Sprints

RGB LED CONTROL V1.0

ARM

Develop the GPIO Driver and use it to control RGB LED on the Tiva-C board using a push button.

Prepared By

Team 1 - Sub Team A

- Alaa Hisham
- Hossam Elwahsh

RGB LED Control V1.0

1. Project Introduction

Develop the GPIO Driver and use it to control a single RGB LED on the Tiva-C (TM4C123G) board using a push button.

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
On Pressing SW1:

- After the first press, the **Red led** is on
- After the second press, the **Green led** is on
- After the third press, the **Blue led** is on
- After the fourth press, **all LEDs** are on
- After the fifth press, should **disable all LEDs**
- After the sixth press, **repeat** steps from 1 to 6

Implement your drivers

- Implement GPIO driver
- Implement LED driver
- Implement Button 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

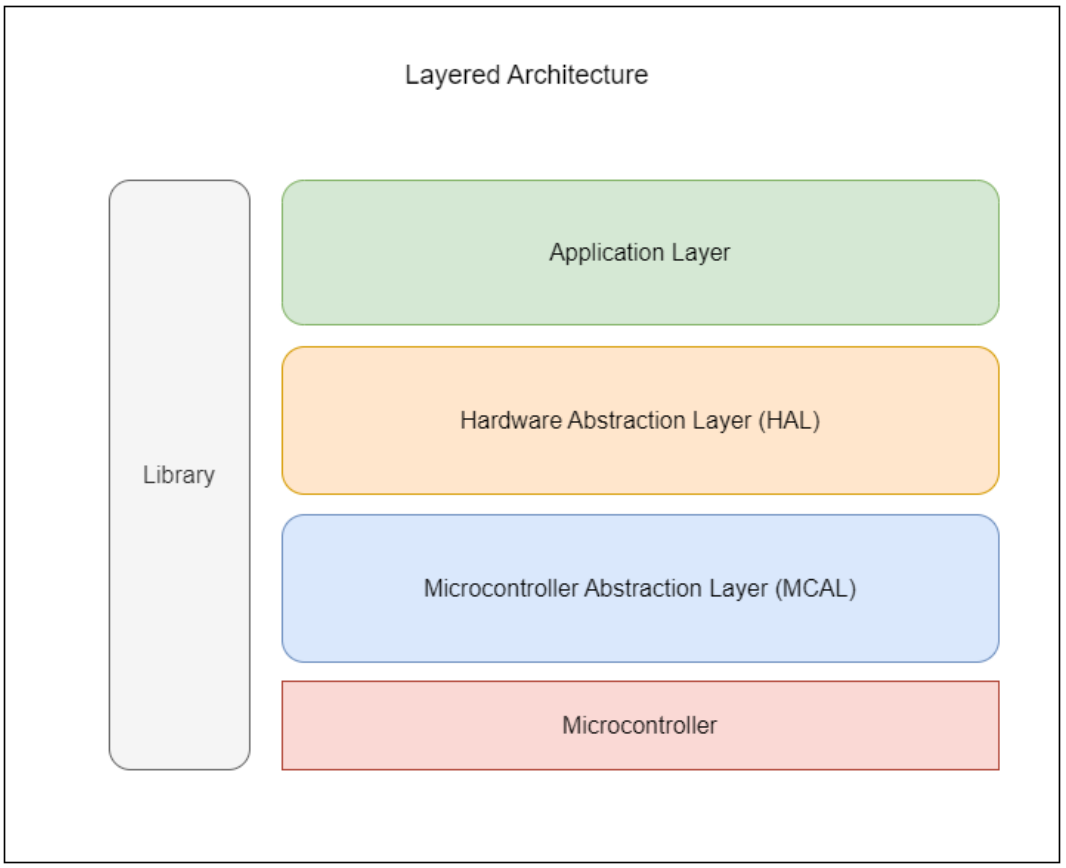
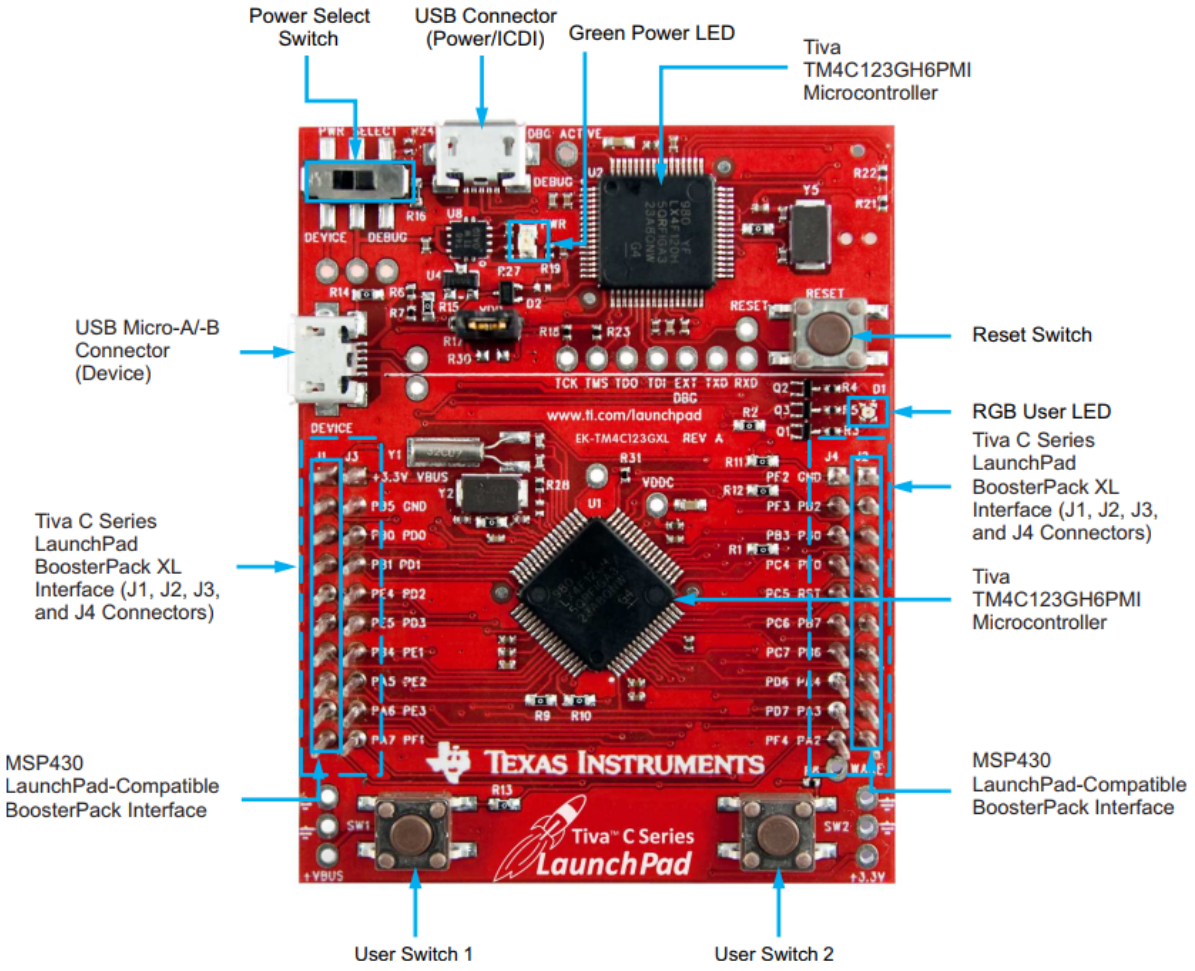


Figure 1. Layered Architecture Design

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. SYSTICK Module

The SysTick driver in the Tiva C TM4C123G microcontroller is a timer module specifically designed for providing accurate timing and generating periodic interrupts. It is a versatile tool that enables precise timekeeping, real-time event scheduling, and system timing synchronization.

2.2.3. 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.

2.2.4. 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.

