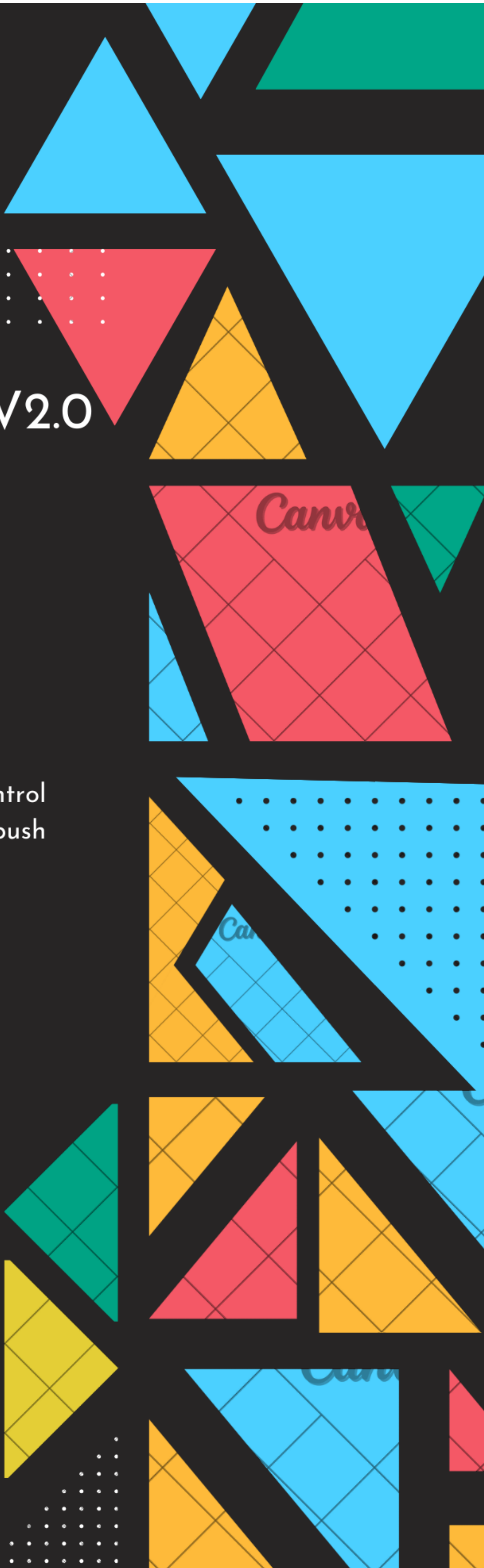
RGB LED CONTROL V2.0

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 V2.0

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 **for 1 sec only**
- After the second press, the **Green led** is on **for 1 sec only**
- After the third press, the **Blue led** is on **for 1 sec only**
- After the fourth press, **all LEDs** are on **for 1 sec only**
- 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
- Implement Systick 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

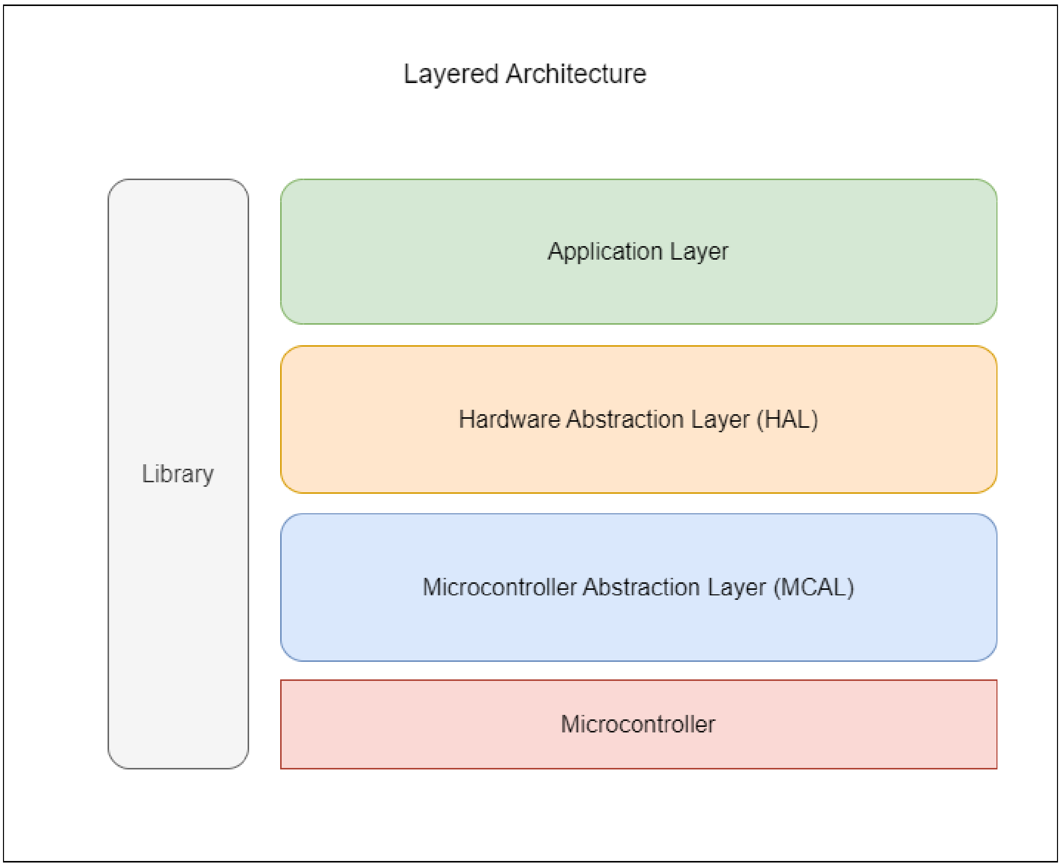
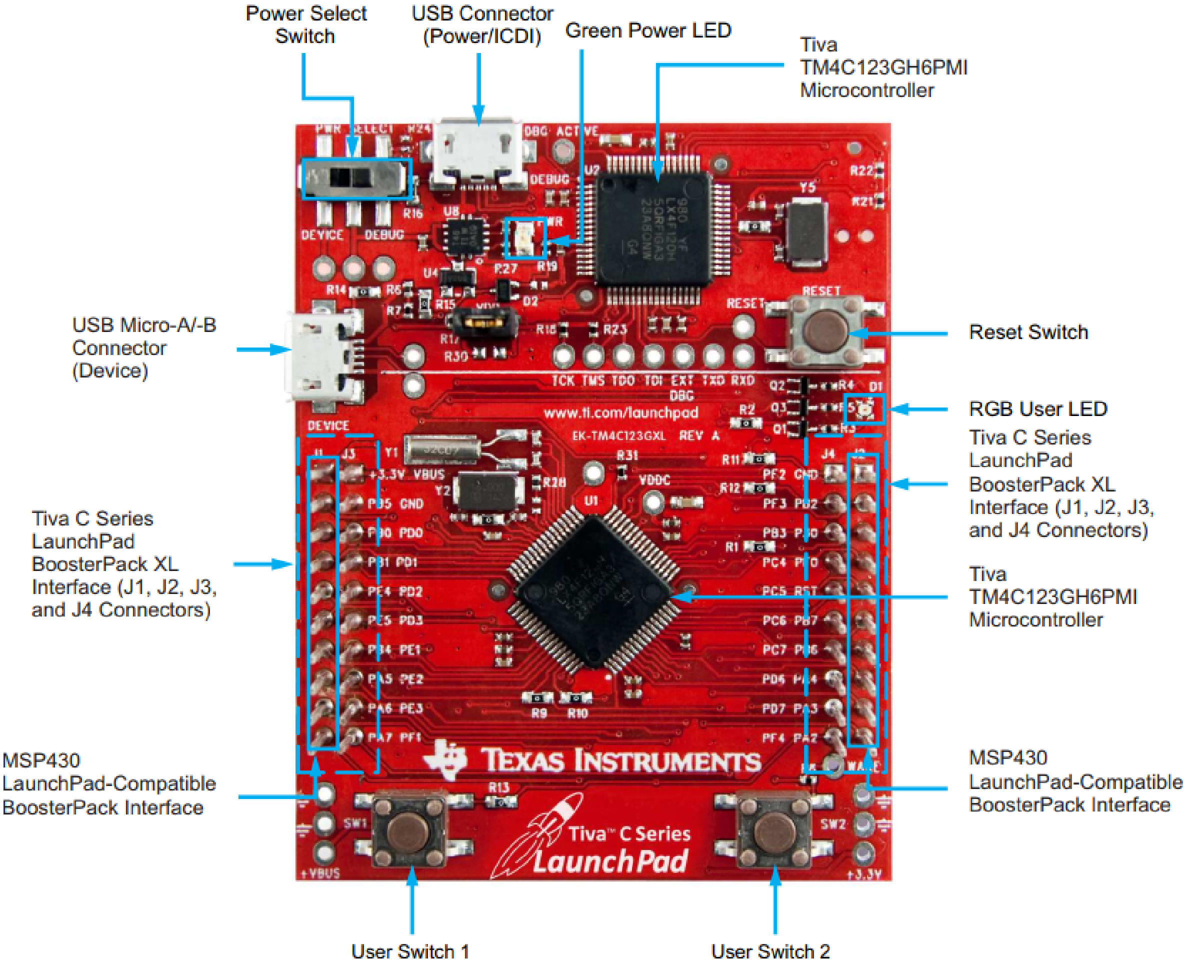

Layered Architecture
Library
Application Layer
Hardware Abstraction Layer (HAL)
Microcontroller Abstraction Layer (MCAL)
Microcontroller

Figure 1. Layered Architecture Design

2.1.3. Tiva C Board Schematic

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


Power Select Switch
USB Connector (Power/ICDI)
Green Power LED
Tiva TM4C123GH6PMI Microcontroller
USB Micro-A/-B Connector (Device)
Reset Switch
RGB User LED
Tiva C Series LaunchPad BoosterPack XL Interface (J1, J2, J3, and J4 Connectors)
Tiva C Series LaunchPad BoosterPack XL Interface (J1, J2, J3, and J4 Connectors)
Tiva TM4C123GH6PMI Microcontroller
MSP430 LaunchPad-Compatible BoosterPack Interface
MSP430 LaunchPad-Compatible BoosterPack Interface
User Switch 1
User Switch 2

