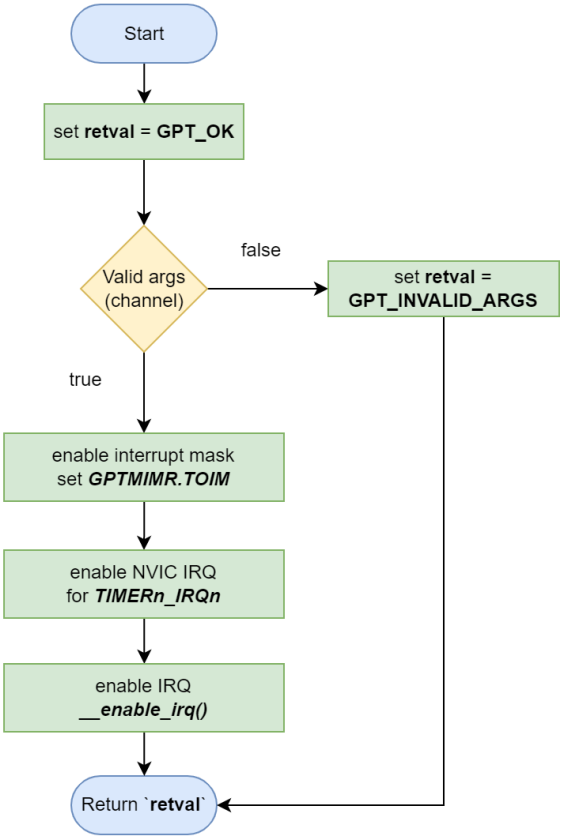
3.1.2.4. gpt_get_elapsed_time



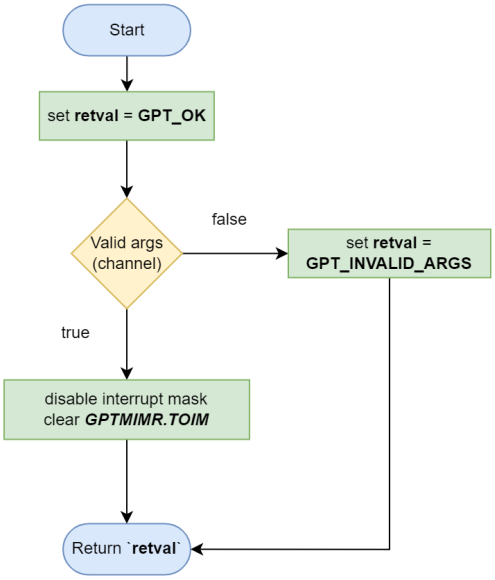
3.1.2.5. gpt_get_remaining_time



3.1.2.6. gpt_enable_notification



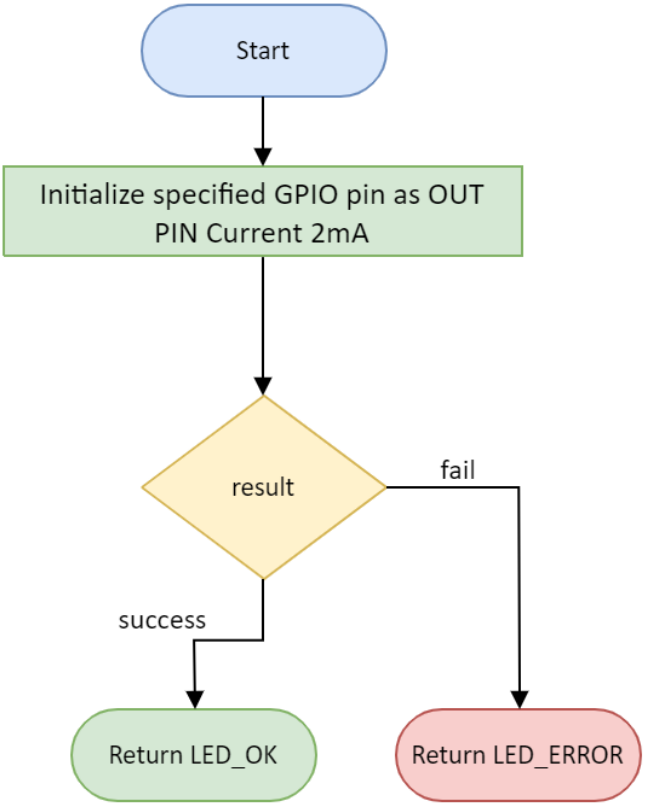
3.1.2.7. gpt_disable_notification



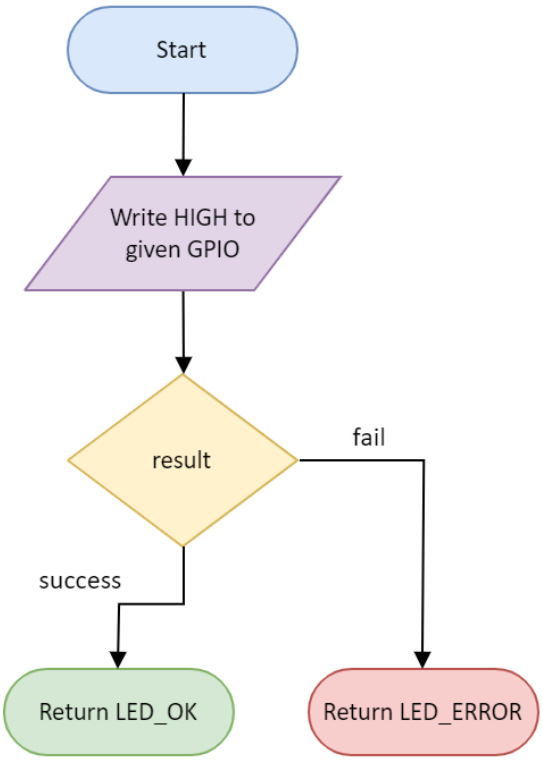
3.2. HAL Layer

3.2.1. LED Module

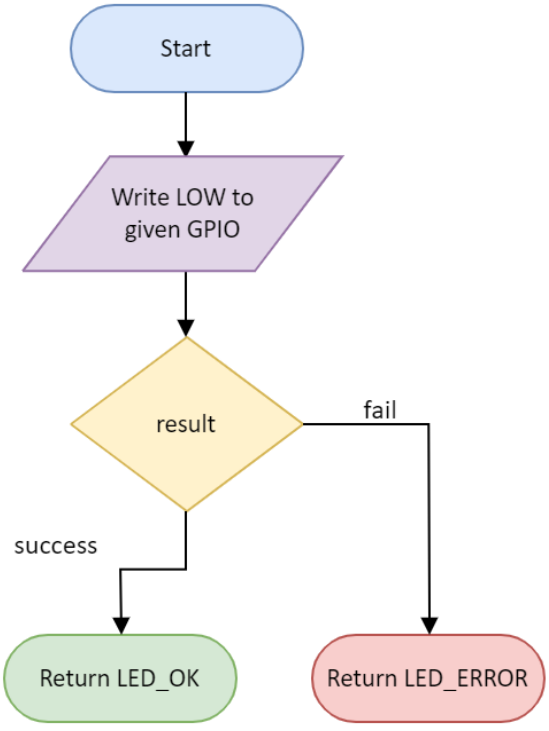