2.2.5. Design

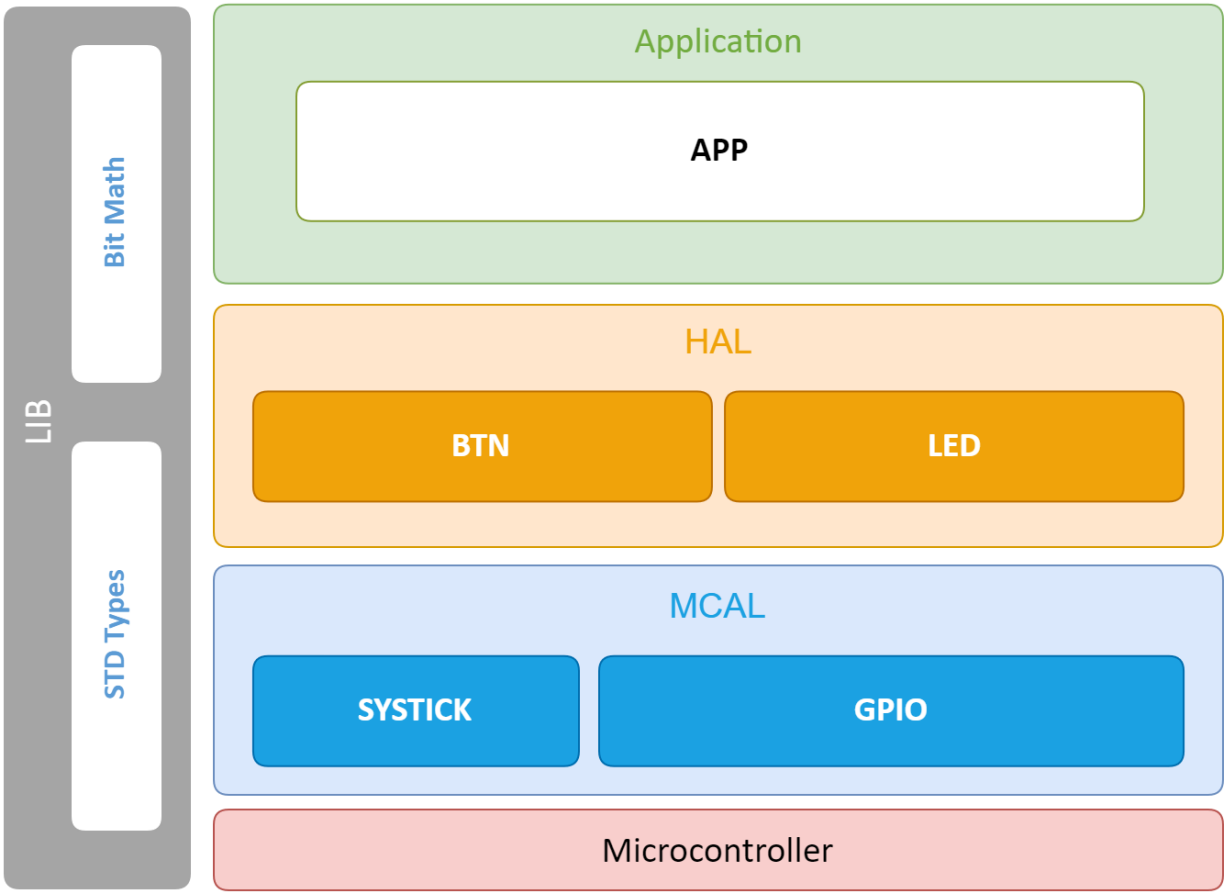


Figure 3. System Modules Design

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

```
| @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. SYSTICK Driver

```
/*-----------------------------------------------------------*/
/- ENUMS
/-----------------------------------------------------------*/
typedef enum{
    /* precision internal oscillator / 4 (divided by 4) */
    CLK_SRC_PIOSC    =   0   ,

    /* system clock */
    CLK_SRC_SYS_CLK          ,

    CLK_SRC_TOTAL
}en_systick_clk_src_t;

typedef enum{
    ST_OK               =   0   ,
    ST_INVALID_CONFIG           ,
    ST_INVALID_ARGS             ,
}en_systick_error_t;



/*-----------------------------------------------------------*/
/- STRUCTURES
/-----------------------------------------------------------*/
typedef struct{

    /**
     * False: Interrupt generation is disabled. Software can use the
     *      COUNT bit to determine if the counter has ever reached 0
     *
     * True: An interrupt is generated to the NVIC when SysTick counts to 0.
     *
     * */
    boolean bool_systick_int_enabled;

    en_systick_clk_src_t en_systick_clk_src;

}st_systick_cfg_t;
```

```
/*-------------------------------------------------------------*/
/- Functions Prototypes
/-------------------------------------------------------------*/


| Initializes SYSTICK driver
|
| Parameters
|           ptr_st_systick_cfg    : Pointer to Systick Configuration
|
|
| Return
|        ST_OK              In case of Successful Operation
|        ST_INVALID_ARGS    In case of Failed Operation (Invalid
|                                                        Arguments Given)
|        ST_INVALID_CONFIG  In case of Failed Operation (Invalid Systick
|                                                        Config Given)
|
en_systick_error_t systick_init(st_systick_cfg_t * ptr_st_systick_cfg);


| Initiates a sync blocking delay
|
| Parameters
|           uint32_ms_delay      : Desired delay in ms
|
| Return
|        ST_OK              In case of Successful Operation
|        ST_INVALID_ARGS    In case of Failed Operation (Invalid
|                                                        Arguments Given)
|        ST_INVALID_CONFIG  In case of Failed Operation (Invalid Systick
|                                                        Config Given)
|
en_systick_error_t systick_ms_delay(uint32_t_ uint32_ms_delay);
```

2.3.3. HAL APIs

2.3.3.1. LED APIs

```
/* LED Pins */
typedef enum{
    LED_PIN_0     =     0     ,
    LED_PIN_1                 ,
    LED_PIN_2                 ,
    LED_PIN_3                 ,
    LED_PIN_4                 ,
    LED_PIN_5                 ,
    LED_PIN_6                 ,
    LED_PIN_7                 ,
    LED_PIN_TOTAL
}en_led_pin_t_;

/* LED Ports */
typedef enum
{
    LED_PORT_A    =     0     ,
    LED_PORT_B,
    LED_PORT_C,
    LED_PORT_D,
    LED_PORT_E,
    LED_PORT_F,
    LED_PORT_TOTAL
}en_led_port_t_;

typedef enum
{
    LED_OK              = 0 ,
    LED_ERROR               ,
}en_led_error_t_;

| Initializes a single LED pin as output
|
| Parameters
|            [in] en_a_ledPort The port where the LED is located
|                              (PORT_A, PORT_B, PORT_C or PORT_D)
|            [in] u8_a_ledPin The pin number of the LED
|                             (DIO_U8_PIN_0 to DIO_U8_PIN_7)
| Return
|     EN_LED_ERROR_t Returns LED_OK if the LED was initialized
|                    successfully, LED_ERROR otherwise.
|
EN_LED_ERROR_t LED_init(EN_DIO_PORT_T en_a_ledPort, u8 u8_a_ledPin);
```

```
| Turn on an LED connected to a specific pin on a specific port.
|
| Parameters
|           [in] en_a_ledPort The port where the LED is connected.
|                            (PORT_A, PORT_B, PORT_C, or PORT_D)
|           [in] u8_a_ledPin The pin number where the LED is connected.
|                           (DIO_U8_PIN_0 to DIO_U8_PIN_7)
| Return
|     The status of the LED operation, either LED_OK or LED_ERROR.
|
EN_LED_ERROR_t LED_on(EN_DIO_PORT_T en_a_ledPort, u8 u8_a_ledPin);
```

```
| Turns off an LED on a specific port and pin.
| Parameters
|     [in]    en_a_ledPort    The port of the LED to turn off
|                               (PORT_A, PORT_B, PORT_C, or PORT_D)
|     [in]    u8_a_ledPin       The pin number of the LED to turn off
|                               (DIO_U8_PIN_0 to DIO_U8_PIN_7)
| Returns
|     EN_LED_ERROR_t LED_OK if successful,
|                    or LED_ERROR if there was an error.
|
EN_LED_ERROR_t LED_off(EN_DIO_PORT_T en_a_ledPort, u8 u8_a_ledPin);
```

2.3.3.2. BTN APIs

```
|
| 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);
```

2.3.4. APP APIs

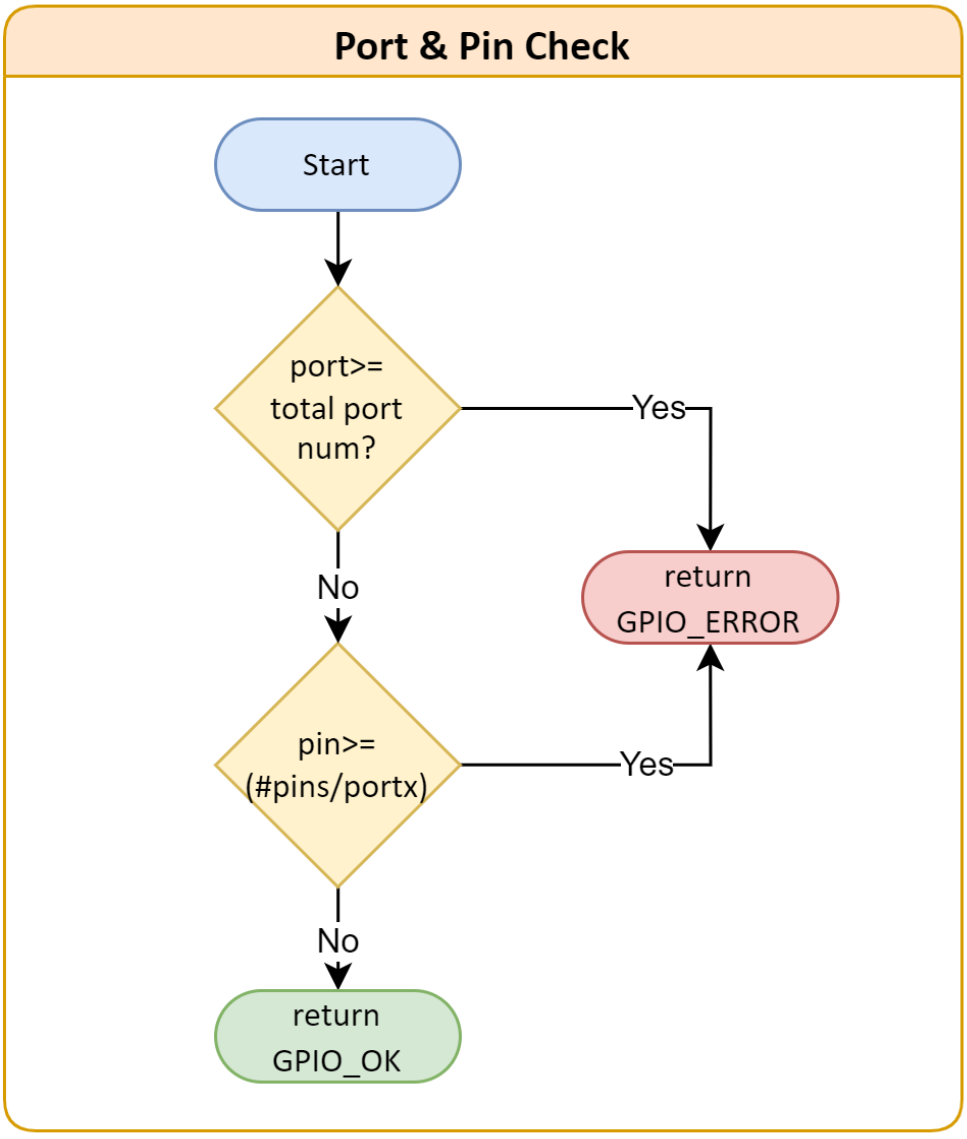
```
| Initializes the required modules by the app
|
| Return
|        APP_OK              :    In case of Successful Operation
|        APP_FAIL            :    In case of Failed Operation
|
en_app_error_t app_init(void);