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

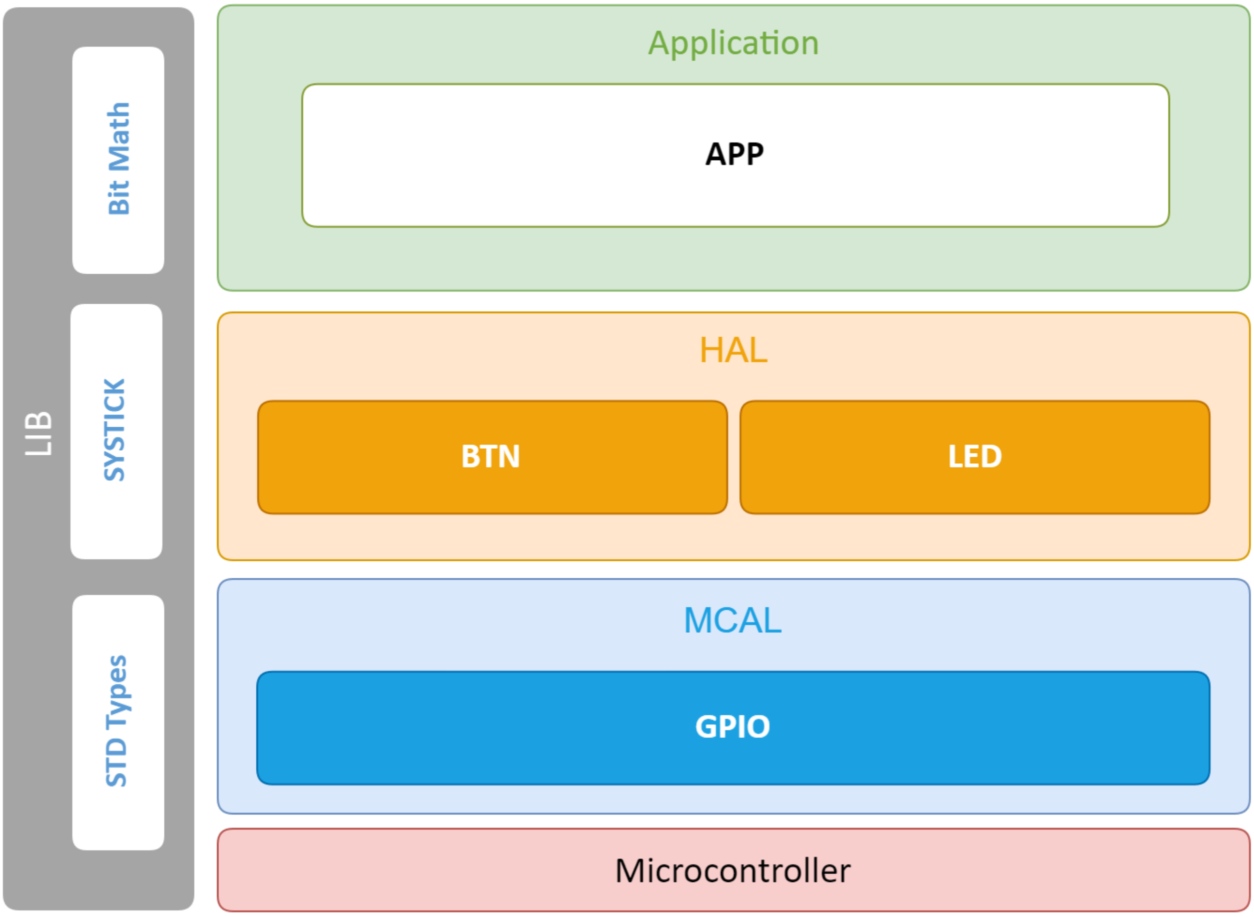


Figure 3. System Modules Design

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

```
/*----------------------------------------------------------/
/- 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. SYSTICK Driver

```
| Initiates a sync blocking delay
|
| Parameters
|           uint32_ms_delay    Desired delay in ms
| @note  Will be canceled if any sync/blocking delay was requested
|
| 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_async_ms_delay(uint32_t_ uint32_ms_delay);

| Sets callback function to be called when an async delay is finished
|
| Parameters
|           fun_ptr_systick_cb Pointer to callback fn
|
| 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_set_callback(fun_systick_callback_t
fun_ptr_systick_cb);
```

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);
```

2.3.4. APP APIs

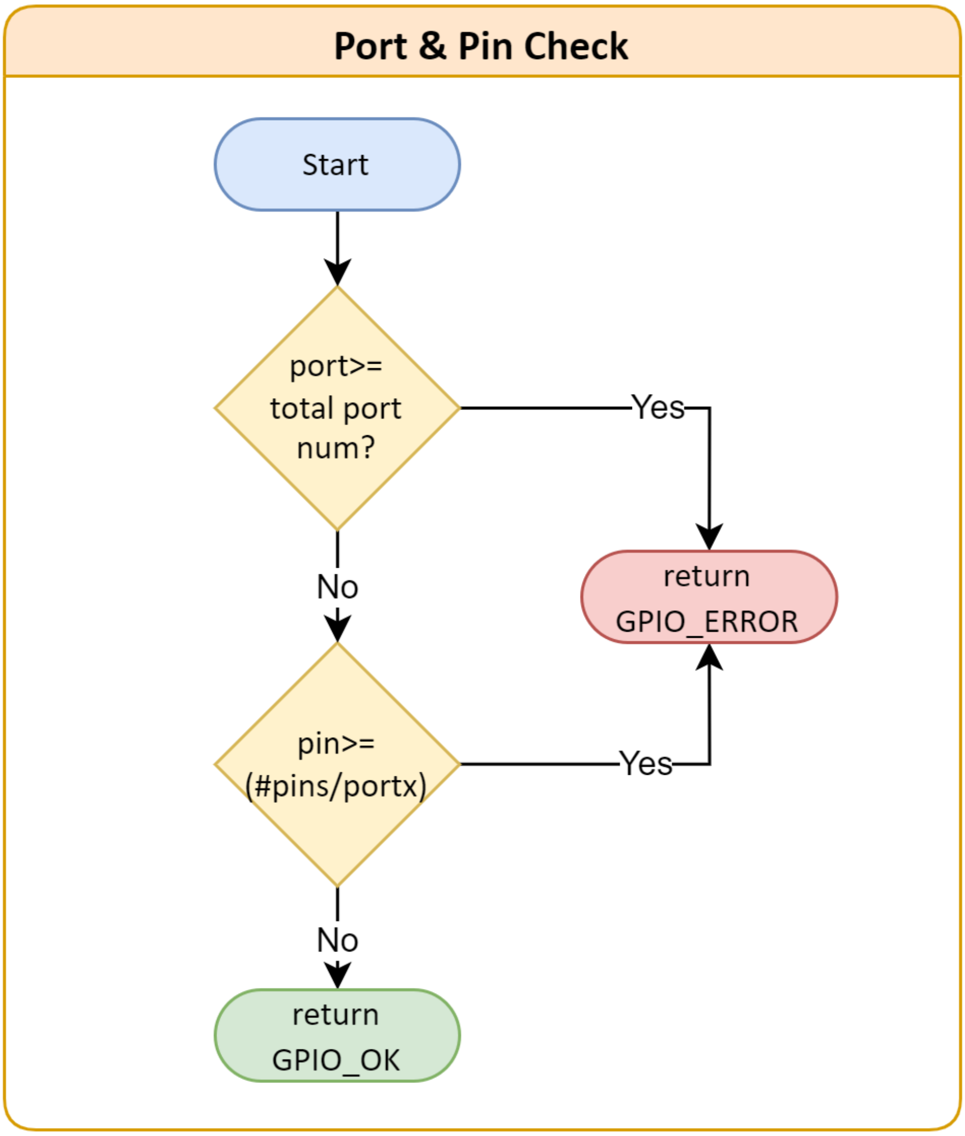
```
| Initializes the required modules by the app
|
| Return
|       APP_OK              :   In case of Successful Operation
|       APP_FAIL            :   In case of Failed Operation
|
en_app_error_t app_init(void);

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

| private function to switch app state
static void app_switch_state(void);

| private function to systick callback
static void app_systick_cb(void);
```