3.2.1.1. led_init



3.2.1.2. led_on

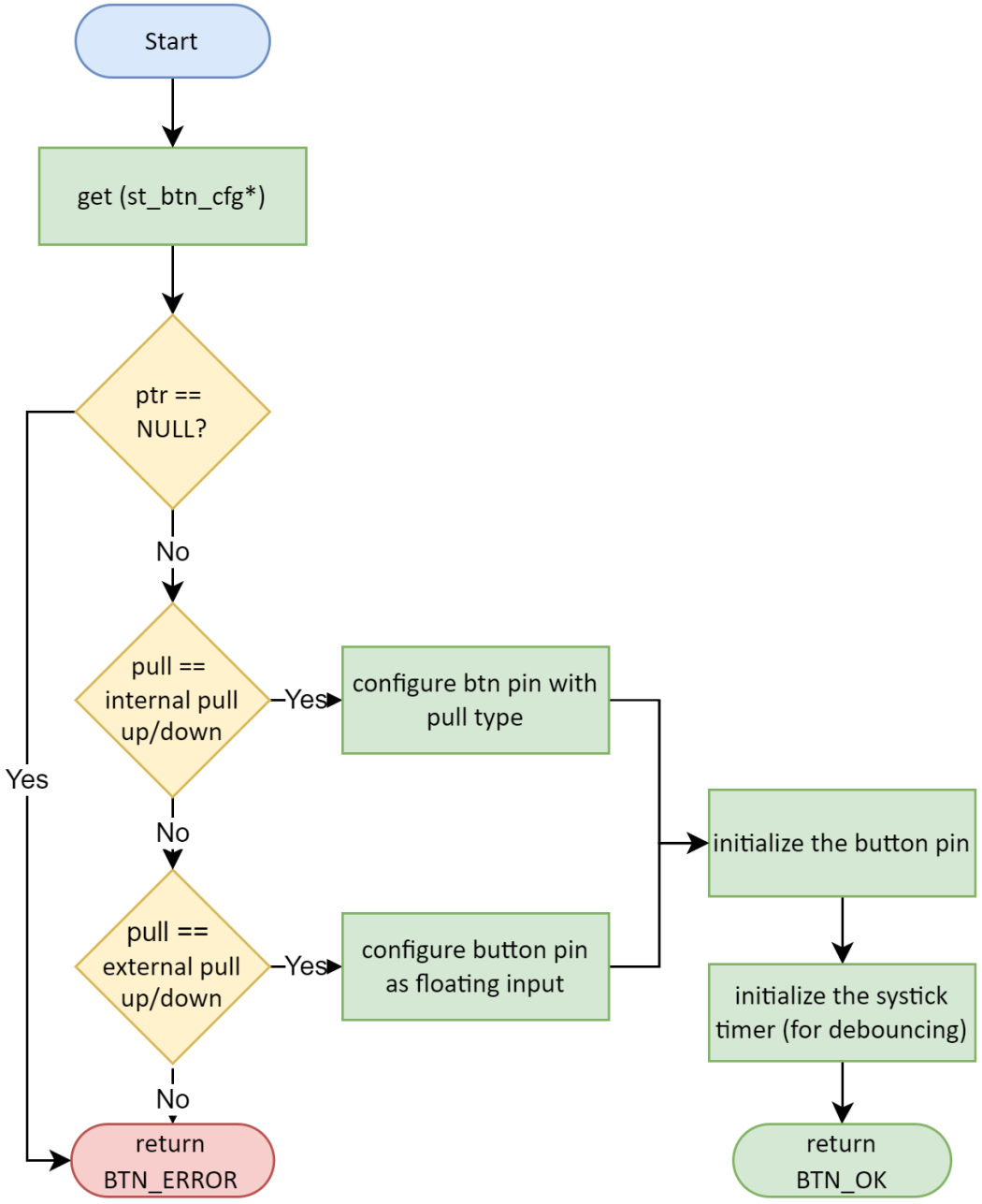


3.2.1.3. led_off

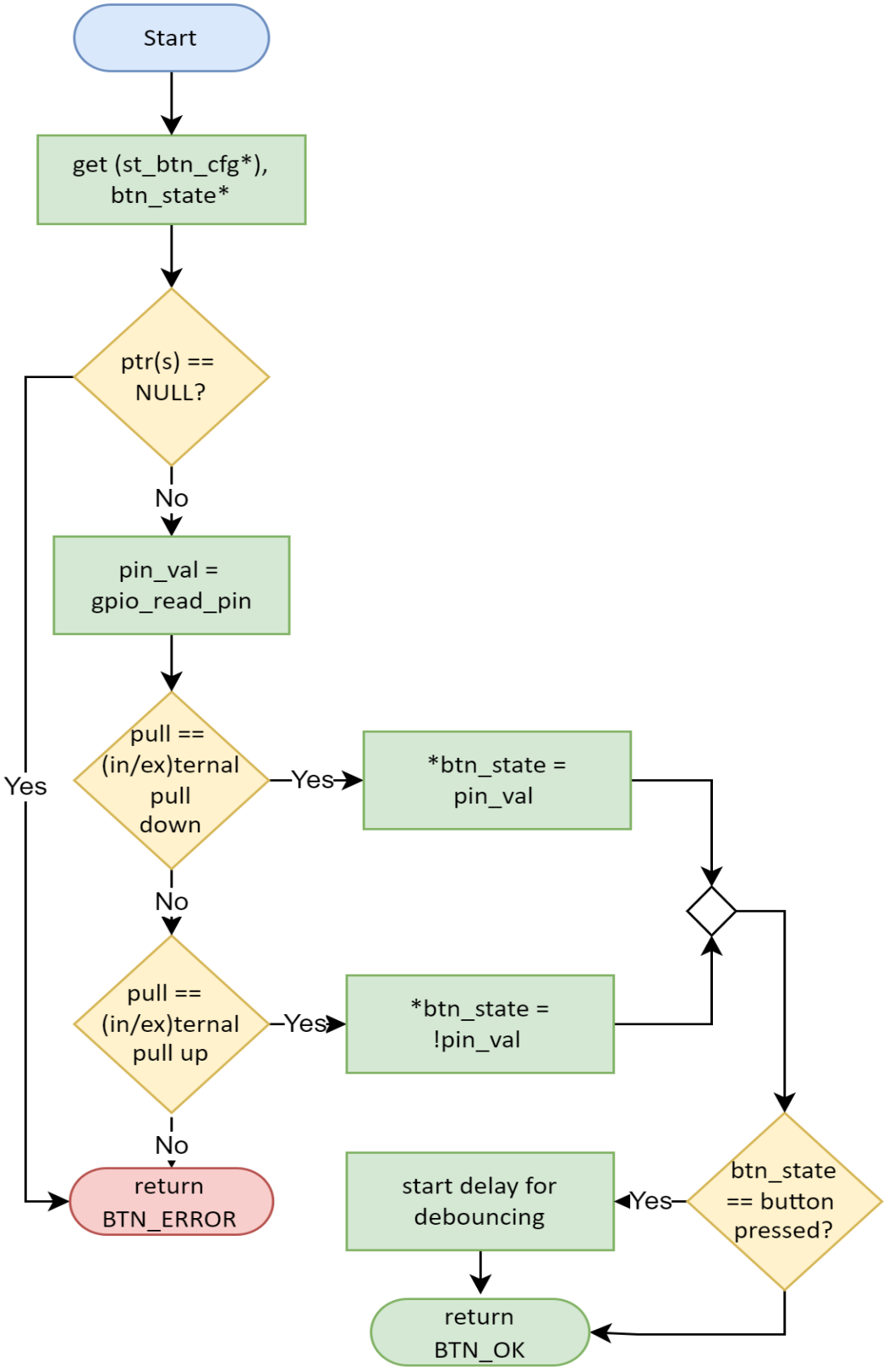


3.2.2. BTN Module

3.2.2.1. BUTTON_init

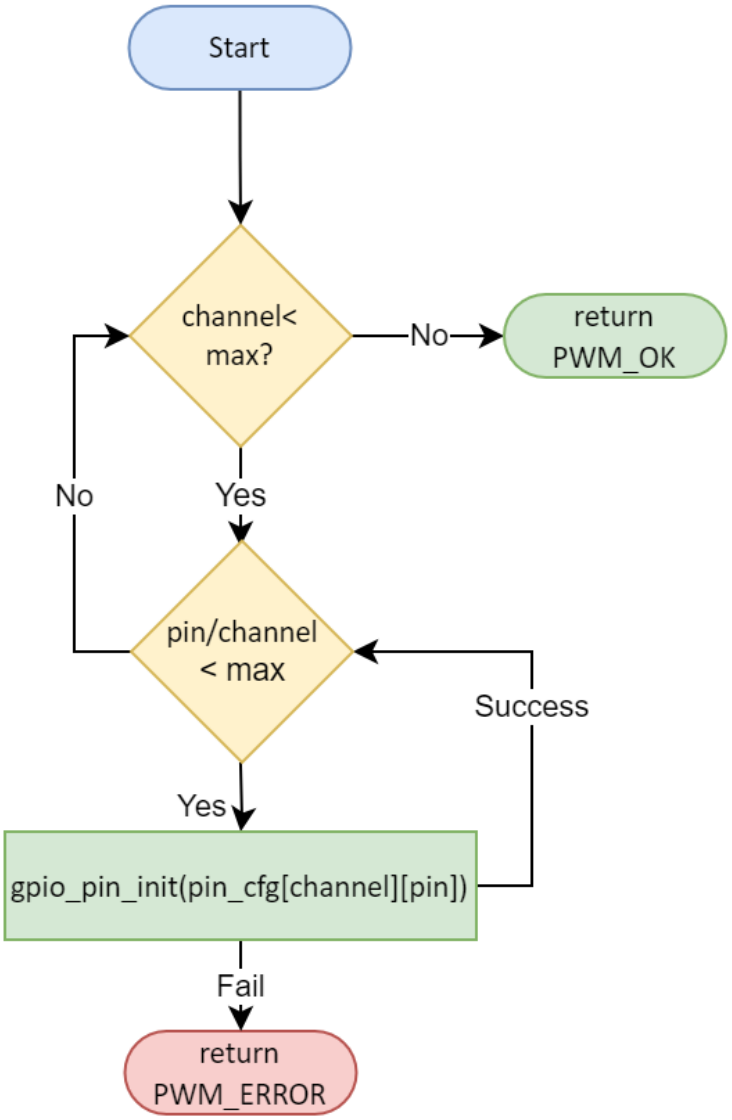


3.2.2.2. BUTTON_read

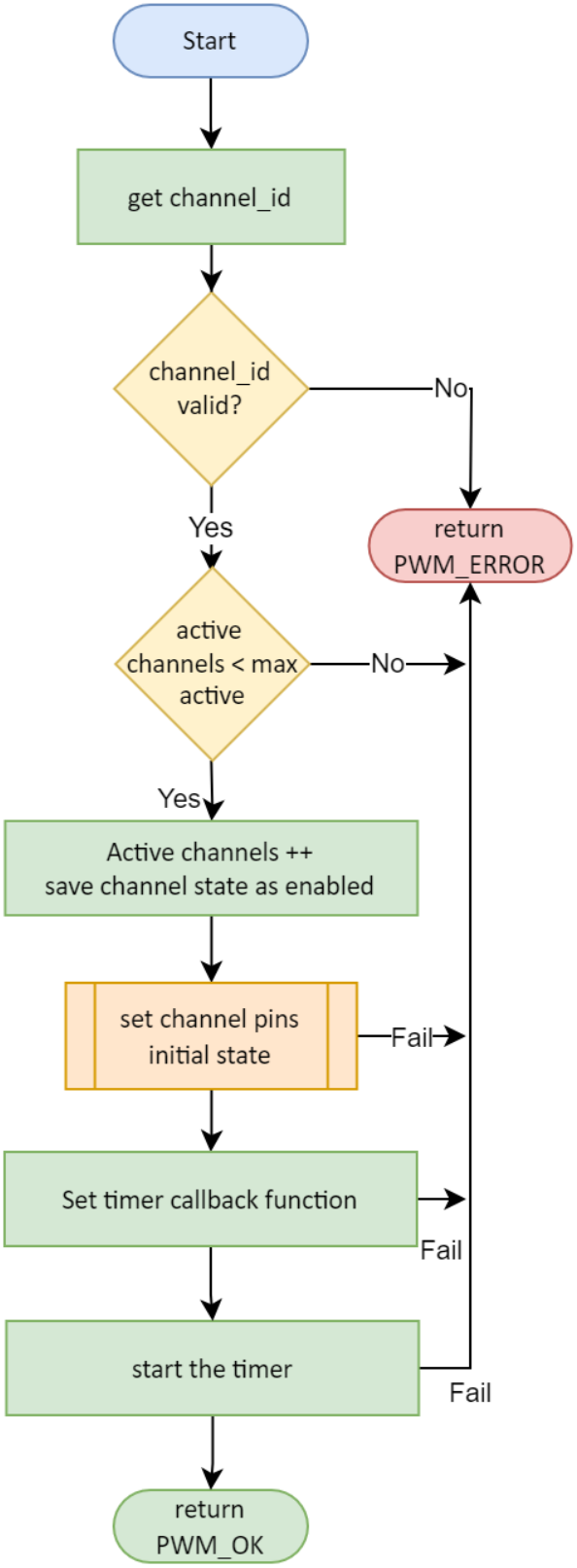


3.2.3. PWM Module

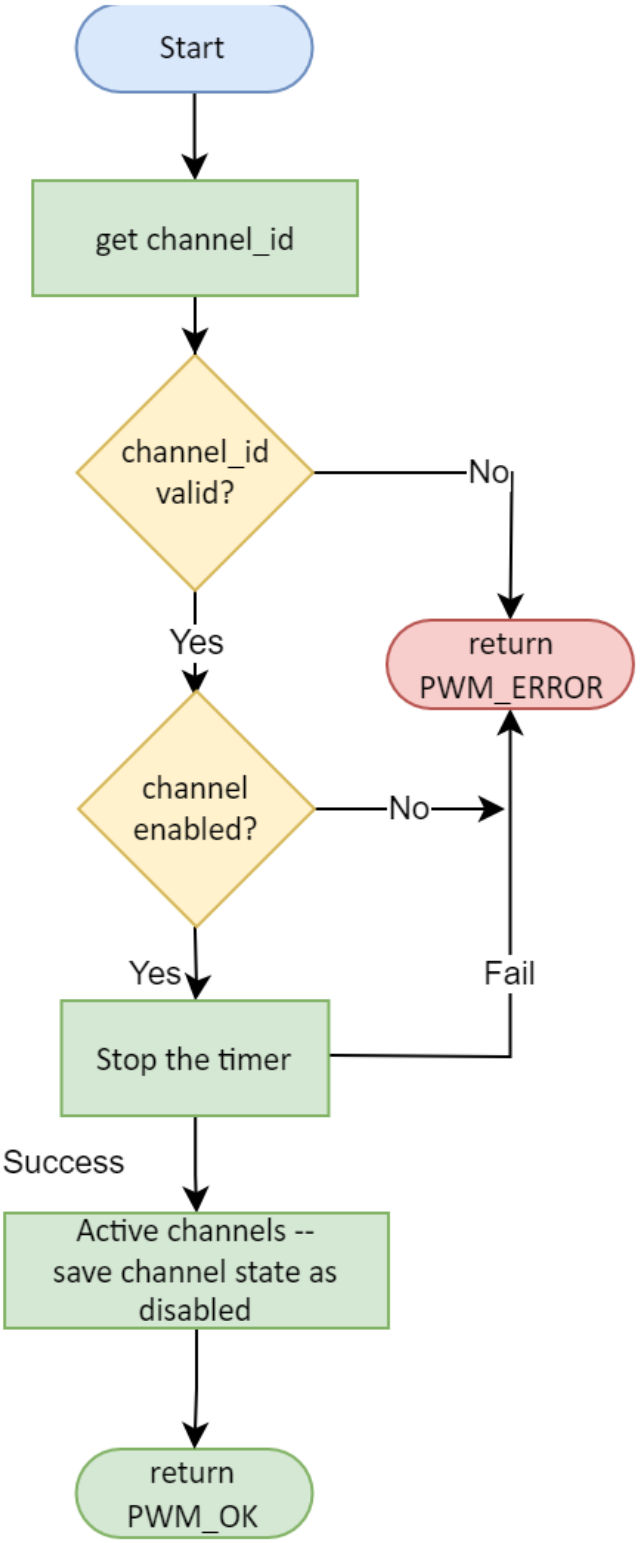
3.2.3.1. Pwm_init