| This function starts the app program and keeps it running indefinitely.
void app_start(void);
```

3. Low Level Design

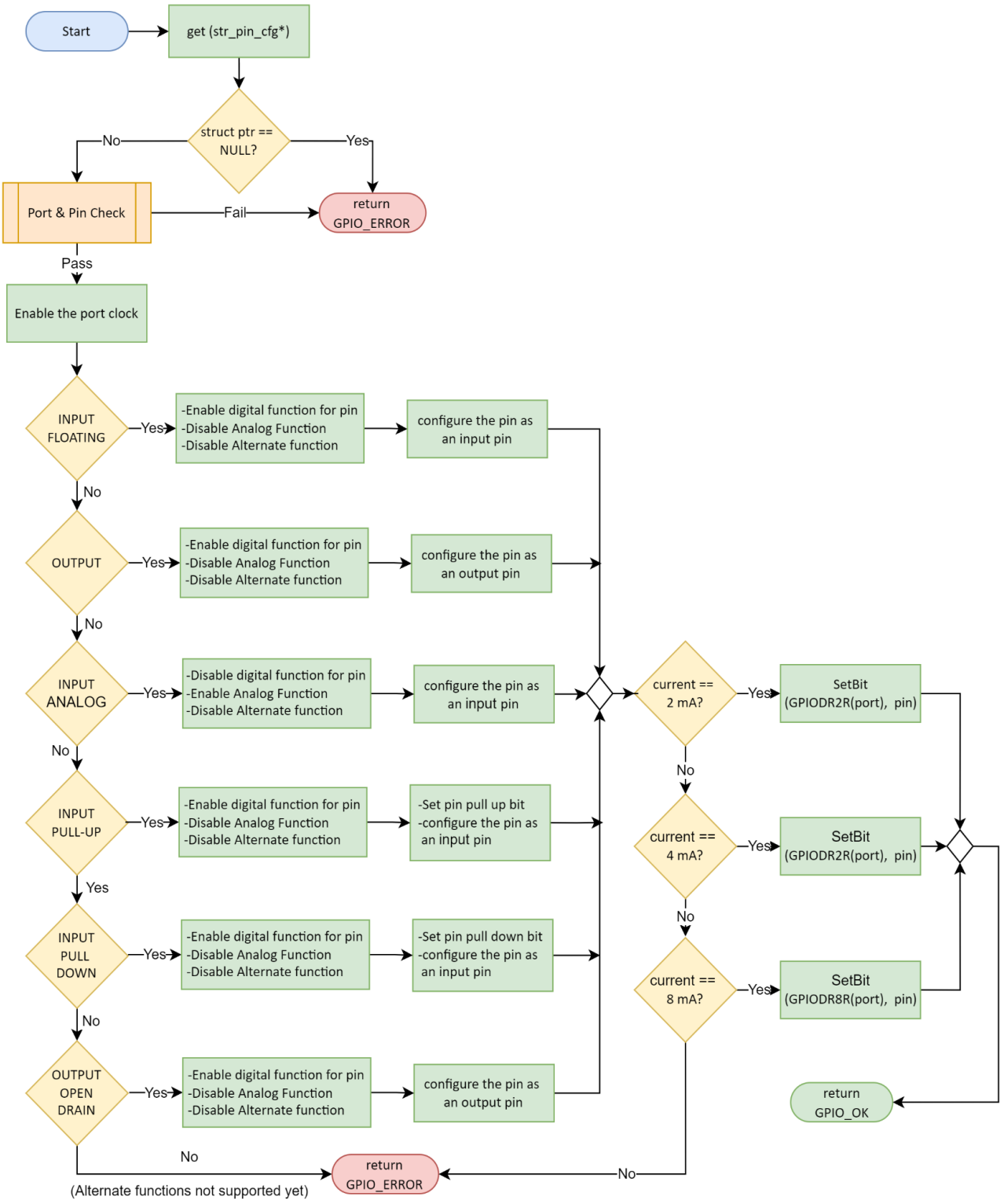
3.1. MCAL Layer

3.1.1. GPIO Module

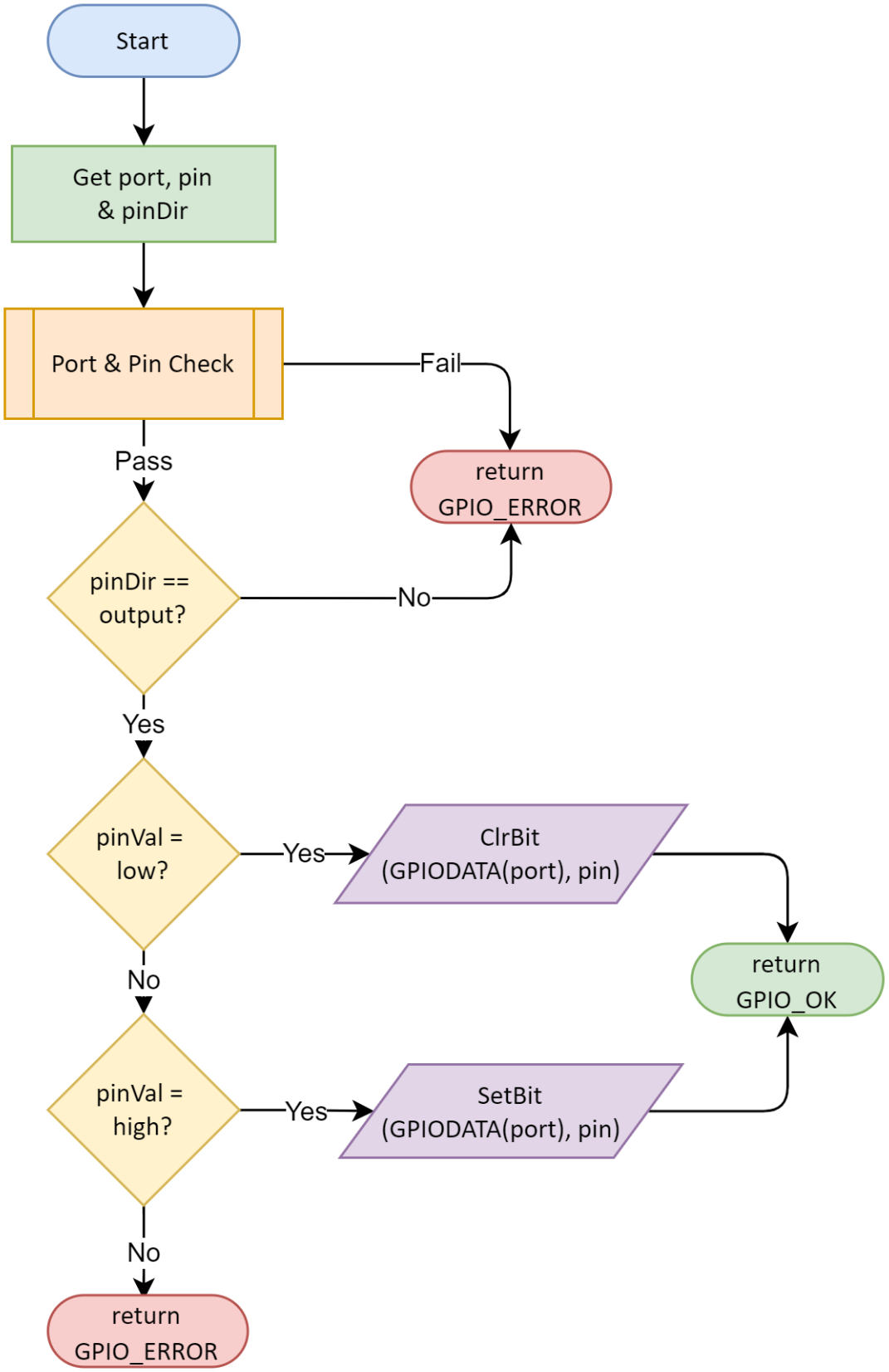
3.1.1.a. sub process



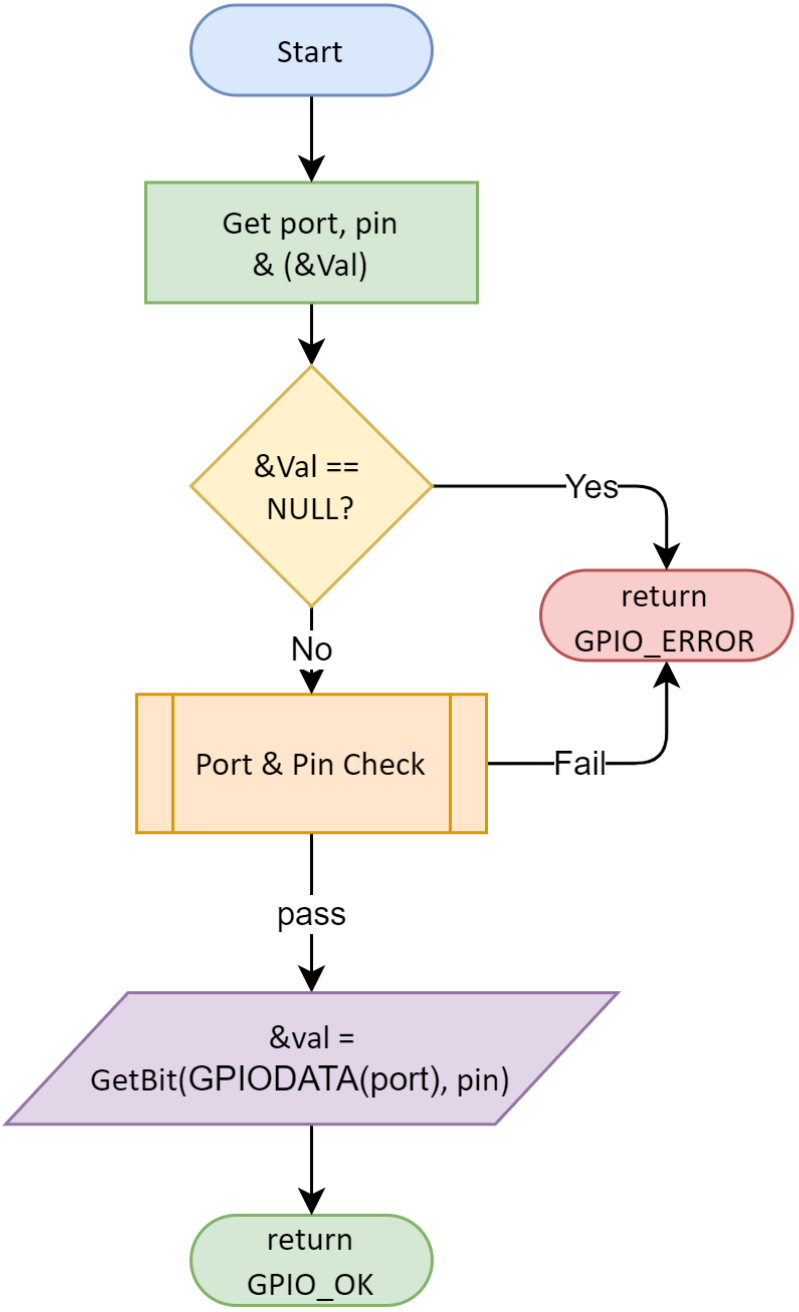