3. Low Level Design

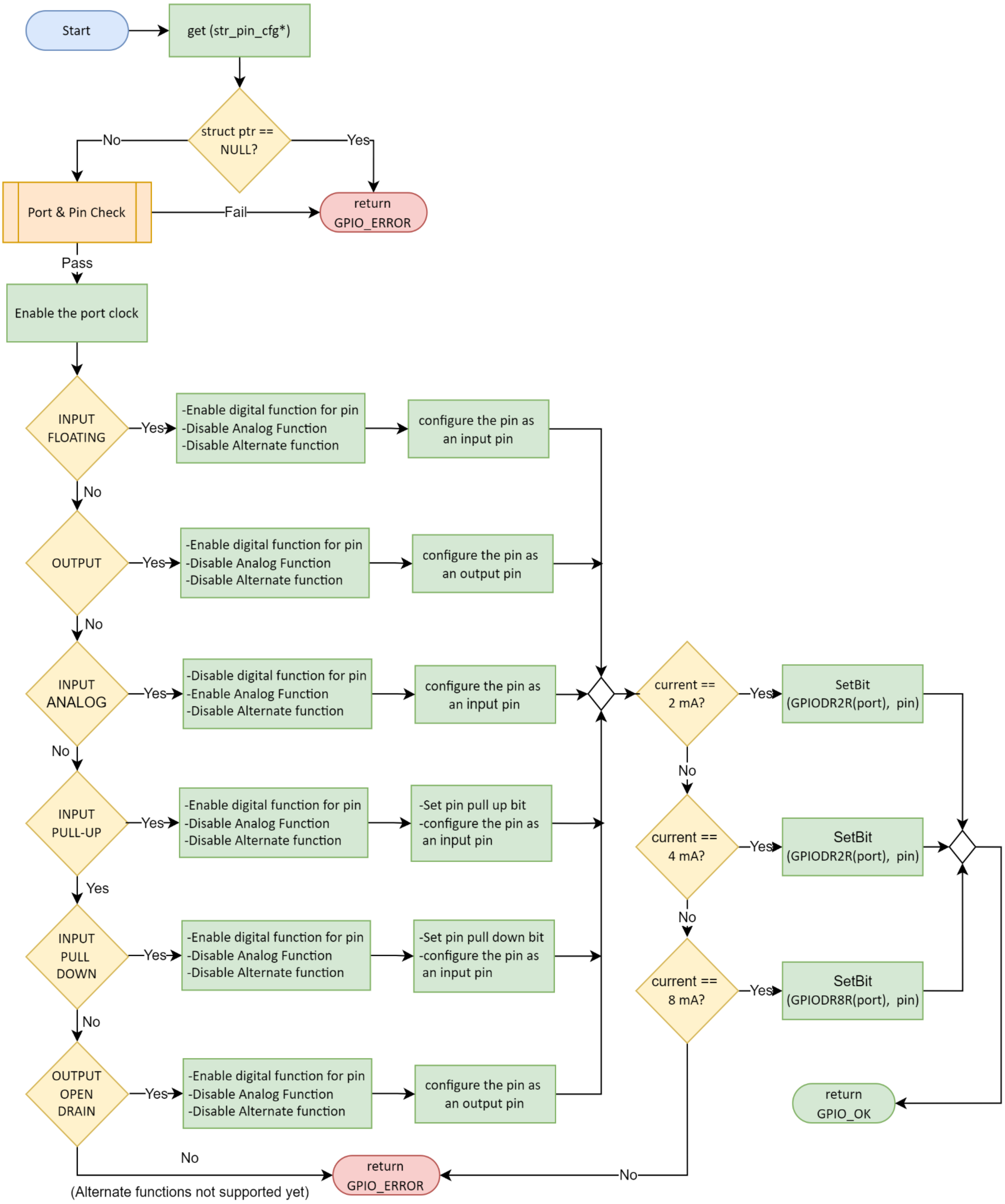
3.1. MCAL Layer

3.1.1. GPIO Module

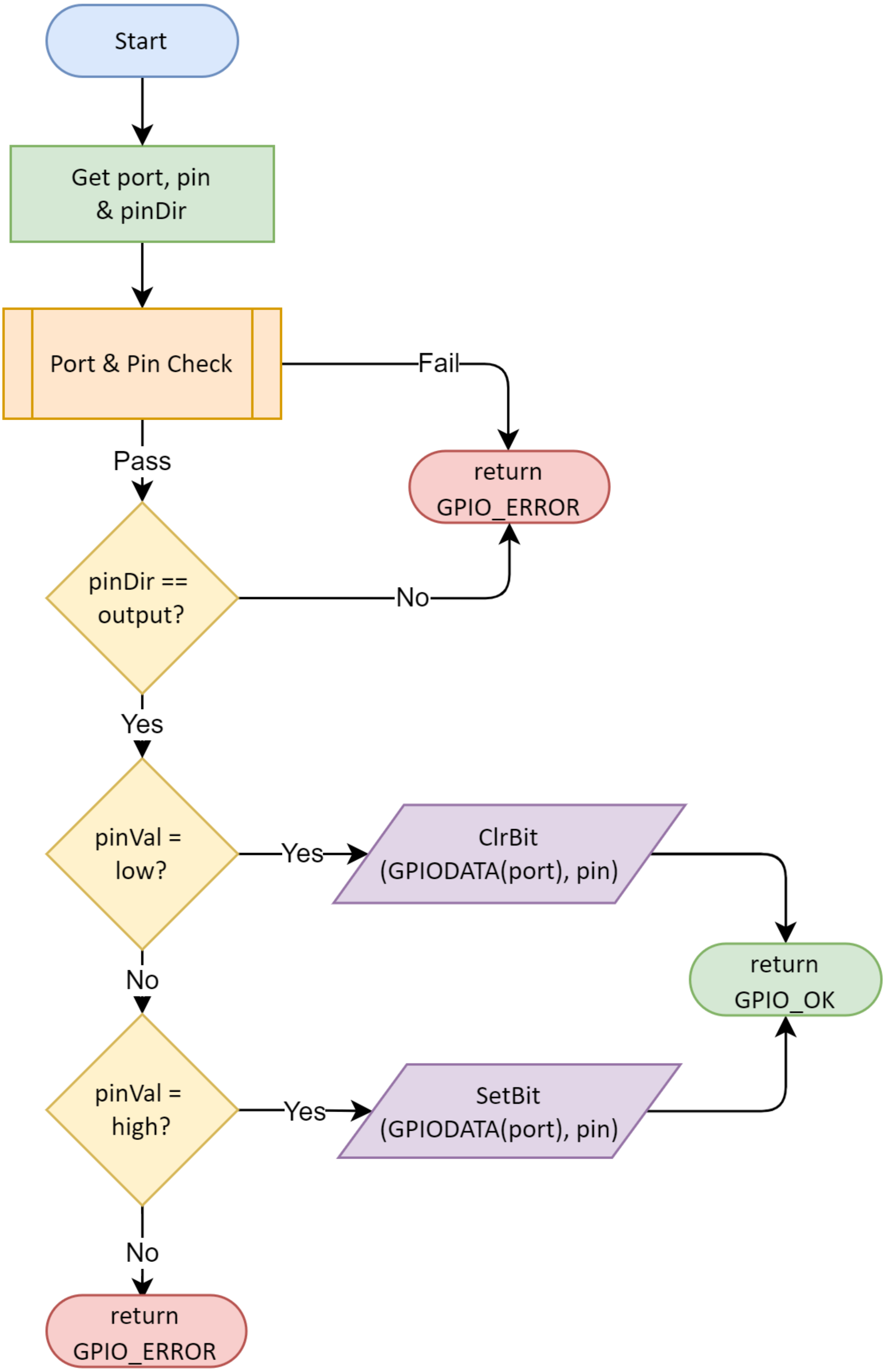
3.1.1.a. sub process



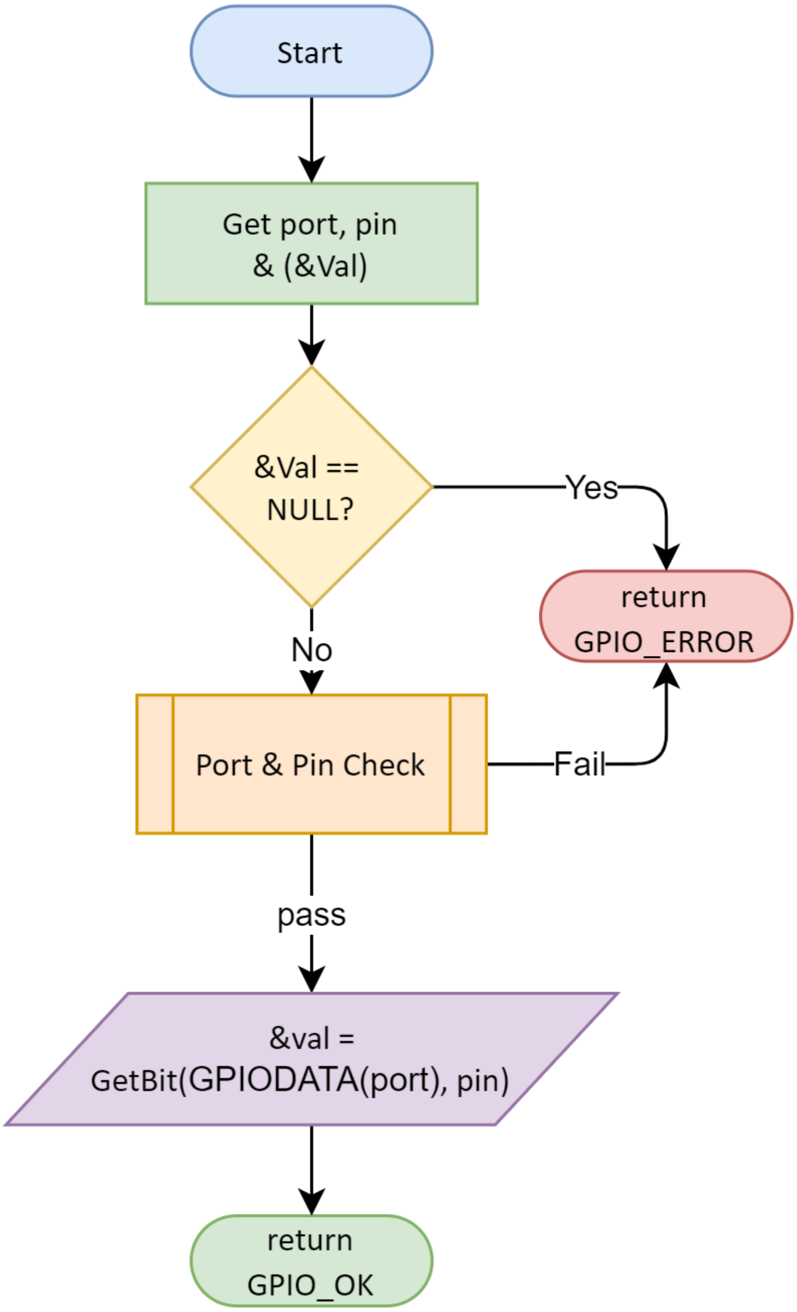
3.1.1.1. GPIO_pin_init


Start
get (str_pin_cfg*)
struct ptr == NULL?
No
Yes
Port & Pin Check
Fail
return GPIO_ERROR
Pass
Enable the port clock
INPUT FLOATING
Yes
-Enable digital function for pin
-Disable Analog Function
-Disable Alternate function
configure the pin as an input pin
No
OUTPUT
Yes
-Enable digital function for pin
-Disable Analog Function
-Disable Alternate function
configure the pin as an output pin
No
INPUT ANALOG
Yes
-Disable digital function for pin
-Enable Analog Function
-Disable Alternate function
configure the pin as an input pin
No
INPUT PULL-UP
Yes
-Enable digital function for pin
-Disable Analog Function
-Disable Alternate function
-Set pin pull up bit
-configure the pin as an input pin
Yes
INPUT PULL DOWN
Yes
-Enable digital function for pin
-Disable Analog Function
-Disable Alternate function
-Set pin pull down bit
-configure the pin as an input pin
No
OUTPUT OPEN DRAIN
Yes
-Enable digital function for pin
-Disable Analog Function
-Disable Alternate function
configure the pin as an output pin
No
(Alternate functions not supported yet)
return GPIO_ERROR
No
current == 2 mA?
Yes
SetBit (GPIODR2R(port), pin)
No
current == 4 mA?
Yes
SetBit (GPIODR2R(port), pin)
No
current == 8 mA?
Yes
SetBit (GPIODR8R(port), pin)
return GPIO_OK