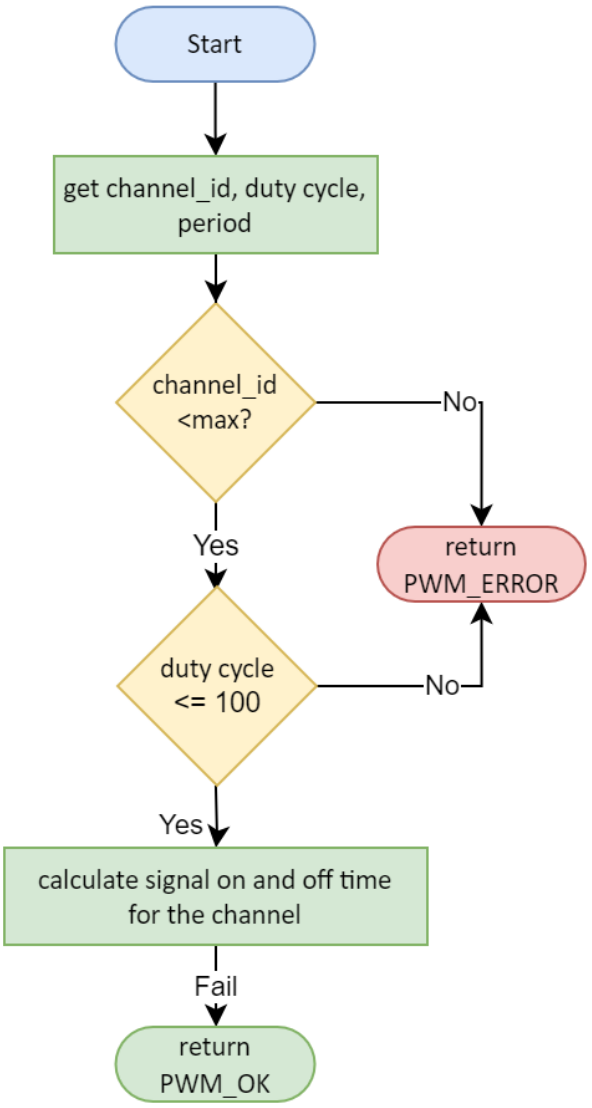
3.2.3.2. Pwm_start



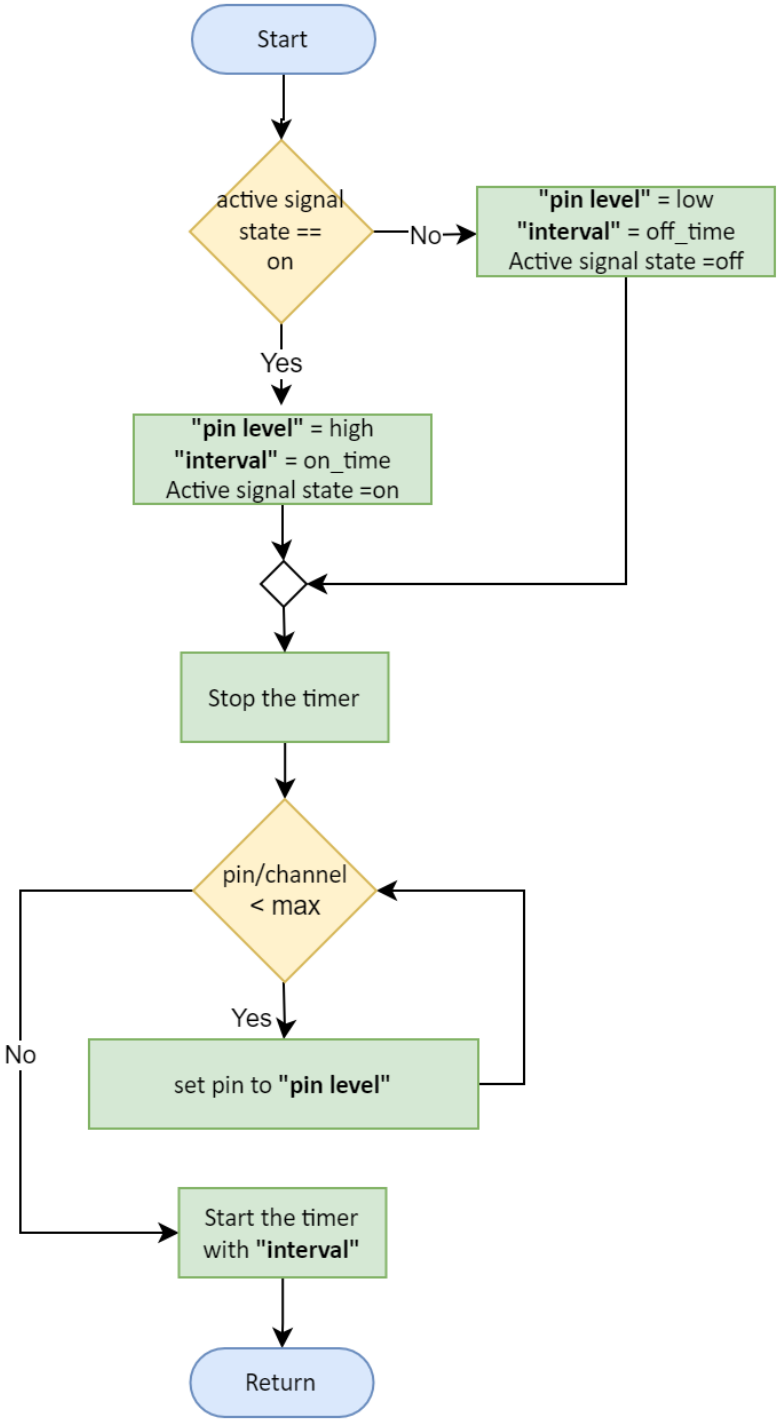
3.2.3.3. Pwm_stop



3.2.3.4. pwm_adjust_signal

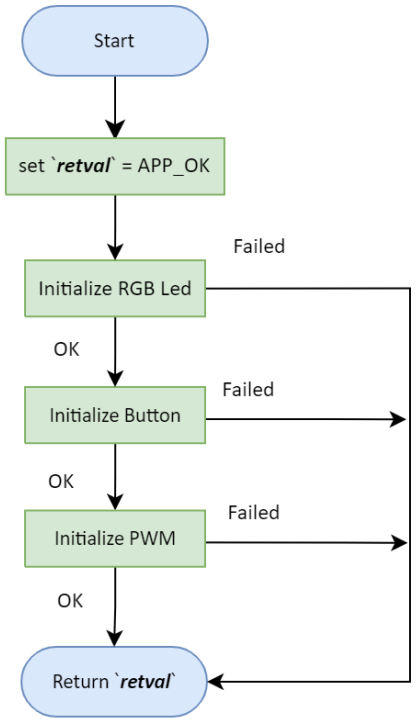


3.2.3.5. Pwm_timer_cbf

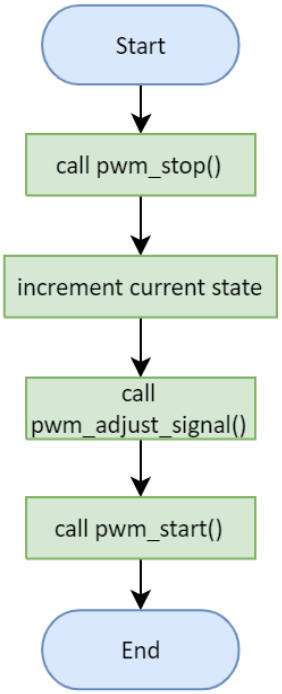


3.3. APP Layer

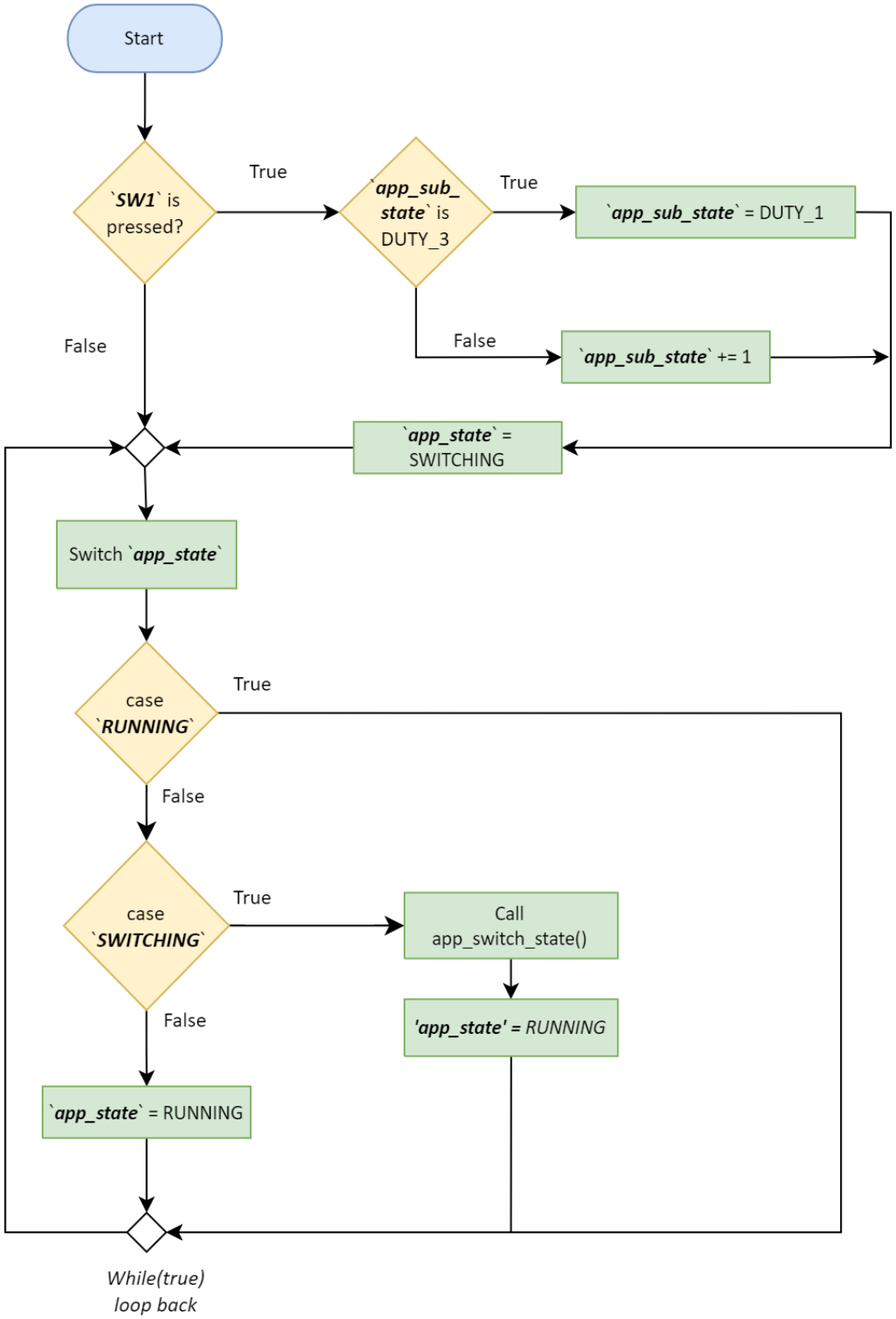
3.3.1. app_init



3.3.1. app_next_state



3.3.3. app_start