3.1.1.1. GPIO_pin_init



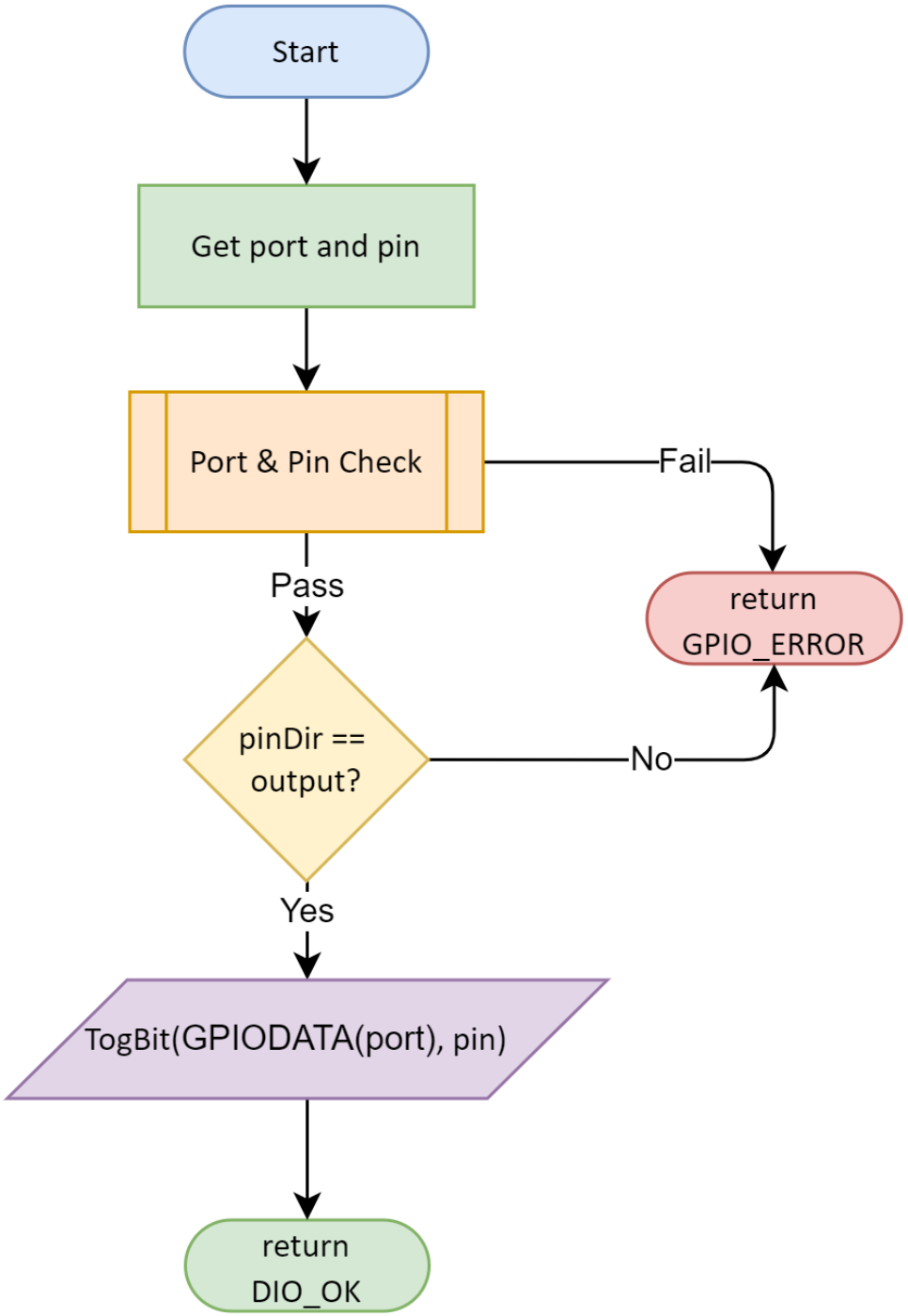
3.1.1.2. GPIO_setPinVal



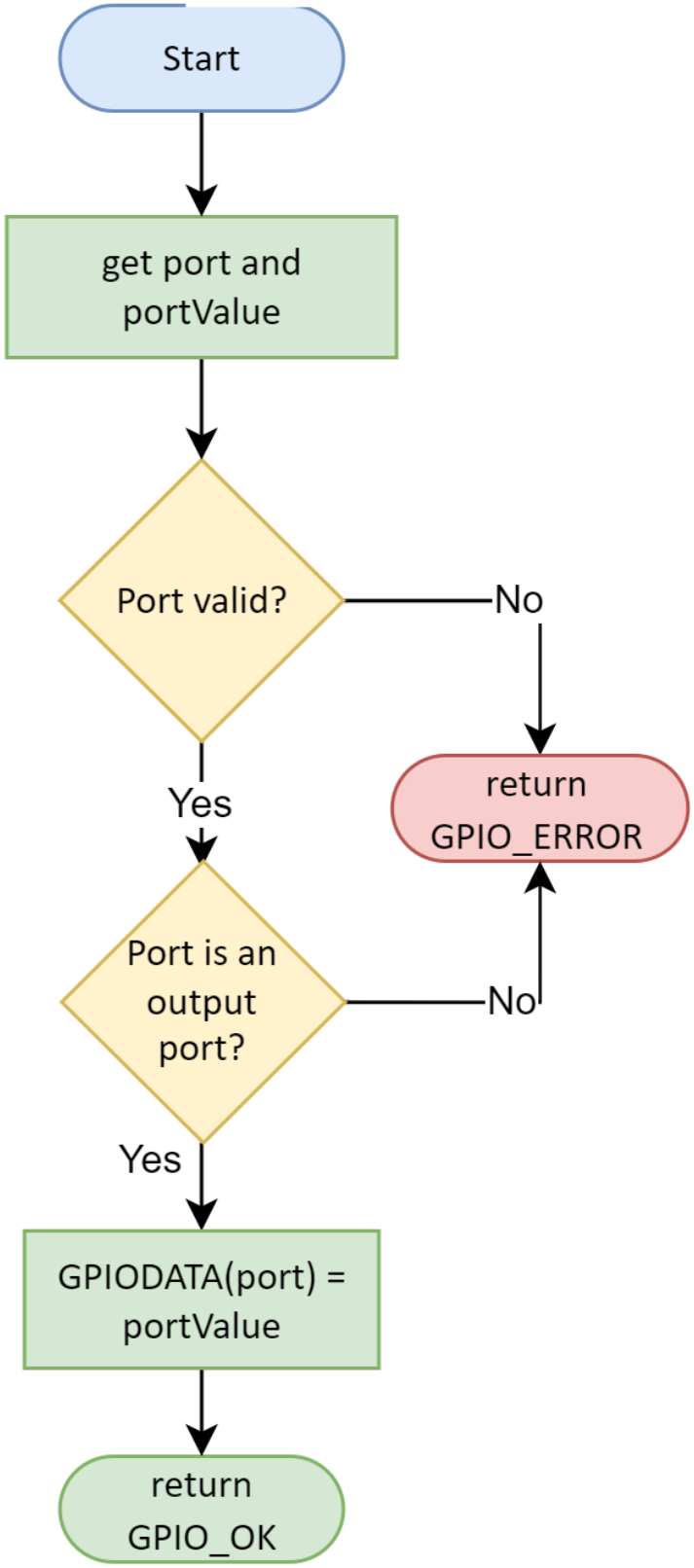
3.1.1.3. DIO_getPinVal



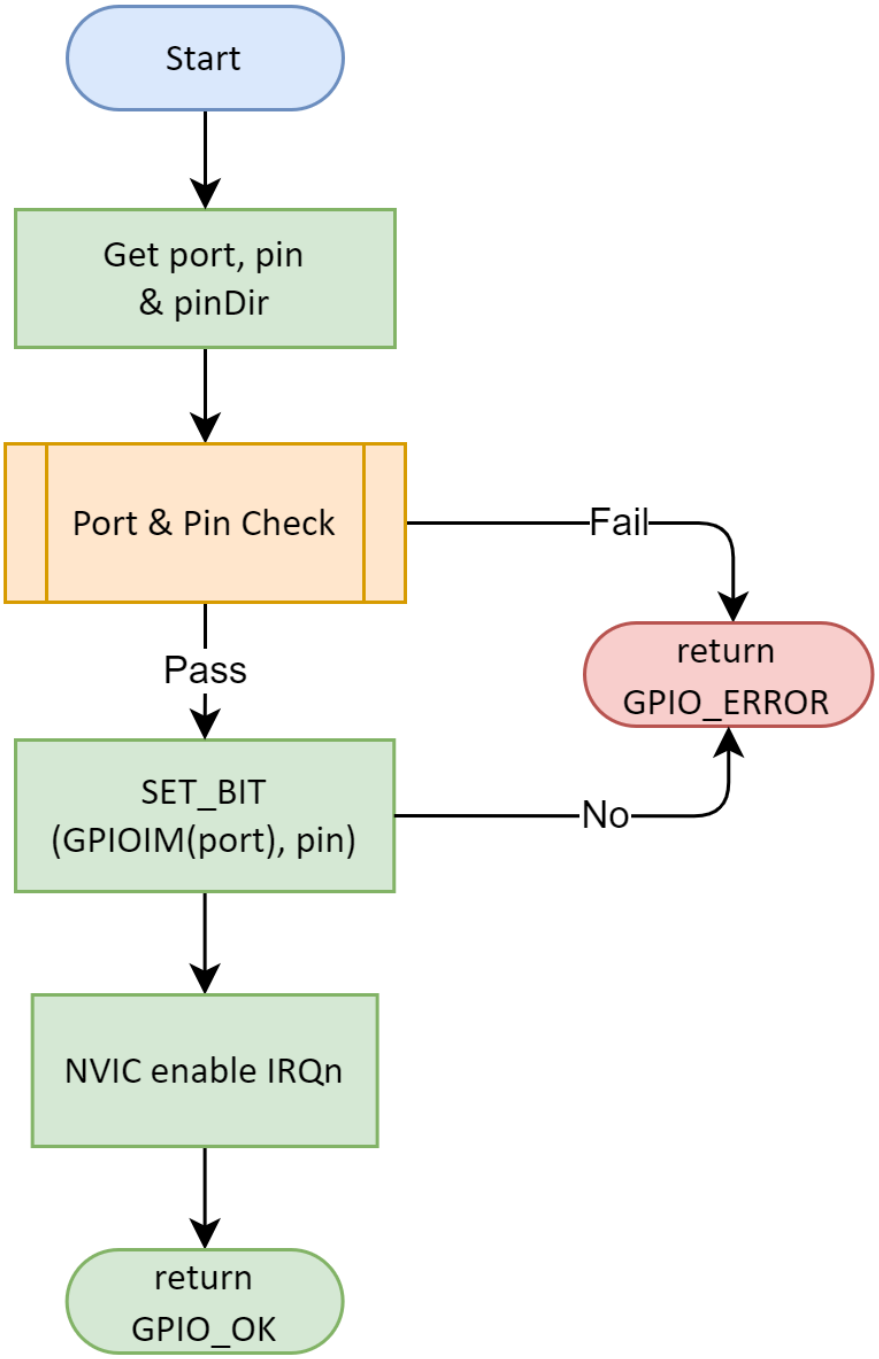
3.1.1.4. DIO_togPinVal



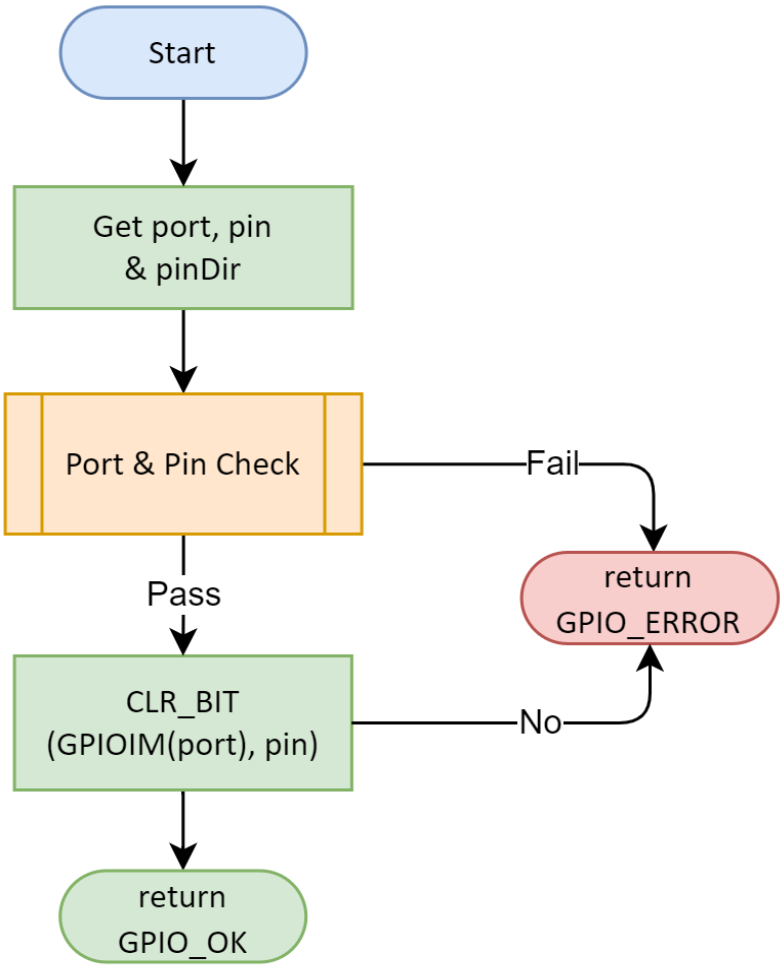
3.1.1.5. GPIO_setPortVal