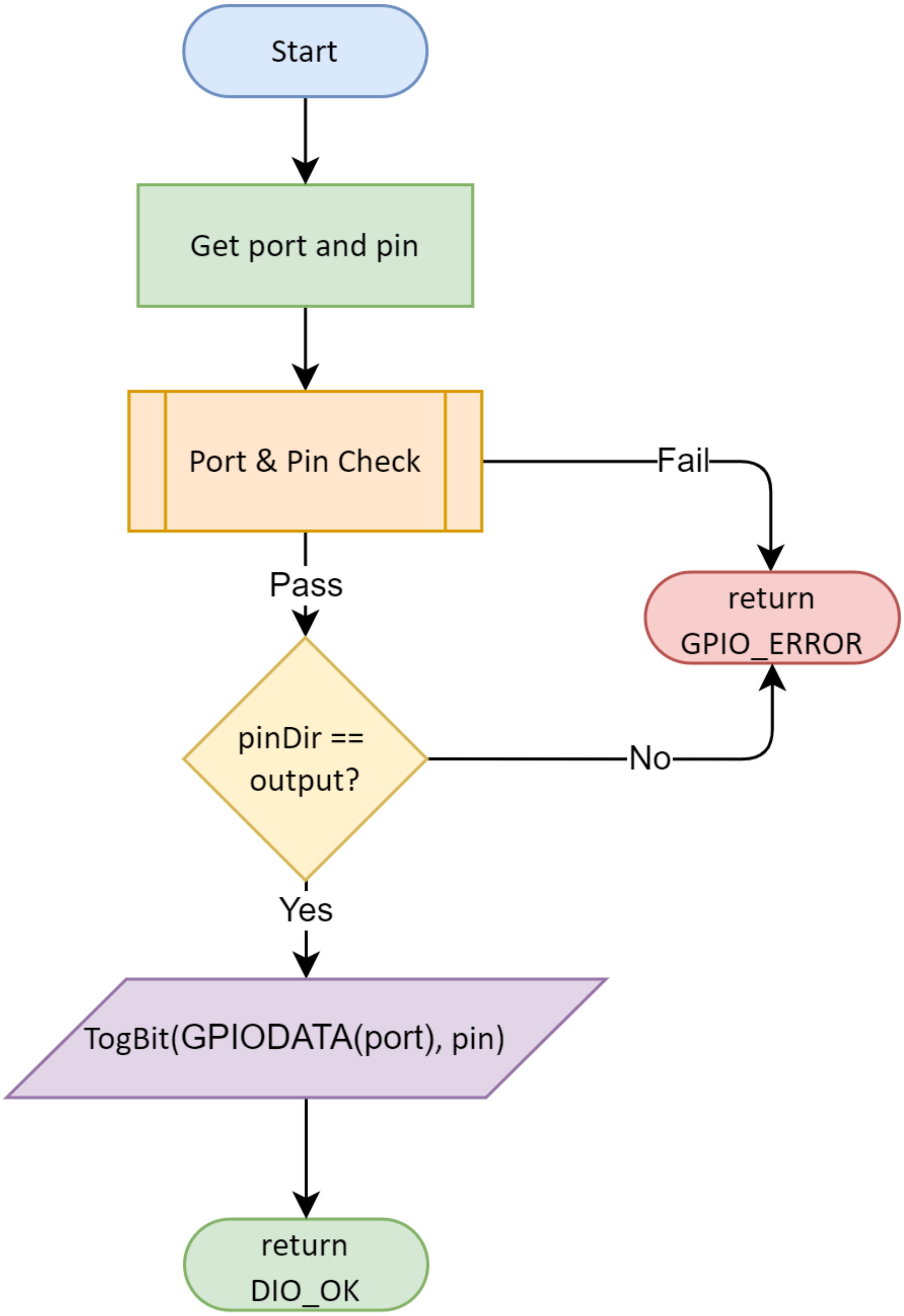
3.1.1.2. GPIO_setPinVal



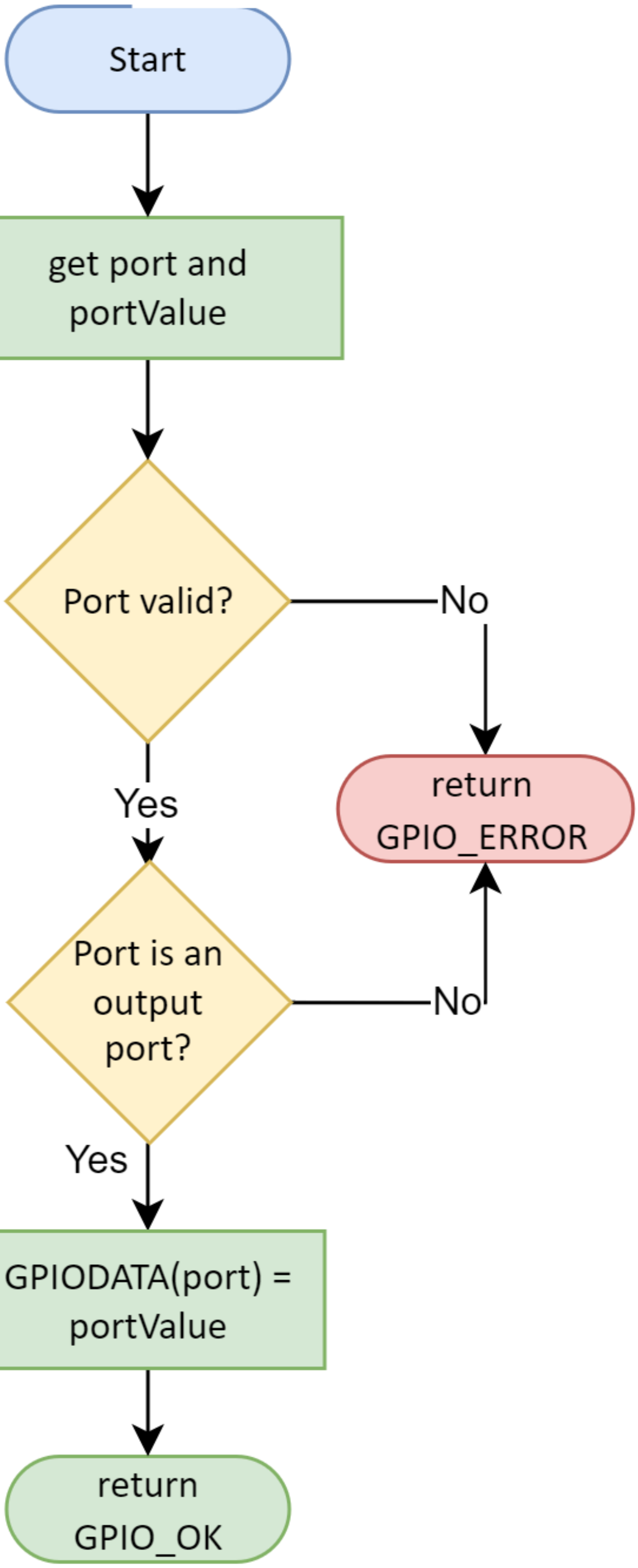
3.1.1.3. GPIO_getPinVal



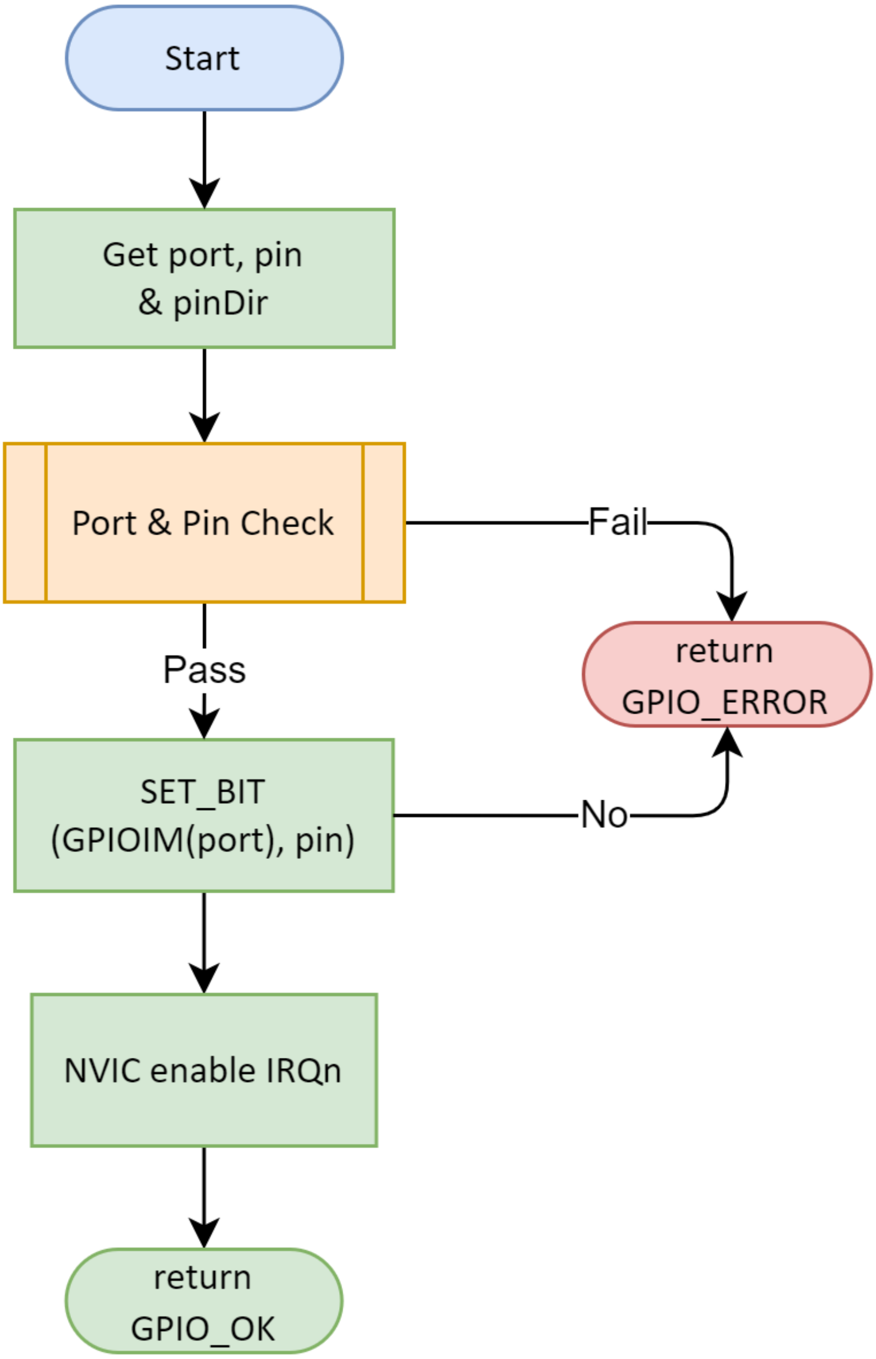
3.1.1.4. GPIO_togPinVal



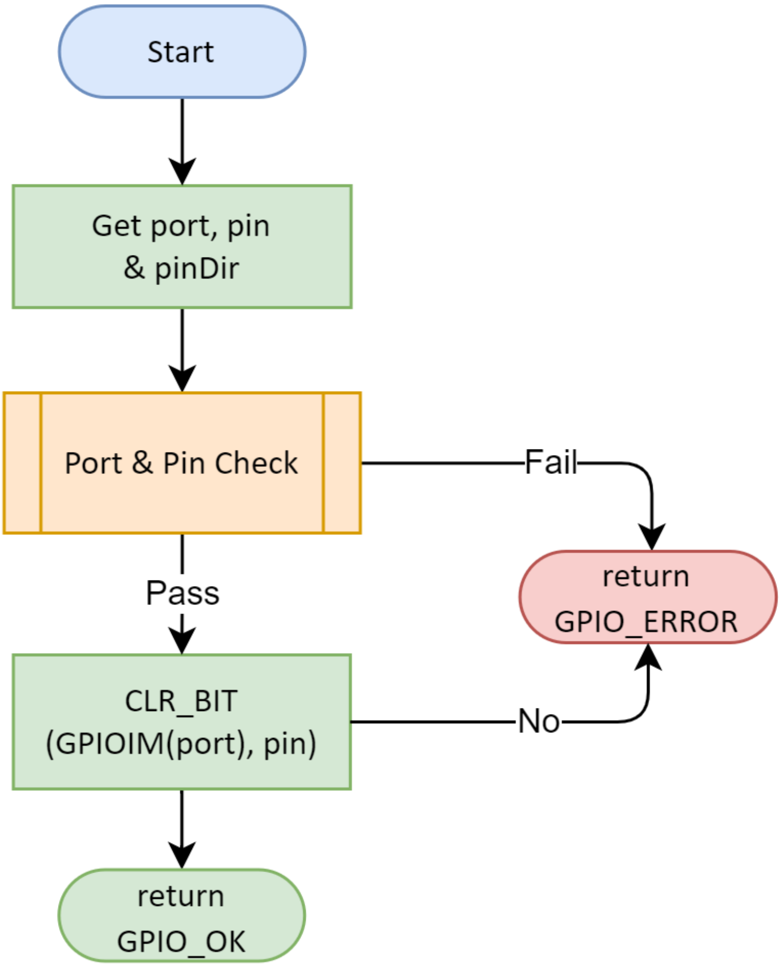
3.1.1.5. GPIO_setPortVal



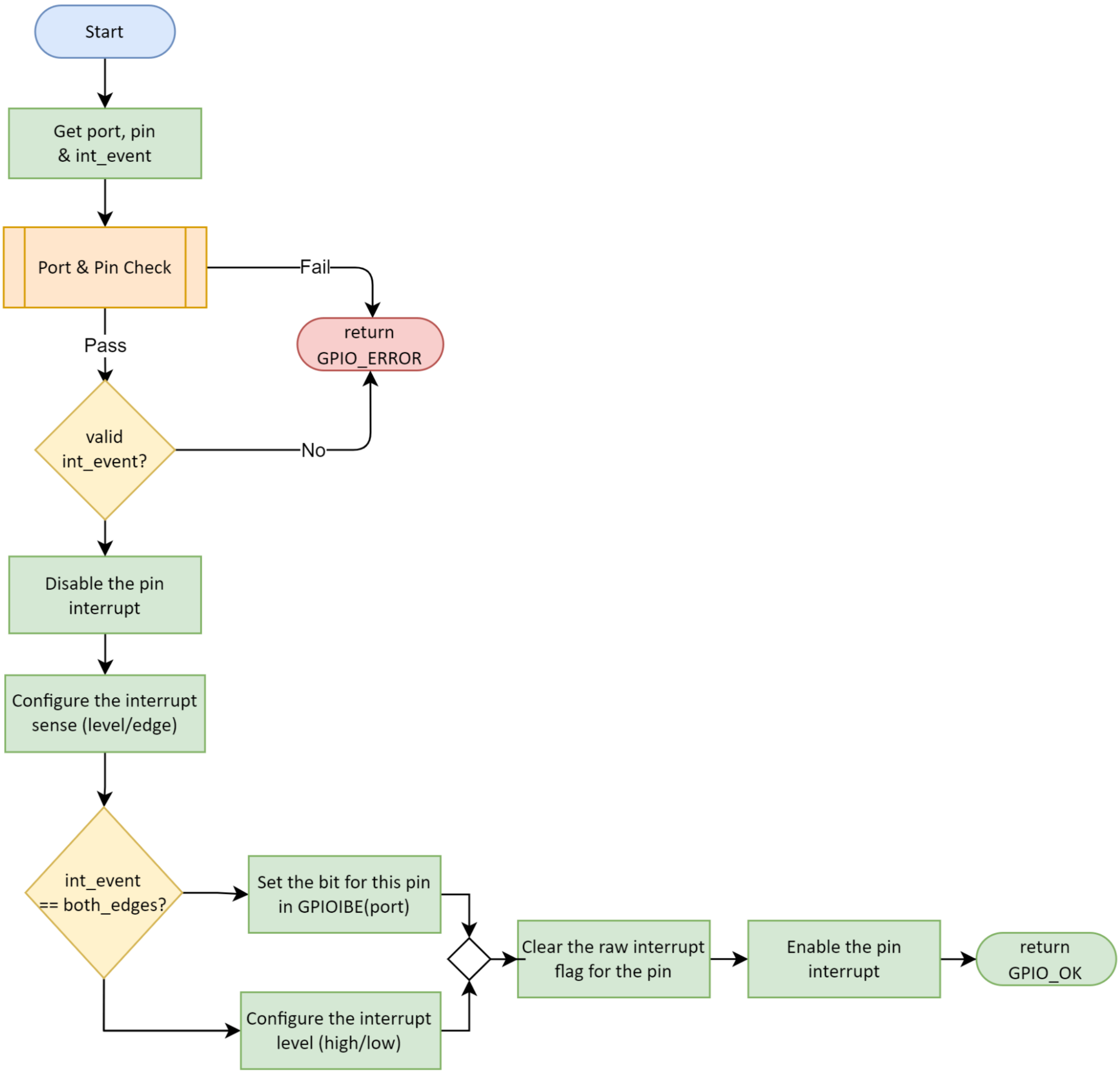
3.1.1.6. GPIO_enableInt



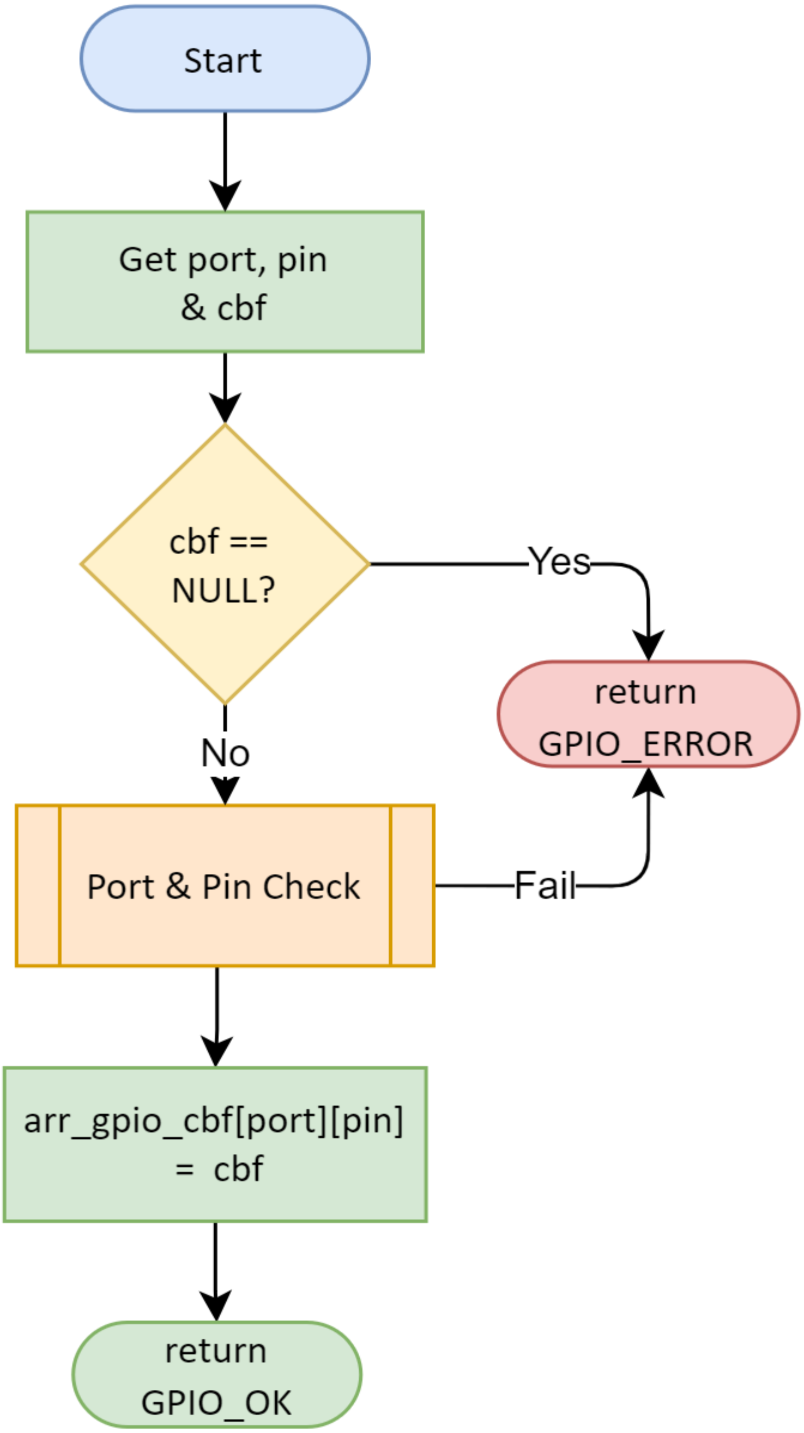
3.1.1.7. GPIO_disableInt