4. Pre-compiling and linking configurations

4.1. GPIO Driver

None

4.2. GPT Driver

4.2.1. Pre-compiled Configurations

```
#define     MC_F_CPU_HZ      16000000UL

#define GPT_CONFIGURED_TIMERS_CHS_COUNT     3
```

4.2.2. Linking configuration

```
typedef enum{
    CH_MODE_ONE_SHOT = 0    ,
    CH_MODE_PERIODIC        ,
    CH_MODE_TOTAL
}en_gpt_channel_mode_t;


typedef enum{
    GPT_INT_ENABLED = 0     ,
    GPT_INT_DISABLED        ,
    GPT_INT_TOTAl
}en_gpt_int_enabled_t;


typedef struct gpt_config {
    en_gpt_channel_t        en_gpt_channel_id;
    en_gpt_channel_mode_t   en_gpt_channel_mode;
    en_gpt_int_enabled_t    en_gpt_int_enabled;
}st_gpt_config_t;


st_gpt_config_t gl_st_gpt_lconfig_arr[GPT_CONFIGURED_TIMERS_CHS_COUNT] = {
        {
            CH_0,
            CH_MODE_ONE_SHOT,
         GPT_INT_ENABLED
         },
         {
            CH_1,
            CH_MODE_PERIODIC,
            GPT_INT_ENABLED
         },
         {
            CH_2,
            CH_MODE_PERIODIC,
            GPT_INT_ENABLED
         },
};
```

4.3. LED Driver

None.

4.4. BTN Driver

None.

4.5. PWM Driver

4.5.1. Pre-compiled Configuration

```
#define PWM_MAX_CHANNELS 4
#define PWM_MAX_PINS_PER_CHANNEL 3
```

4.5.2. Linking configuration

```
const st_pwm_signal_cfg_t arr_gl_st_signal_cfg[PWM_MAX_CHANNELS] =
{
{{{(en_pwm_port_t)GPIO_PORT_F, (en_pwm_pin_t)GPIO_PIN_1}, {NULL}, {NULL}}},
{{{(en_pwm_port_t)GPIO_PORT_F, (en_pwm_pin_t)GPIO_PIN_2}, {NULL}, {NULL}}},
{{{(en_pwm_port_t)GPIO_PORT_F, (en_pwm_pin_t)GPIO_PIN_3}, {NULL}, {NULL}}},
{{{NULL}, {NULL}, {NULL}}}
};
```

5. References

1. Draw IO
2. Layered Architecture | Baeldung on Computer Science
3. Microcontroller Abstraction Layer (MCAL) | Renesas
4. Hardware Abstraction Layer - an overview | ScienceDirect Topics
5. What is a module in software, hardware and programming?
6. Embedded Basics – API’s vs HAL’s
7. https://ti.com/launchpad