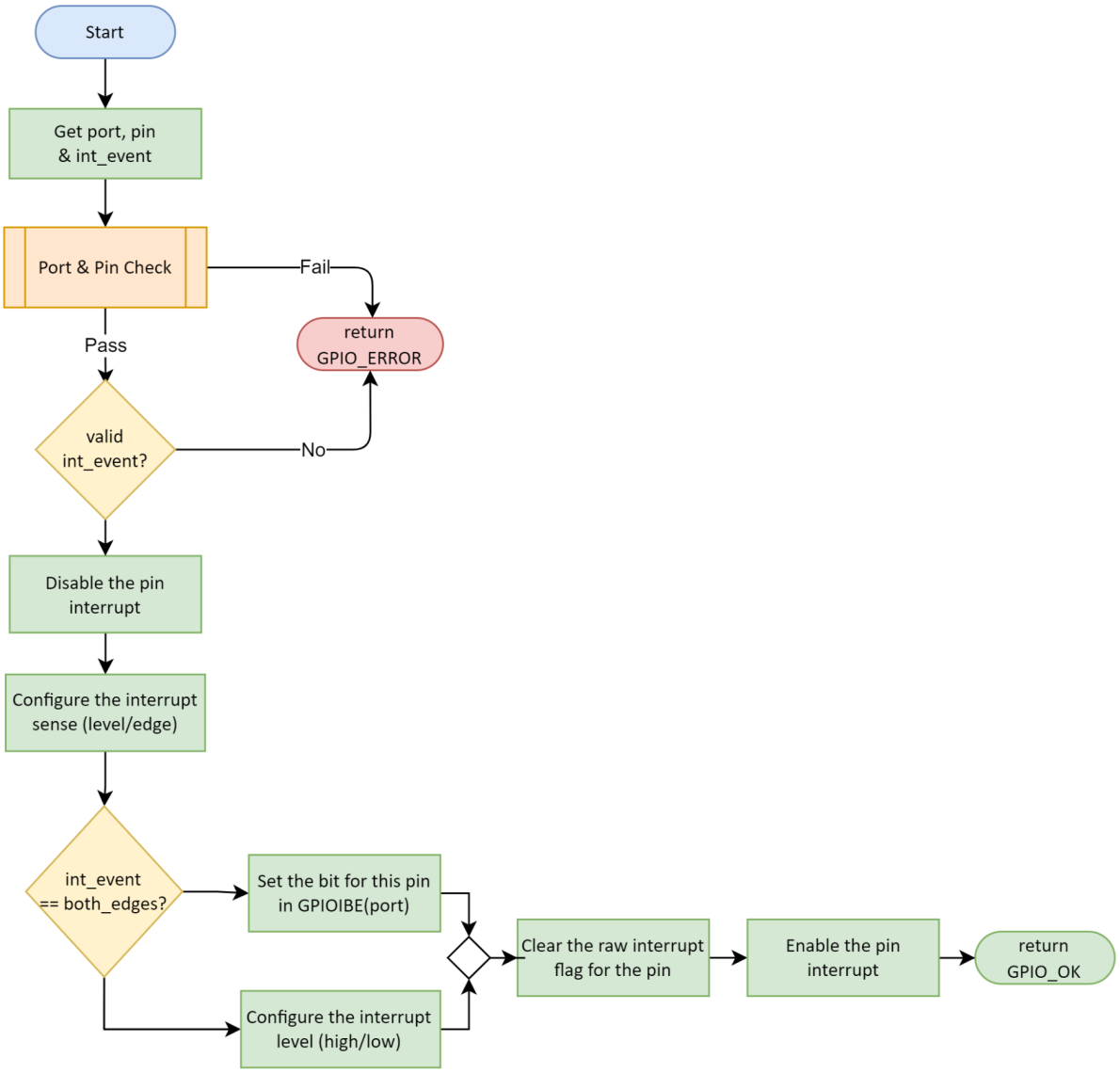
3.1.1.6. GPIO_enableInt



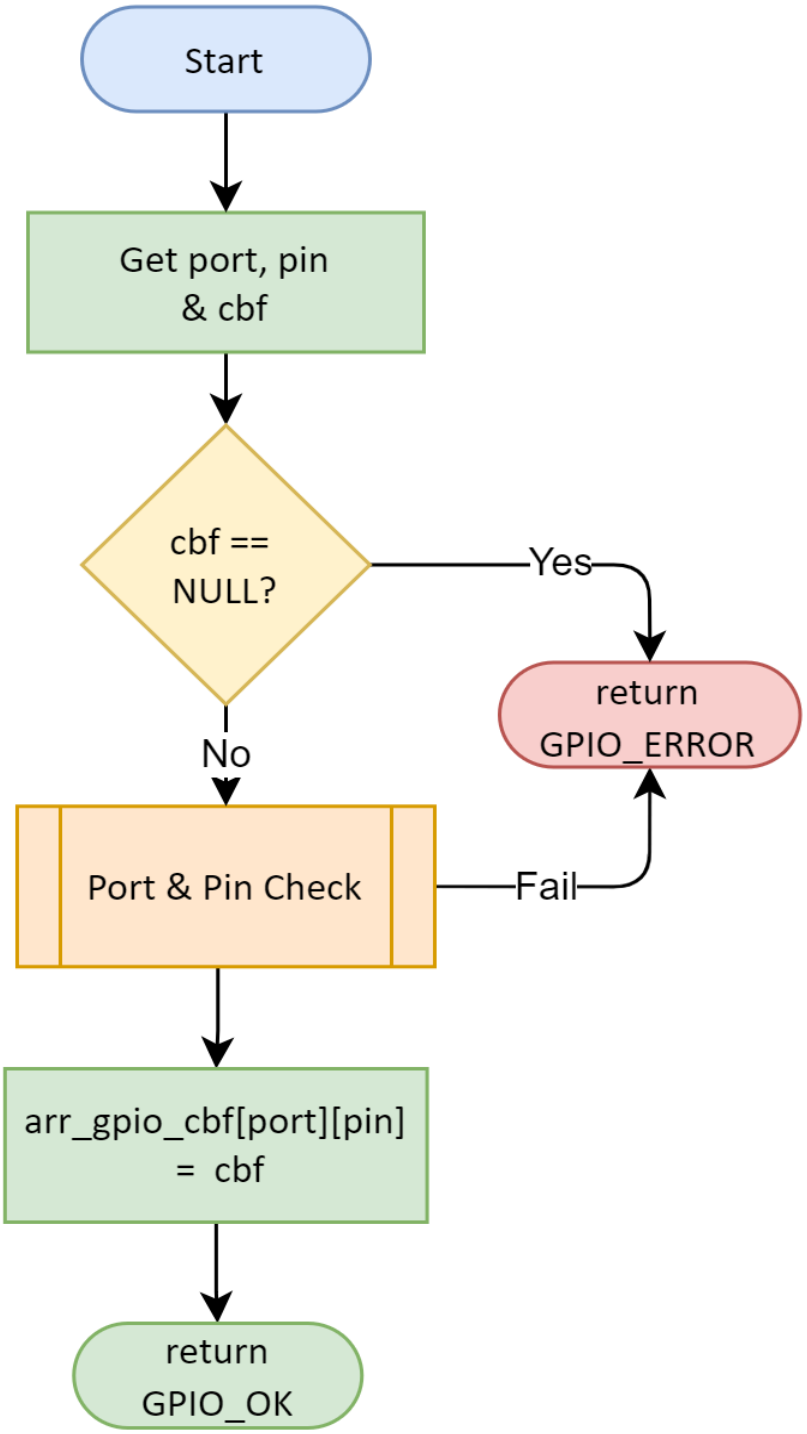
3.1.1.7. GPIO_disableInt



3.1.1.8. GPIO_setIntSense

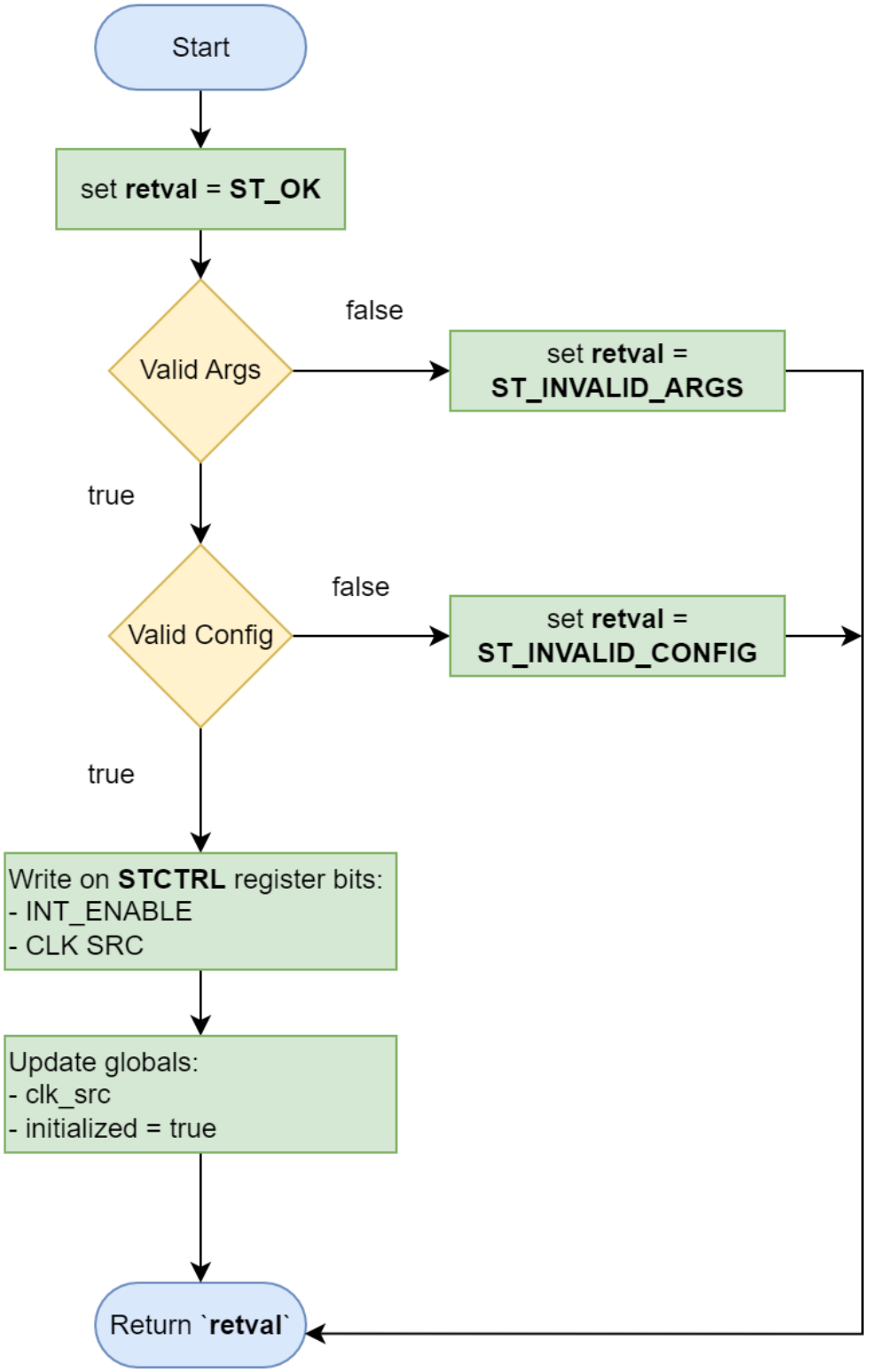


3.1.1.9. GPIO_setIntCallback

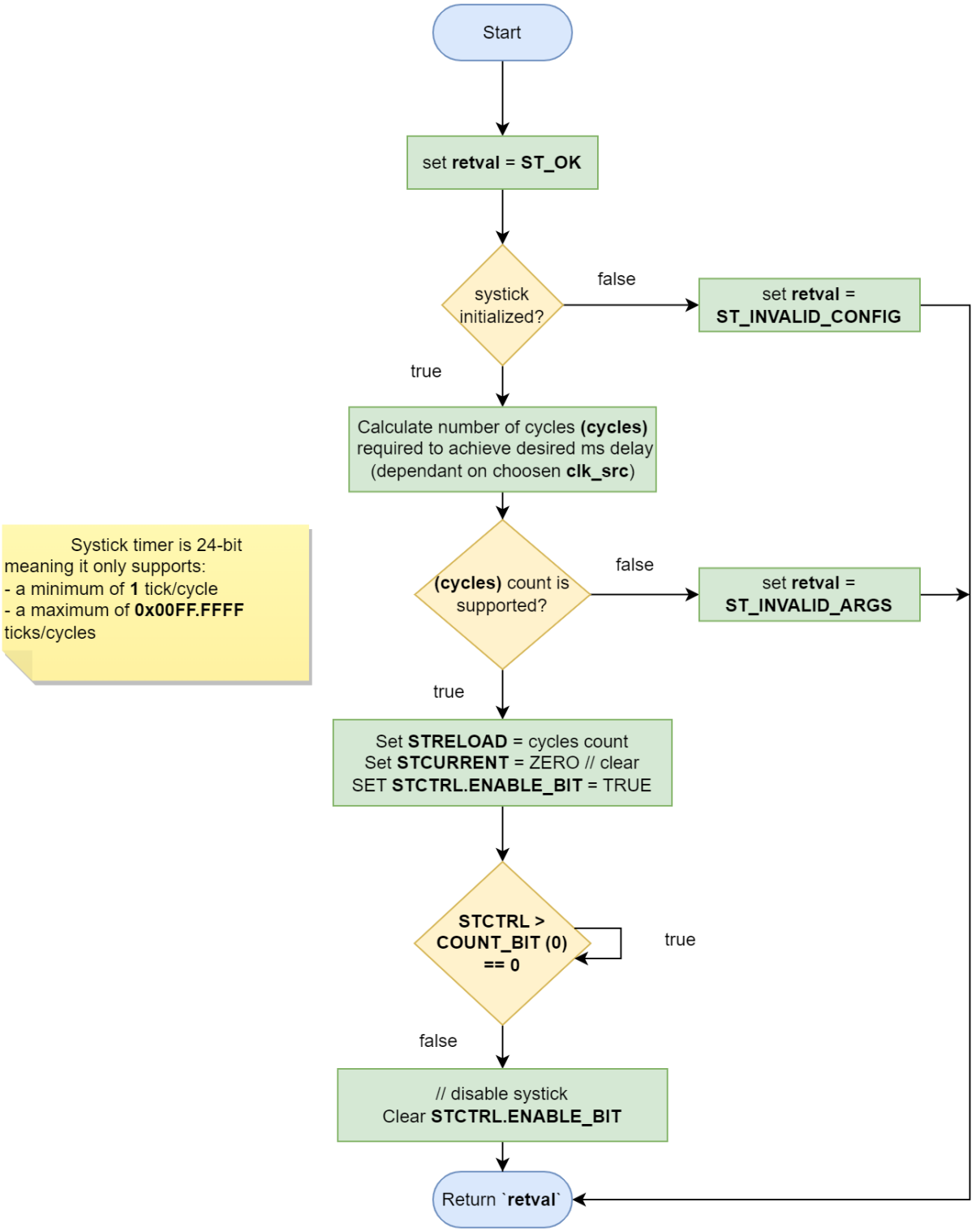


3.1.2. SYSTICK Module

3.1.2.1. systick_init