3.1.1.8. GPIO_setIntSense

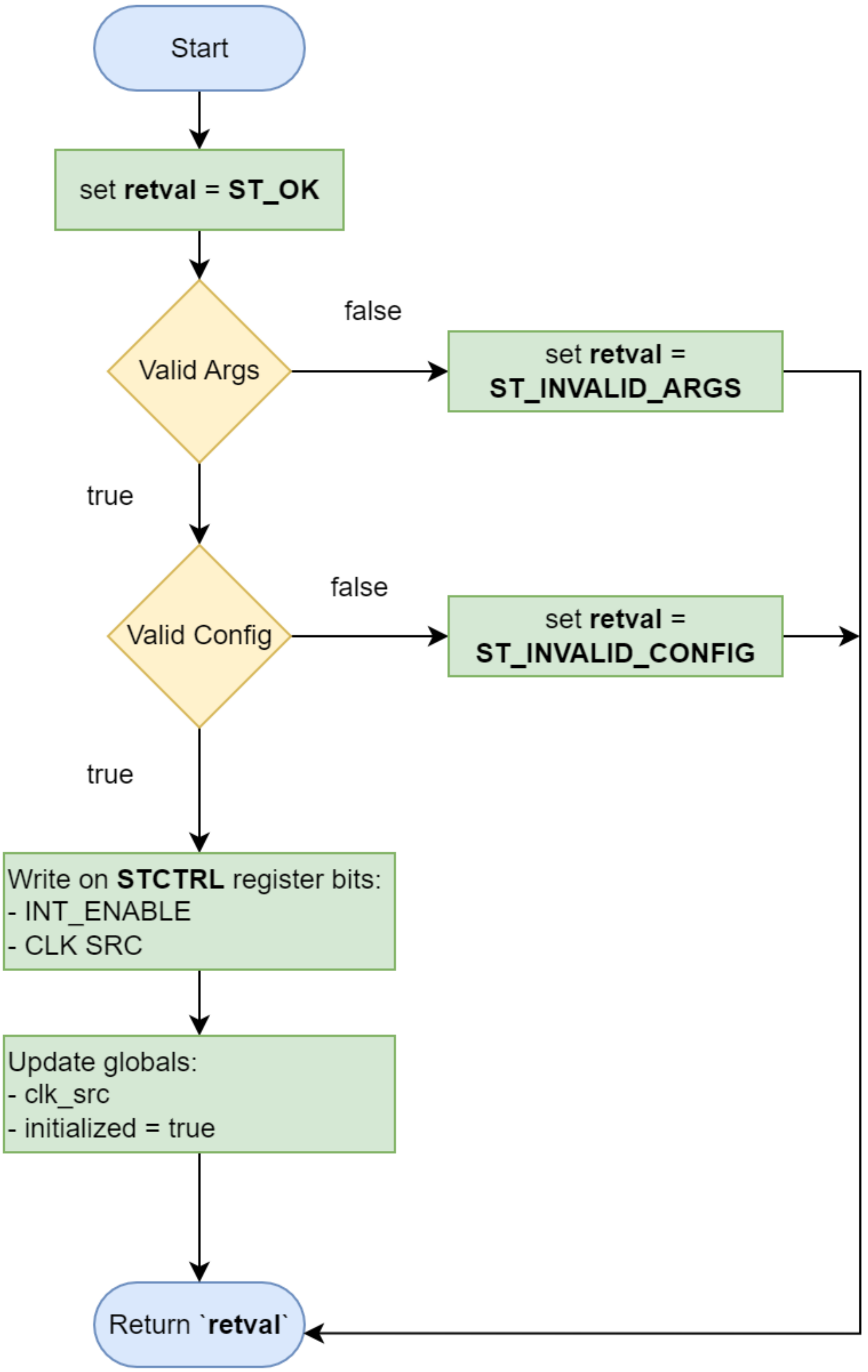

Start
Get port, pin & int_event
Port & Pin Check
Fail
return GPIO_ERROR
Pass
valid int_event?
No
Disable the pin interrupt
Configure the interrupt sense (level/edge)
int_event == both_edges?
Set the bit for this pin in GPIOIBE(port)
Configure the interrupt level (high/low)
Clear the raw interrupt flag for the pin
Enable the pin interrupt
return GPIO_OK

3.1.1.9. GPIO_setIntCallback

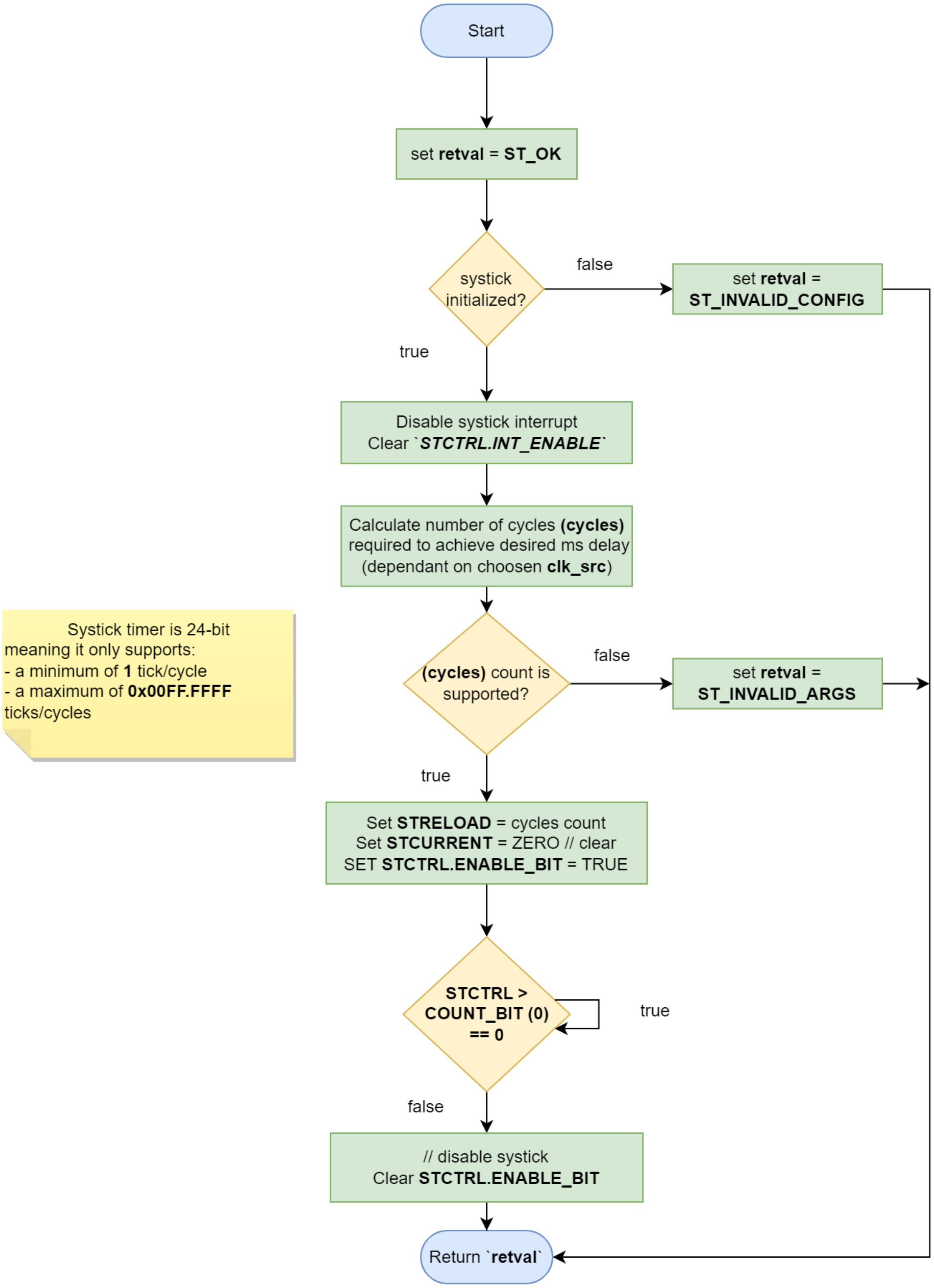


3.1.2. SYSTICK Module

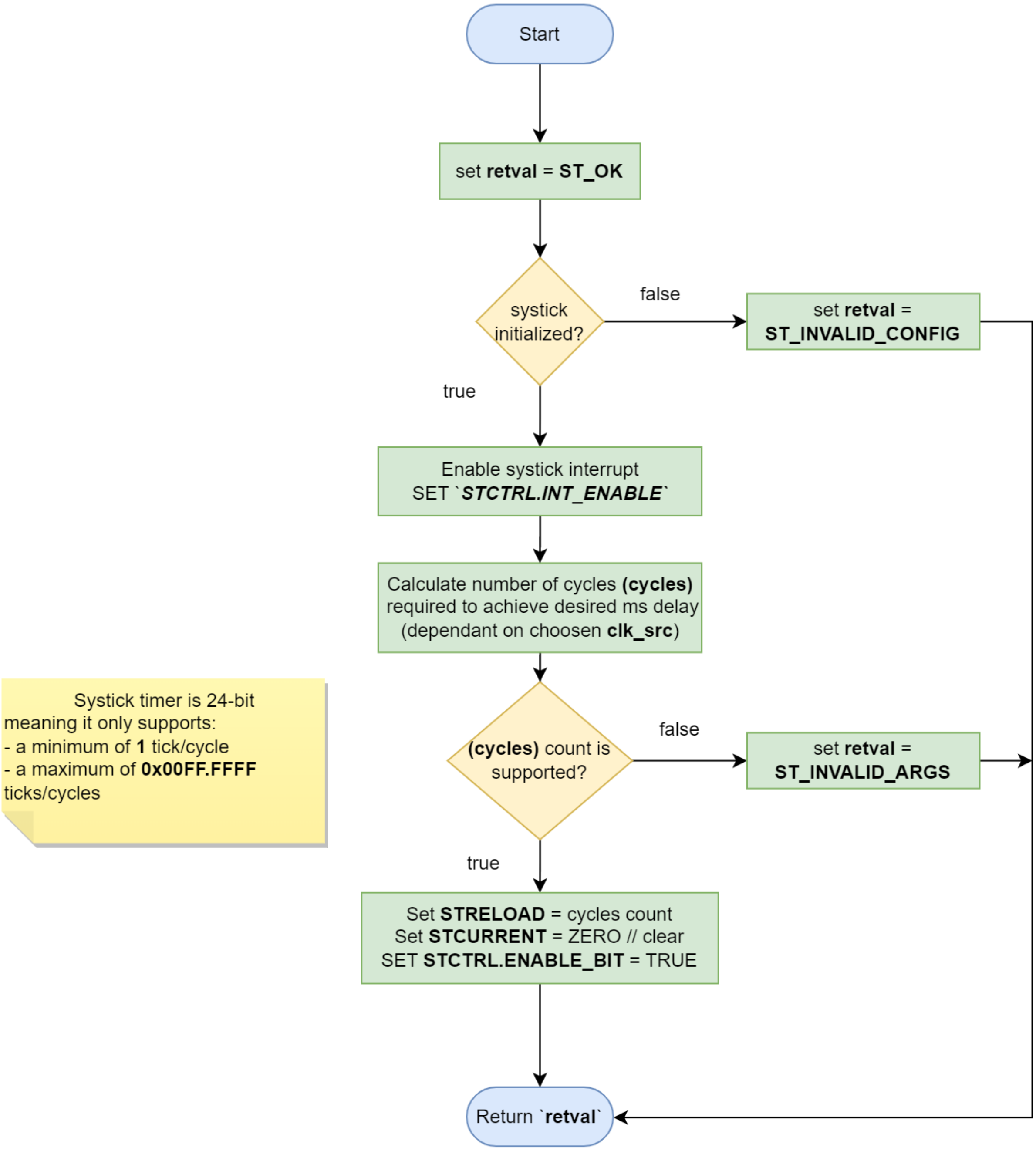
3.1.2.1. systick_init


Start
set **retval** = **ST_OK**
Valid Args
false
set **retval** = **ST_INVALID_ARGS**
true
Valid Config
false
set **retval** = **ST_INVALID_CONFIG**
true
Write on **STCTRL** register bits:
- INT_ENABLE
- CLK SRC
Update globals:
- clk_src
- initialized = true
Return `**retval**`

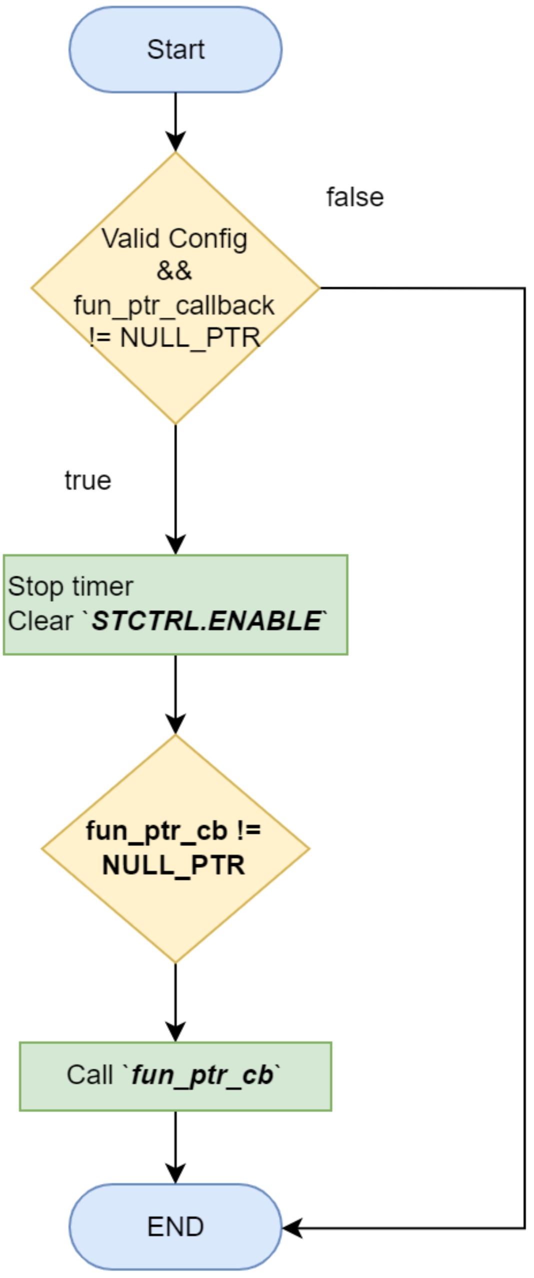
3.1.2.2. systick_ms_delay


Start
set retval = ST_OK
systick initialized?
false
set retval = ST_INVALID_CONFIG
true
Disable systick interrupt
Clear `STCTRL.INT_ENABLE`
Calculate number of cycles (cycles) required to achieve desired ms delay (dependant on choosen clk_src)
Systick timer is 24-bit meaning it only supports:
- a minimum of 1 tick/cycle
- a maximum of 0x00FF.FFFF ticks/cycles
(cycles) count is supported?
false
set retval = ST_INVALID_ARGS
true
Set STRELOAD = cycles count
Set STCURRENT = ZERO // clear
SET STCTRL.ENABLE_BIT = TRUE
STCTRL > COUNT_BIT (0) == 0
true
false
// disable systick
Clear STCTRL.ENABLE_BIT
Return `retval`

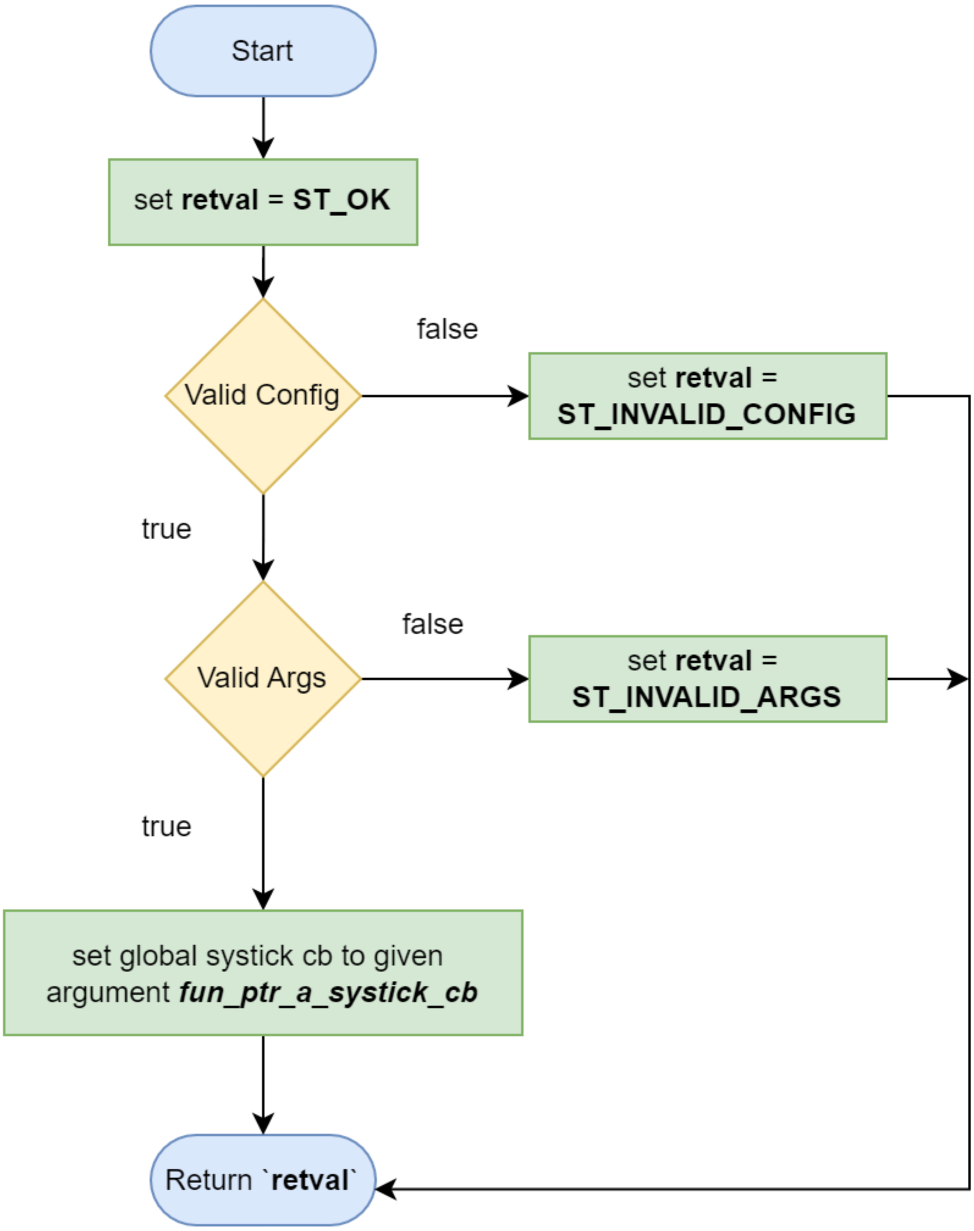
3.1.2.3. systick_async_ms_delay


Start
set **retval** = **ST_OK**
systick initialized?
false
set **retval** = **ST_INVALID_CONFIG**
true
Enable systick interrupt
SET `*STCTRL.INT_ENABLE*`
Calculate number of cycles **(cycles)** required to achieve desired ms delay (dependant on choosen **clk_src**)
Systick timer is 24-bit meaning it only supports:
- a minimum of **1** tick/cycle
- a maximum of **0x00FF.FFFF** ticks/cycles
(cycles) count is supported?
false
set **retval** = **ST_INVALID_ARGS**
true
Set **STRELOAD** = cycles count
Set **STCURRENT** = ZERO // clear
SET **STCTRL.ENABLE_BIT** = TRUE
Return `**retval**`