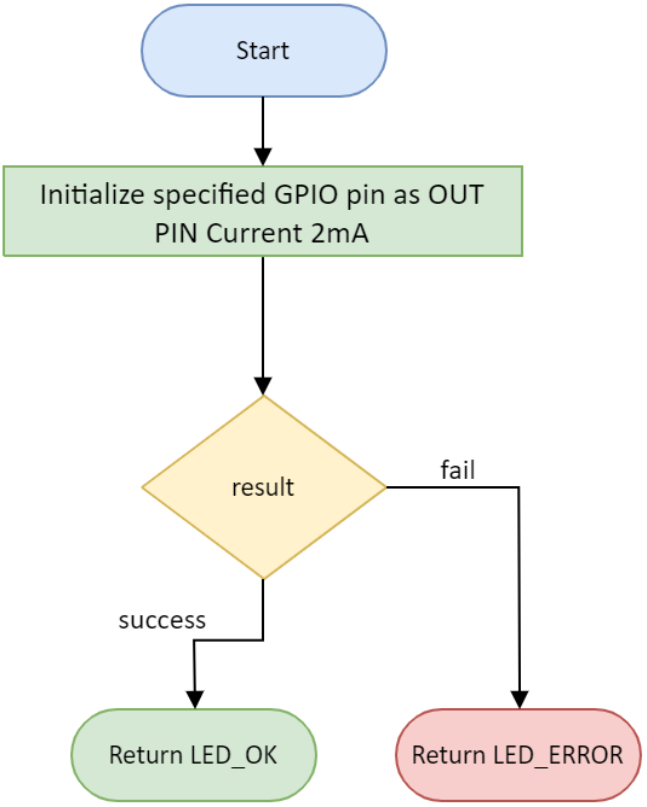
3.1.2.2. systick_ms_delay



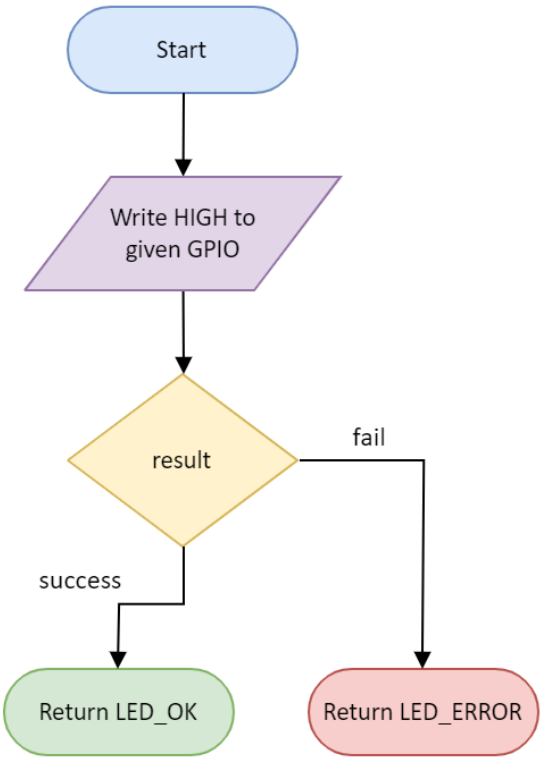
3.2. HAL Layer

3.2.1. LED Module

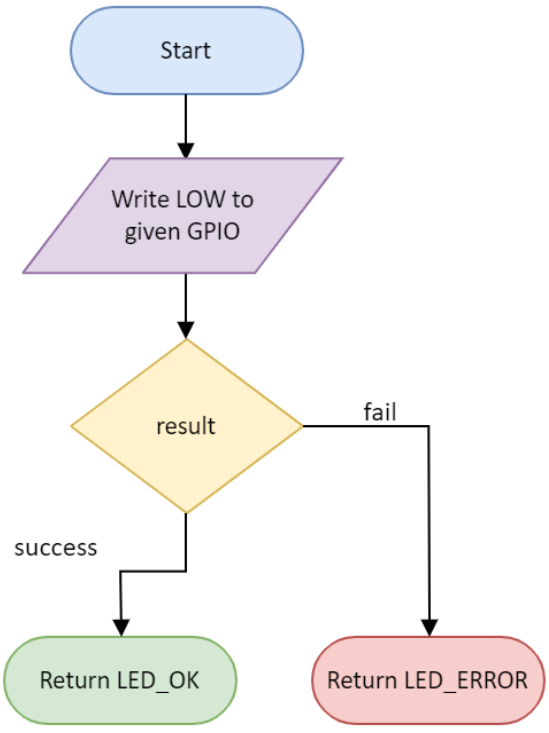
3.2.1.1. LED_init



3.2.1.2. LED_on

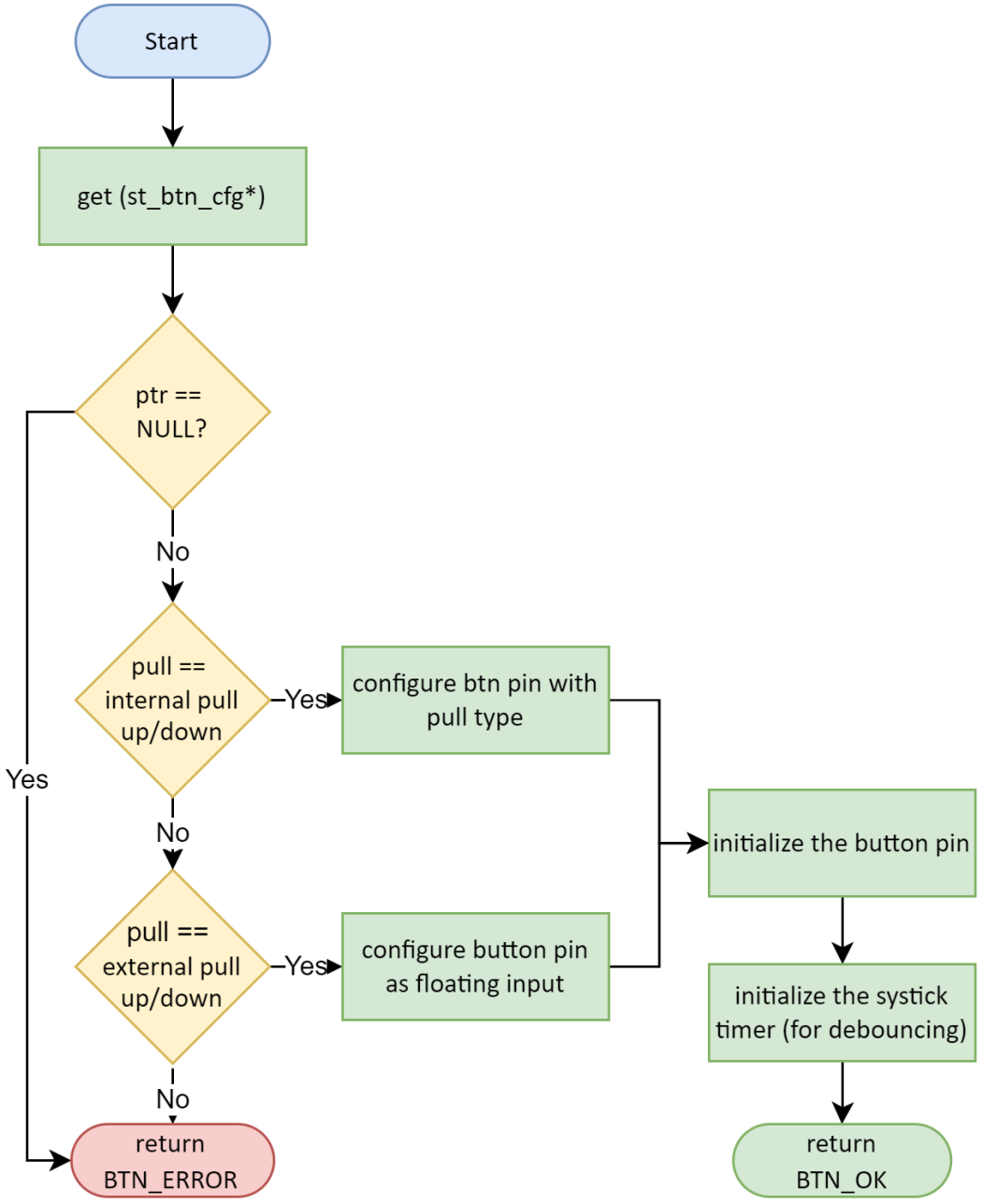


3.2.1.3. LED_off