3.1.2.4. sysTick_Handler



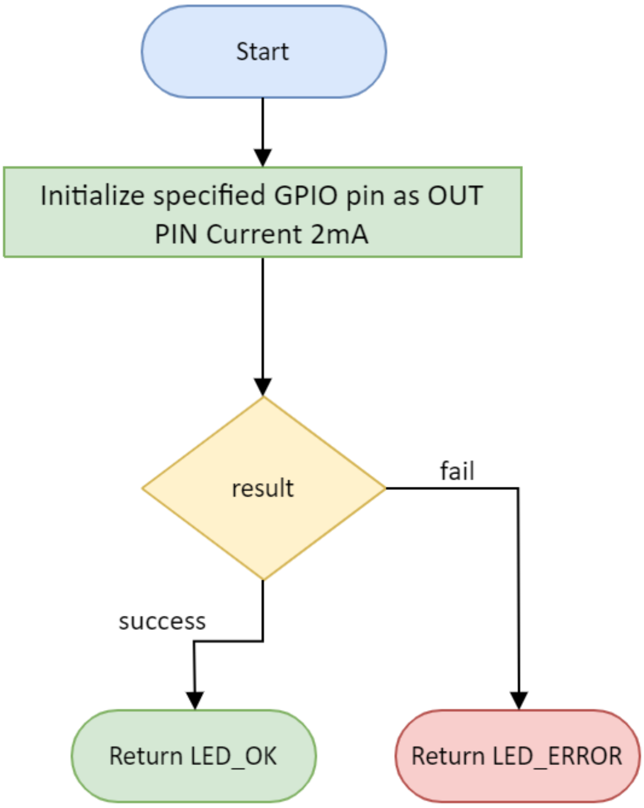
3.1.2.5. systick_set_callback


Start
set retval = ST_OK
Valid Config
false
set retval = ST_INVALID_CONFIG
true
Valid Args
false
set retval = ST_INVALID_ARGS
true
set global systick cb to given argument fun_ptr_a_systick_cb
Return `retval`