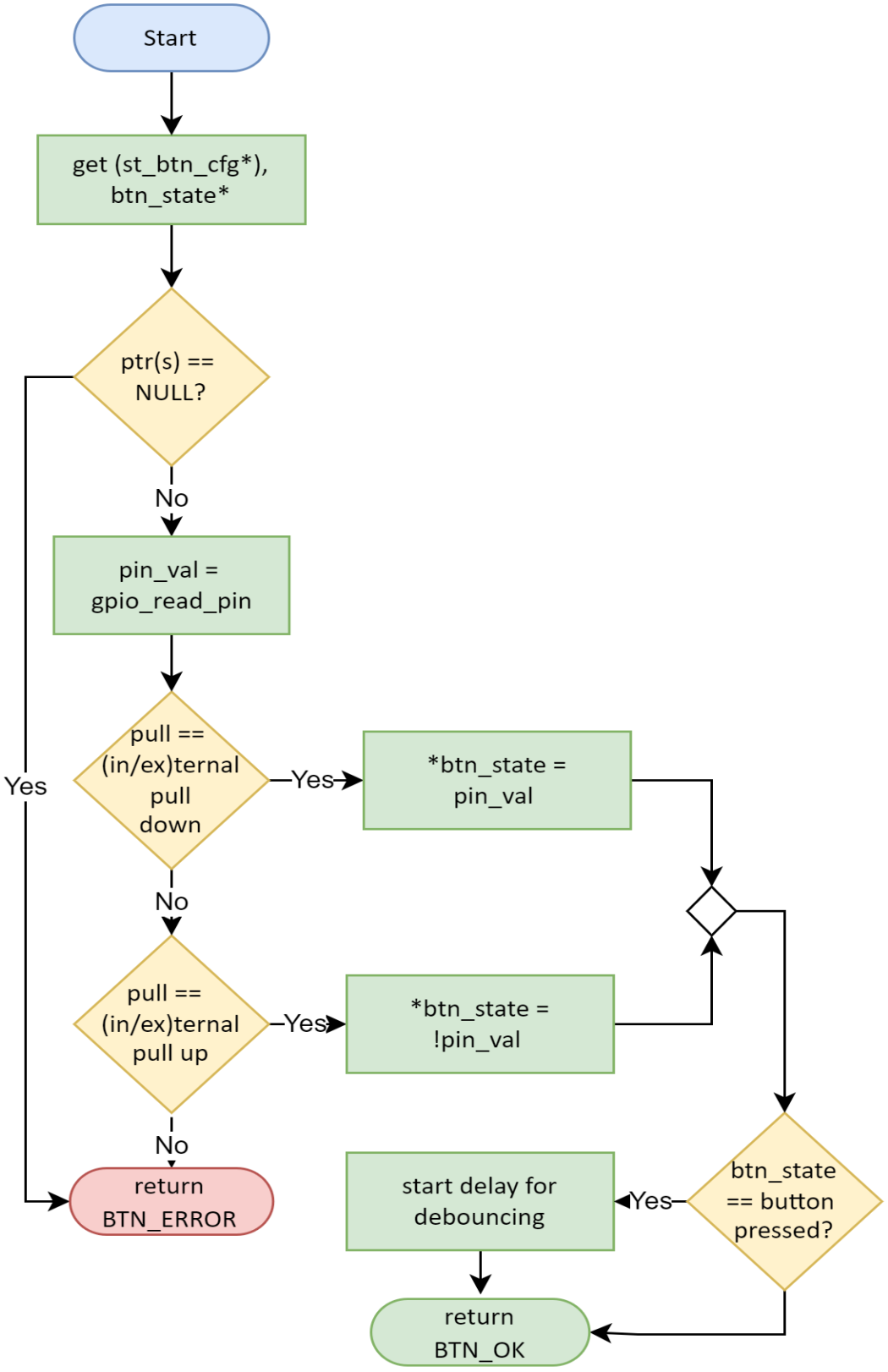


3.2.2. BTN Module

3.2.2.1. BUTTON_init

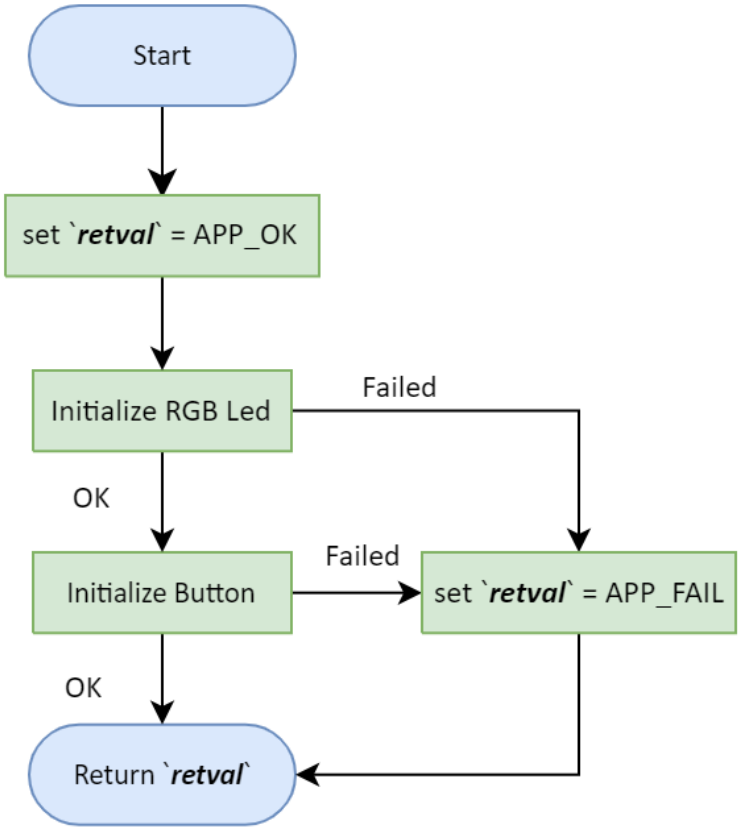


3.2.2.2. BUTTON_read

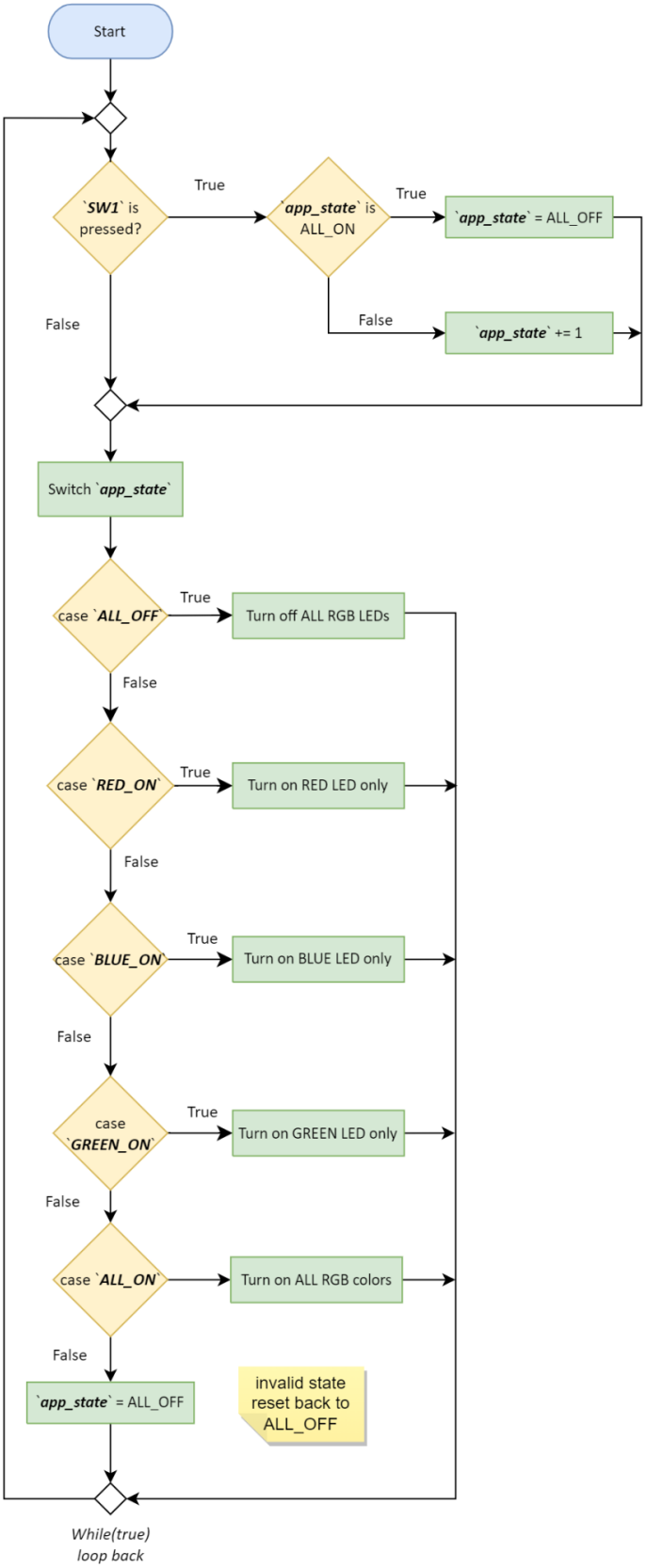


3.3. APP Layer

3.3.1. app_init



3.3.2. app_start



4. Pre-compiling and linking configurations

4.1. GPIO Driver

None

4.2. SYSTICK Driver

4.2.1. Pre-compiled Configurations

```
#define SYS_CLOCK_MHZ   8

#if SYS_CLOCK_MHZ < 8
    #warning System clock below 8 MHZ is not supported by systick
#endif
#define PIOSC_MHZ       16
```

4.3. LED Driver

None.

4.4. BTN Driver

None.

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