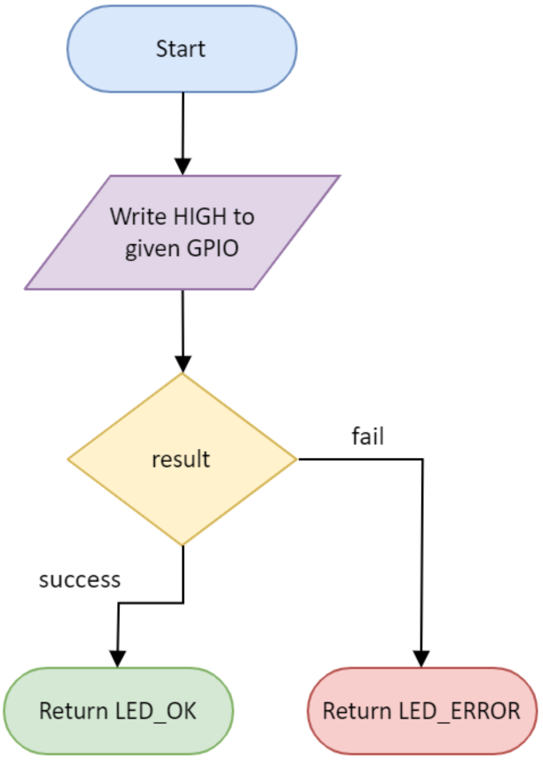
3.2. HAL Layer

3.2.1. LED Module

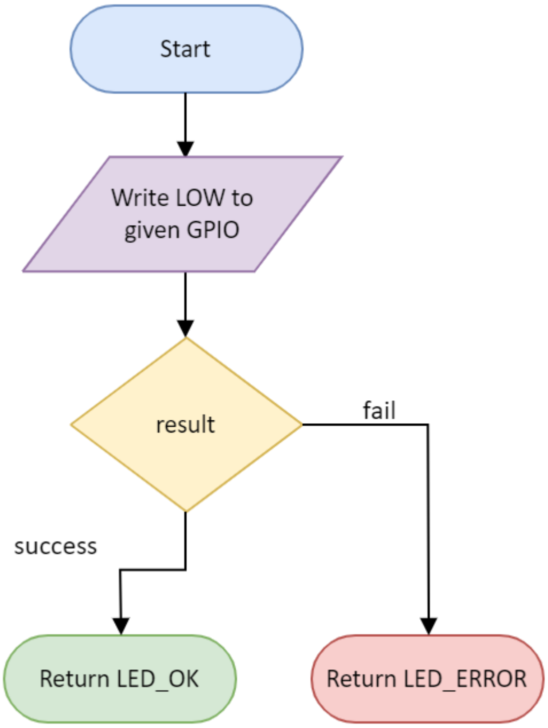
3.2.1.1. led_init



3.2.1.2. led_on

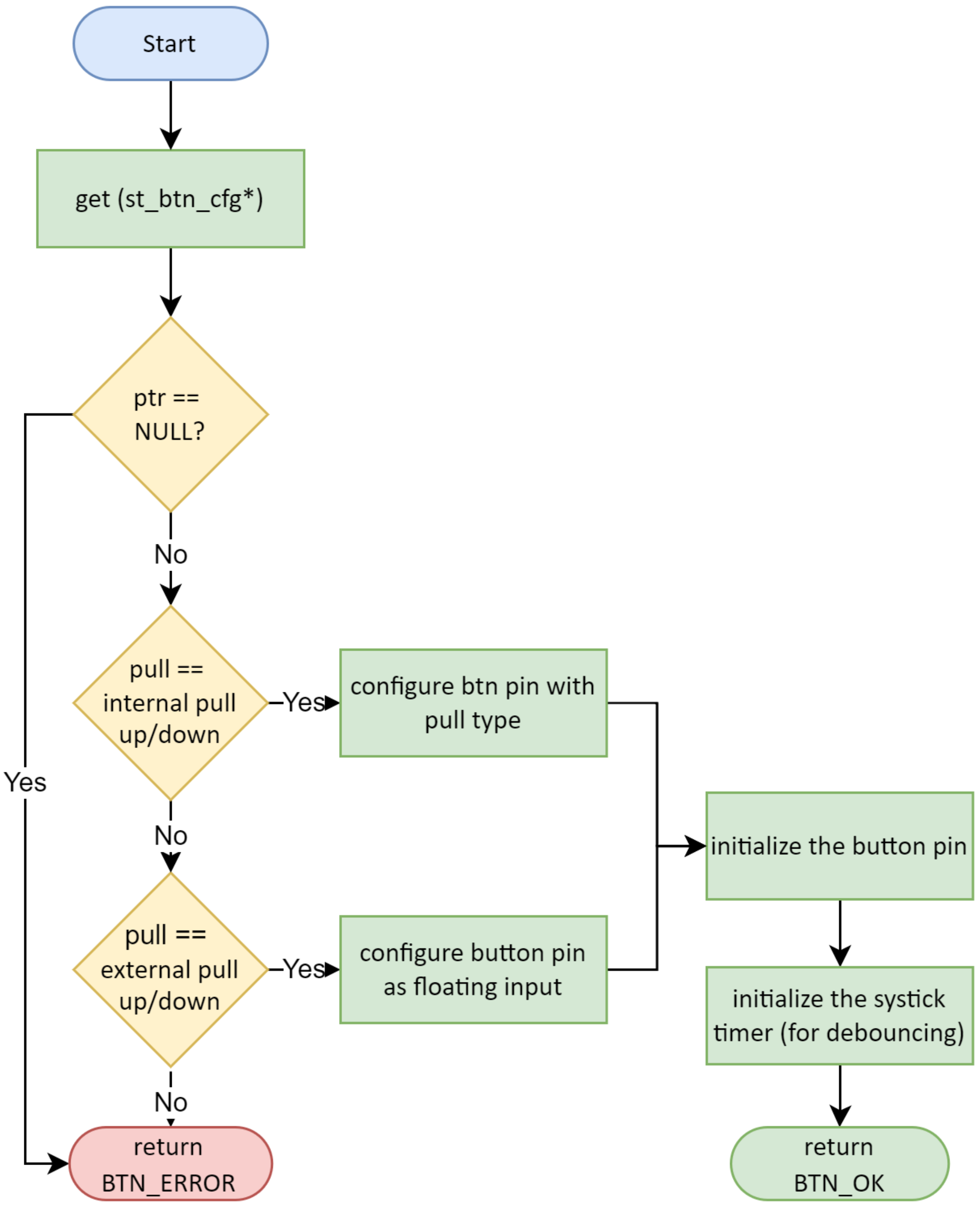


3.2.1.3. led_off

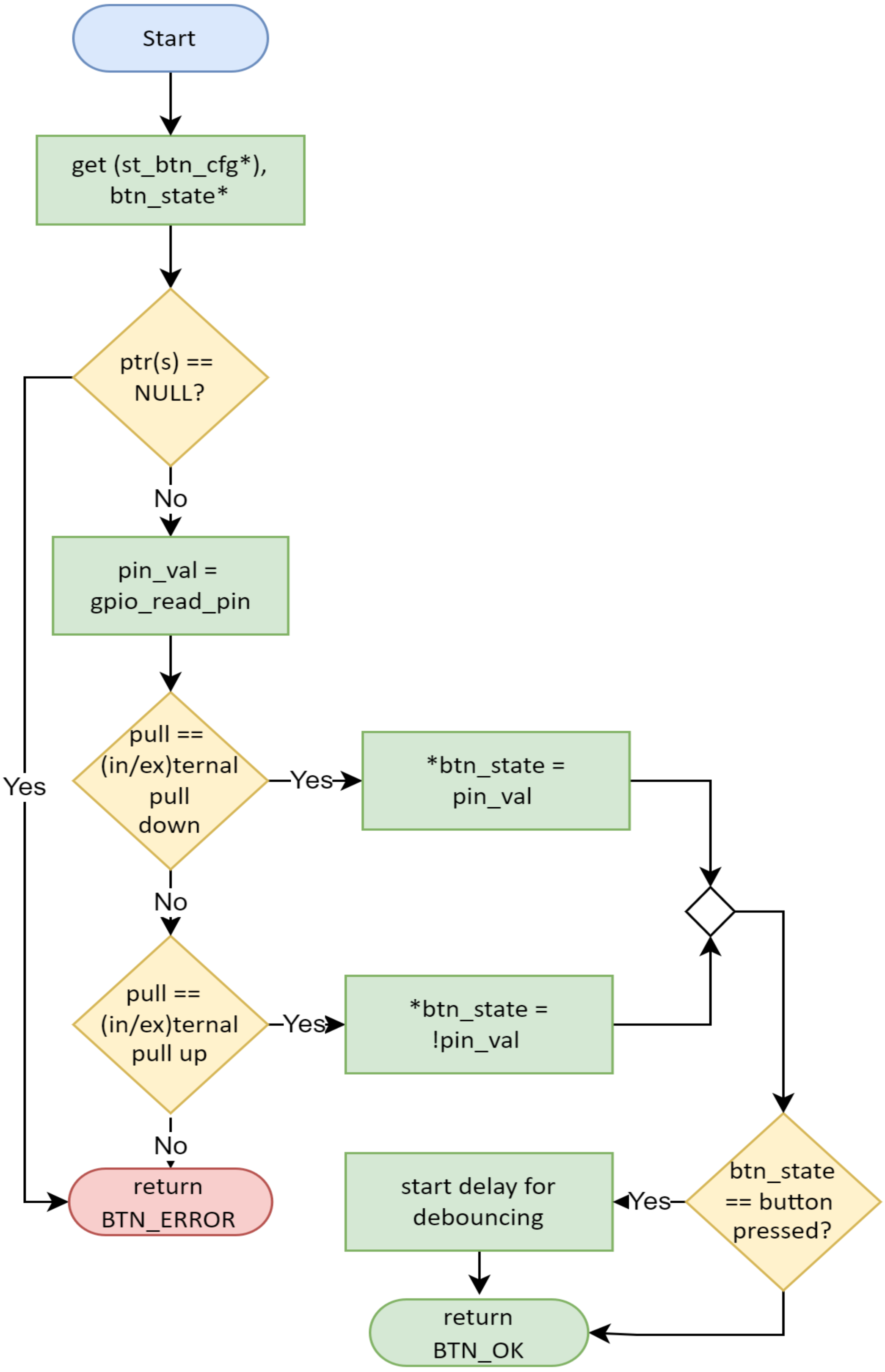


3.2.2. BTN Module

3.2.2.1. BUTTON_init

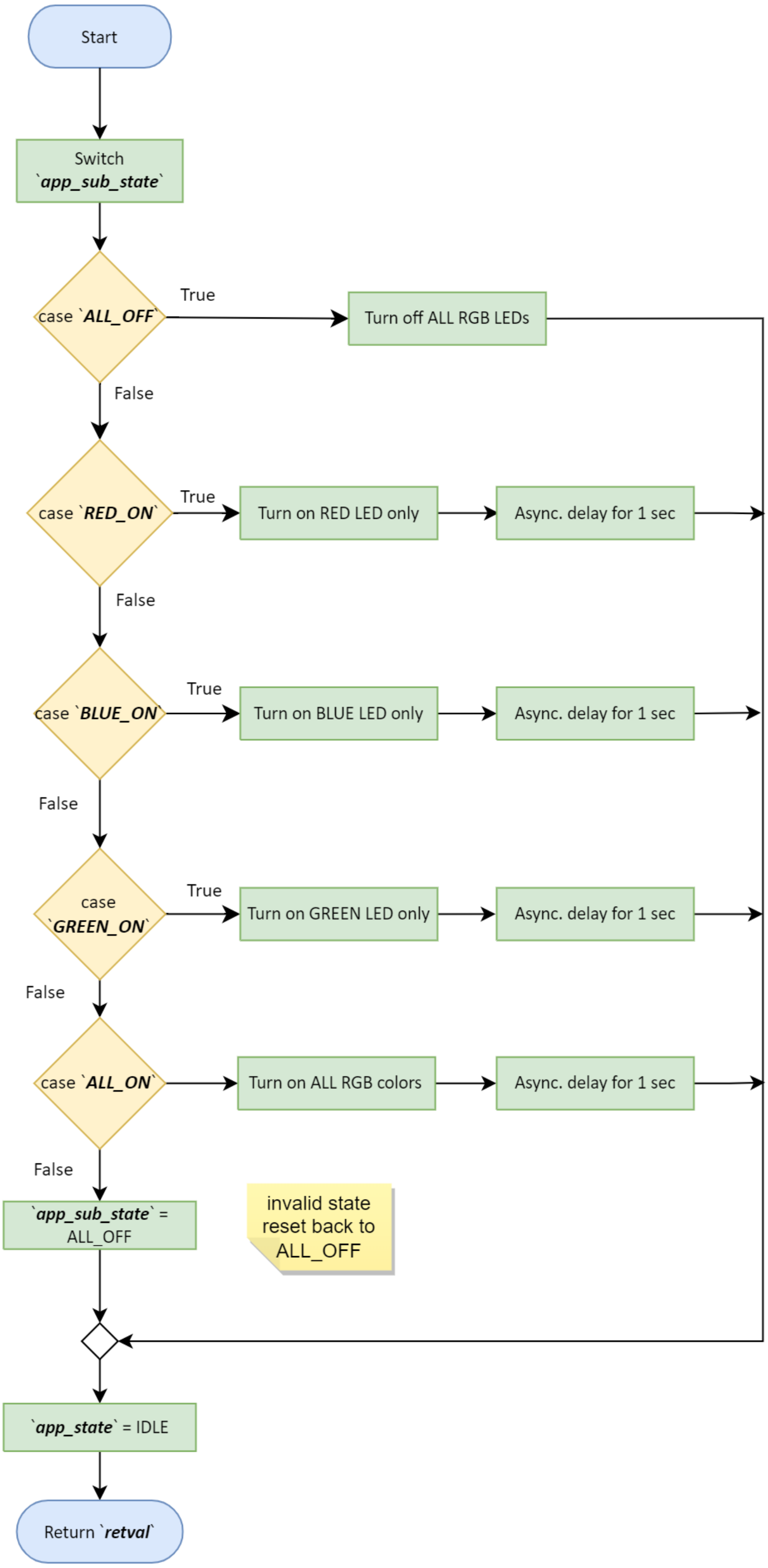

Start
get (st_btn_cfg*)
ptr == NULL?
No
Yes
pull == internal pull up/down
Yes
configure btn pin with pull type
No
pull == external pull up/down
Yes
configure button pin as floating input
No
return BTN_ERROR
initialize the button pin
initialize the systick timer (for debouncing)
return BTN_OK

3.2.2.2. BUTTON_read

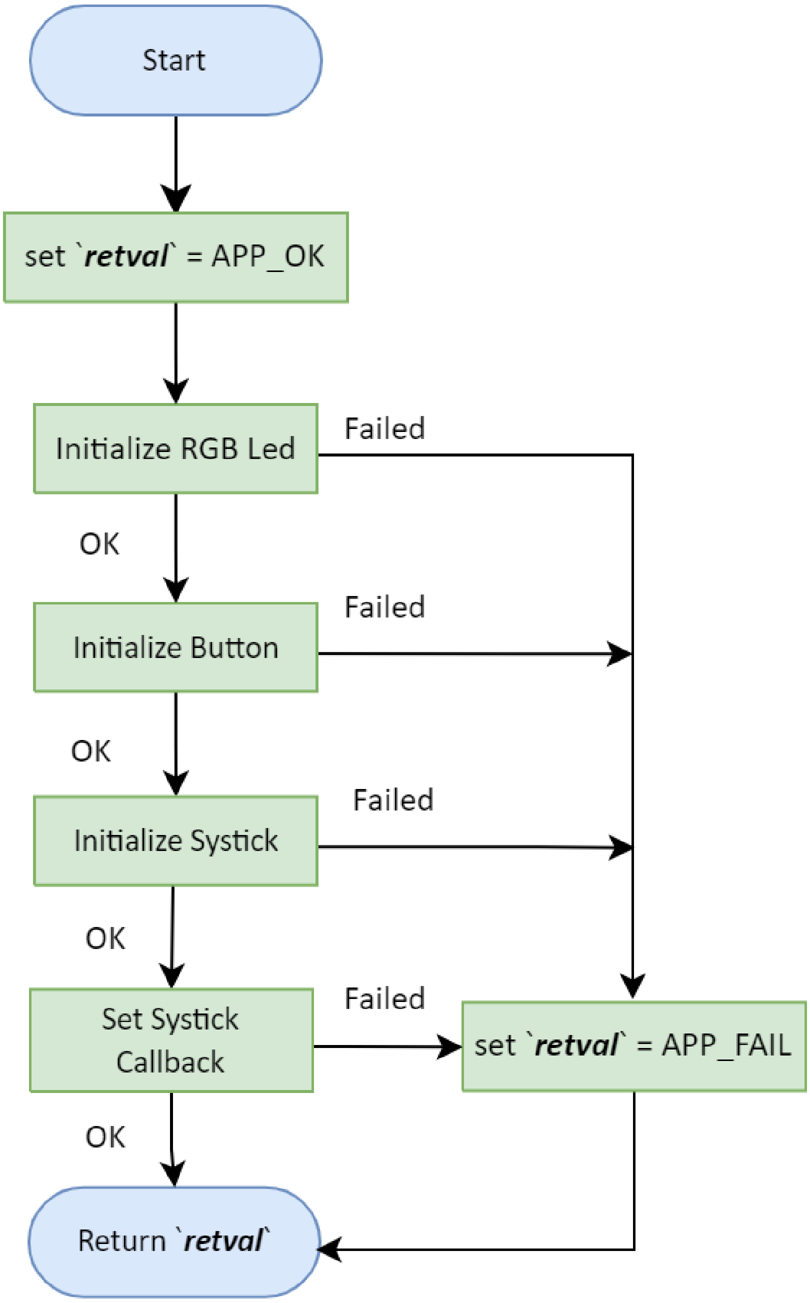


3.3. APP Layer

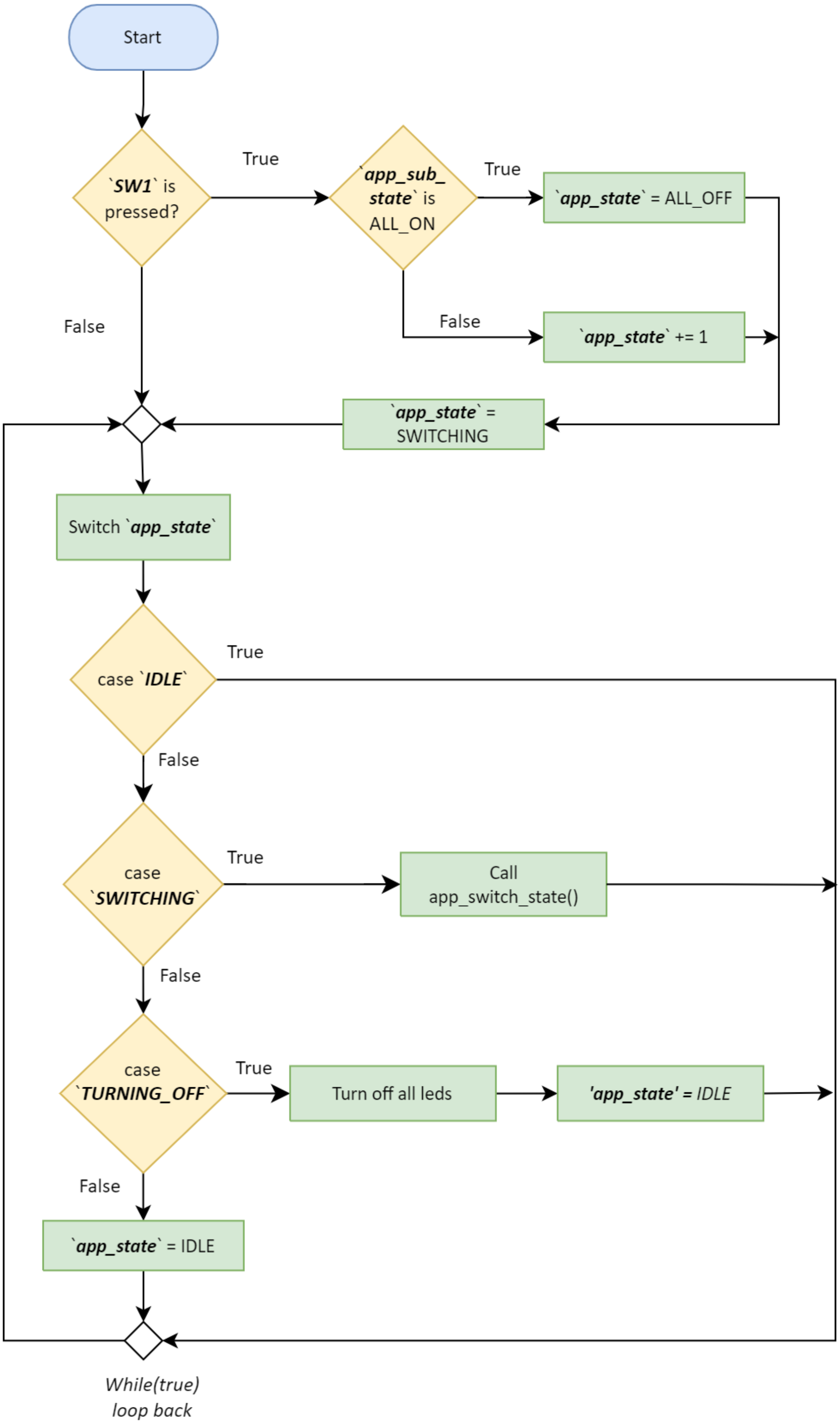
3.3.1. App_switch_state



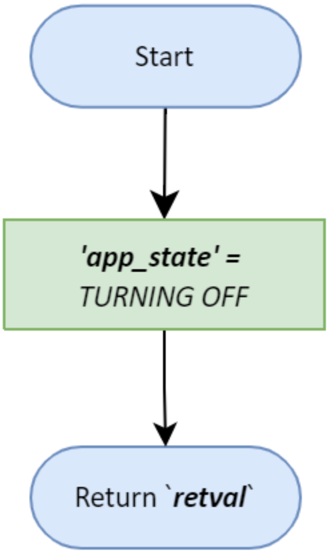
3.3.2. app_init


Start
set `retval` = APP_OK
Initialize RGB Led
Failed
OK
Initialize Button
Failed
OK
Initialize Systick
Failed
OK
Set Systick Callback
Failed
set `retval` = APP_FAIL
OK
Return `retval`

3.3.3. app_start



3.3.4. app_systick_cb



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
```

4.2.2. Linking configuration

```
st_systick_cfg_t gl_st_systick_cfg_0 =
{
        .en_systick_clk_src = CLK_SRC_PIOSC,
        .fun_ptr_systick_cb = NULL_PTR
};
```

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